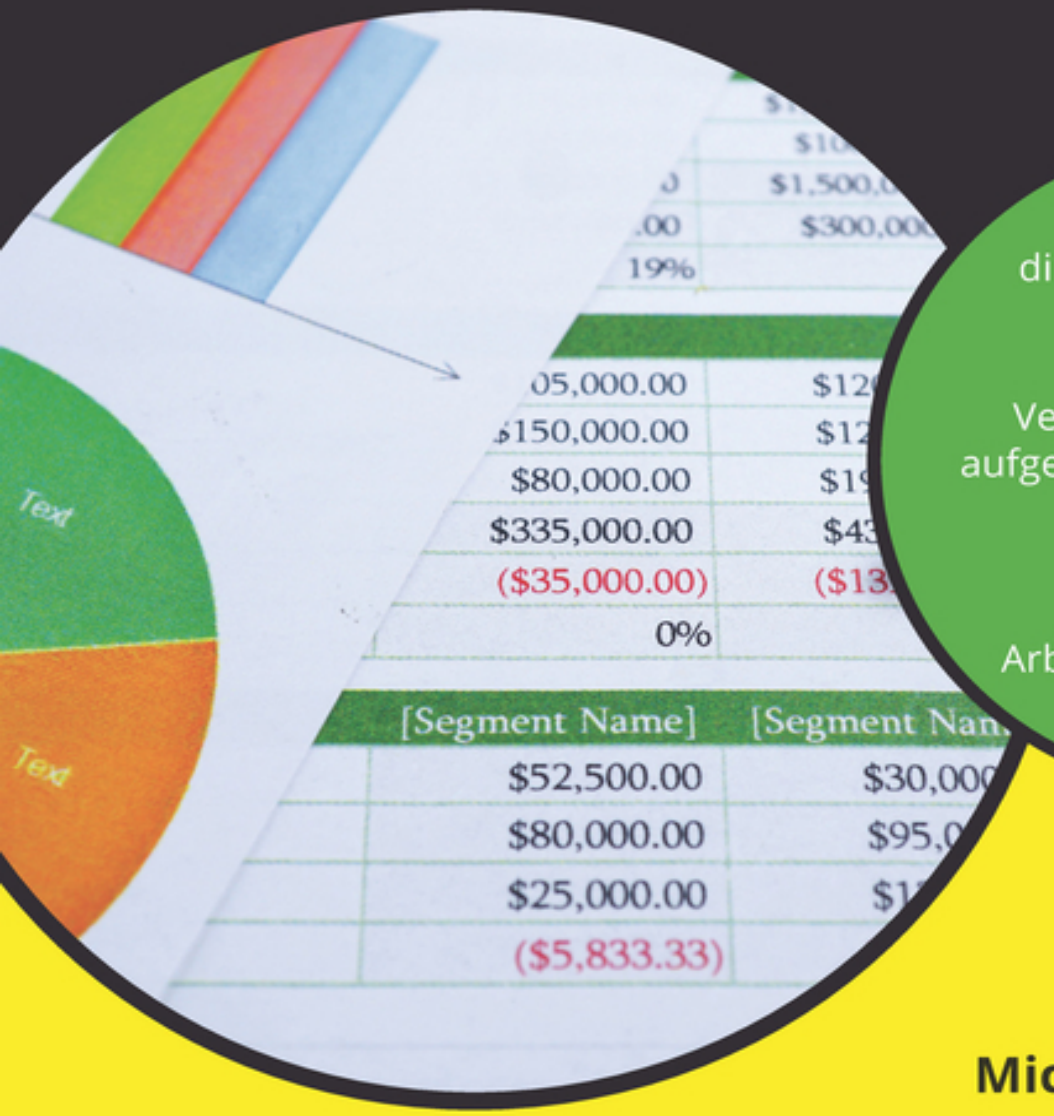
LERNEN EINFACH GEMACHT

2. Auflage

Excel Makros

für dummies®



Über 100 Makros direkt nutzen und Zeit sparen

Verstehen, wie Makros aufgebaut sind und wie man sie anpasst

Makros anderer Arbeitsmappen effizient einsetzen

Michael Alexander

Excel Makros für Dummies

Schummelseite

Dank der Excel-Tastenkombinationen können Sie bestimmte Aktionen direkt mit der Tastatur auslösen. Sie können so schneller arbeiten, da Sie seltener zur Maus greifen müssen, um einen bestimmten Befehl auszuwählen. Wenn Sie sich angewöhnen, diese Tastenkombinationen zu verwenden, ist Ihre Arbeit auch im Visual-Basic-Editor effizienter.

STANDARDTASTENKOMBINATIONEN FÜR DEN VISUAL-BASIC-EDITOR

Mit den folgenden Tastenkombinationen können Sie schnell im Visual-Basic-Editor navigieren.

Tastenkombination	*Aktion*
Alt + F11	zwischen Excel-Fenster und Visual-Basic-Editor wechseln
⇧ + F10	Kontextmenü des aktiven Fensters anzeigen (entspricht einem Rechtsklick mit der Maus)
Strg + R	Projekt-Explorer öffnen
F4	Eigenschaftenfenster öffnen
F2	Objekt-Browser öffnen
F1	VBA-Hilfe öffnen
F7	geöffnetes Modul-Fenster aktivieren

TASTENKOMBINATIONEN FÜR DAS BEARBEITEN VON CODE IM CODEFENSTER

Früher oder später werden Sie zur gleichen Zeit an mehreren Excel-Makros arbeiten. Mit der Maus von einem zum anderen und innerhalb einer Makroprozedur zu navigieren, kann etwas umständlich sein. Die folgenden Tastenkombinationen ermöglichen es, schnell zu einer Prozedur zu springen, sich in den Modulen zu bewegen und sogar die Stelle zu finden, an der eine bestimmte Variable deklariert wurde.

Tastenkombination	*Aktion*
Strg + ↓	nächste Prozedur auswählen
Strg + ↑	vorige Prozedur auswählen
Strg + Bild ⇩	Bildschirminhalt um eine Seite nach unten bewegen
Strg + Bild ⇧	Bildschirminhalt um eine Seite nach oben bewegen
⇧ + F2	zur ausgewählten Funktion oder Variable gehen
Strg + ⇧ + F2	zur letzten Position zurückgehen
Strg + Pos1	zum Anfang des Moduls gehen
Strg + Ende	zum Ende des Moduls gehen
Strg + →	Einfügemarke wortweise nach rechts bewegen
Strg + ←	Einfügemarke wortweise nach links bewegen
Ende	Einfügemarke an das Ende der Zeile bewegen
Pos1	Einfügemarke an den Anfang der Zeile bewegen
⇤	aktuelle Zeile einrücken
⇧ + ⇤	Einrückung der aktuellen Zeile entfernen
Strg + J	Eigenschaften und Methoden des aktuellen Objekts anzeigen

TASTENKOMBINATIONEN FÜR DAS DEBUGGEN VON CODE IM VISUAL-BASIC-EDITOR

Das Debuggen Ihres Codes ist ein wichtiger Arbeitsschritt bei der Programmierung von Excel-Makros. Sie können auf die Debugging-Features zwar auch über das Menü des Visual-Basic-Editors zugreifen, jedoch kommen Sie mit den folgenden Tastenkombinationen schneller zum Ziel.

Tastenkombination	*Aktion*
F5	aktuelle Prozedur ausführen oder deren Ausführung fortsetzen, falls sie unterbrochen wurde
Strg + Untbr	Ausführung der aktuell laufenden Prozedur unterbrechen
F8	Debug-Modus aktivieren und Code zeilenweise ausführen
Strg + F8	Code bis zur aktuellen Cursorposition ausführen
⇧ + F8	im Debug-Modus die aktuelle Zeile überspringen
F9	Haltepunkt für die aktuelle Zeile ein- beziehungsweise ausschalten
Strg + ⇧ + F9	alle Haltepunkte entfernen
Alt + D + L	aktuelles Visual-Basic-Projekt kompilieren

Michael Alexander

Excel Makros für dummies®

2. Auflage

Übersetzung aus dem Amerikanischen von

Rainer G. Haselier

WILEY-VCH

WILEY-VCH GmbH

Excel Makros für Dummies

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

2. Auflage 2021

Coverfoto:© amazing studios / stock.adobe.com
Korrektur: Matthias Delbrück

Print ISBN: 978-3-527-71841-2
ePub ISBN: 978-3-527-83315-3

Inhaltsverzeichnis

Cover

Titelblatt

Impressum

Einleitung

- Über dieses Buch
- Törichte Annahmen über den Leser
- Icons in diesem Buch
- Die Beispieldateien zu diesem Buch
- Wie es weitergeht

Teil I: Heiliges Makro, Batman!

Kapitel 1: Makrogrundlagen

- Warum sollten Sie Makros verwenden?
- Grundlagen der Makroaufzeichnung
- Absolute und relative Makroaufzeichnung im Vergleich
- Weitere Konzepte für die Makroaufzeichnung
- Makros im Einsatz – Beispiele

Kapitel 2: Machen Sie sich mit dem Visual-Basic-Editor vertraut

- Im Visual-Basic-Editor arbeiten
- Mit dem Projekt-Explorer arbeiten
- Mit dem Codefenster arbeiten
- Die VBA-Entwicklungsumgebung anpassen

Kapitel 3: Die Anatomie von Makros

- Ein kurzer Überblick über das Excel-Objektmodell
- Ein kurzer Blick auf Variablen
- Ereignisprozeduren verstehen
- Fehlerbehandlung kurz und bündig

Teil II: Aufgaben in Arbeitsmappen mit wenig Arbeit erledigen

Kapitel 4: Makros für Arbeitsmappen

- Eine neue Arbeitsmappe aus dem Nichts erstellen
- Eine Arbeitsmappe speichern, wenn eine bestimmte Zelle geändert wird
- Eine Arbeitsmappe vor dem Schließen speichern
- Ein Tabellenblatt vor dem Schließen schützen
- Beim Öffnen einer Arbeitsmappe den Blattschutz aufheben
- Beim Öffnen einer Arbeitsmappe ein bestimmtes Tabellenblatt anzeigen
- Eine vom Benutzer ausgewählte Arbeitsmappe öffnen
- Feststellen, ob eine Arbeitsmappe bereits geöffnet ist
- Überprüfen, ob eine Arbeitsmappe in einem Ordner vorhanden ist
- Alle Arbeitsmappen auf einmal schließen
- Alle Arbeitsmappen in einem Ordner drucken
- Arbeitsmappe erst dann schließen, wenn eine bestimmte Zelle Inhalte besitzt
- Backup der aktuellen Arbeitsmappe mit aktuellem Datum erstellen

Kapitel 5: Makros für Tabellenblätter

- Ein neues Tabellenblatt einfügen und benennen
- Alle Tabellenblätter bis auf das aktive löschen
- Alle Tabellenblätter bis auf das aktive ausblenden
- Alle Tabellenblätter einer Arbeitsmappe einblenden
- Tabellenblätter verschieben
- Tabellenblätter nach Namen sortieren
- Tabellenblätter nach Farben gruppieren
- Ein Tabellenblatt in eine neue Arbeitsmappe kopieren
- Eine neue Arbeitsmappe für jedes Tabellenblatt erstellen
- Bestimmte Tabellenblätter drucken
- Alle Tabellenblätter schützen

Den Blattschutz aller Tabellenblätter aufheben
Ein Inhaltsverzeichnis Ihrer Arbeitsmappe erstellen
Mit einem Doppelklick in einem Tabellenblatt rein- und rauszoomen
Die aktive Zeile und Spalte hervorheben
Teil III: Datenbearbeitung mit einem Klick
Kapitel 6: Makros für Zellen und Zellbereiche
Einen Bereich auswählen und formatieren
Benannte Bereiche erstellen und auswählen
Alle Zellen eines Bereichs in einer Schleife durchlaufen
In einen Bereich leere Zeilen einfügen
Alle Zeilen und Spalten einblenden
Leere Zeilen löschen
Leere Spalten löschen
Den Scrollbereich einschränken
Alle Formeln in einer Arbeitsmappe auswählen und formatieren
Die erste leere Zeile oder Spalte finden und auswählen
Kapitel 7: Makros für das Bearbeiten von Zelldaten
Einen Zellbereich kopieren und einfügen
Alle Formeln eines Zellbereichs in Werte konvertieren
Den Befehl »Text in Spalten« auf alle Spalten anwenden
Nachstehende Minuszeichen konvertieren
Führende und nachstehende Leerzeichen aller Zellen in einem Bereich entfernen
US-Postleitzahlen auf die ersten fünf Stellen kürzen
Zellen mit führenden Nullen versehen
Leere Zellen durch einen Wert ersetzen
Vor oder nach dem Zellinhalt Text einfügen
Nicht druckbare Zeichen entfernen
Duplikate in einem Datenbereich hervorheben
Alle Zeilen bis auf Zeilen mit Duplikaten ausblenden

Festlegen, in welchen Spalten AutoFilter eingesetzt werden darf

Gefilterte Zeilen in eine neue Arbeitsmappe kopieren

In der Statusleiste die Spalten anzeigen, nach denen gefiltert wird

Teil IV: Berichte, E-Mails und externe Datenquellen mit Makros steuern

Kapitel 8: Die Erstellung von Berichten automatisieren

Alle Pivot-Tabellen einer Arbeitsmappe aktualisieren

Eine Liste mit Informationen zu den Pivot-Tabellen einer Arbeitsmappe erstellen

Die Titel aller Datenfelder der Pivot-Tabelle anpassen

Für alle Datenelemente Summe verwenden

Alle Datenelemente als Zahlen formatieren

Alle Datenfelder alphabetisch sortieren

Datenelemente benutzerdefiniert sortieren

Einschränkungen für Pivot-Tabellen festlegen

Einschränkungen für PivotTable-Felder festlegen

Drilldown-Tabellenblätter automatisch entfernen

Eine Pivot-Tabelle für jedes Element des Berichtsfilters drucken

Für jedes Element des Berichtsfilters eine neue Arbeitsmappe erstellen

Die Größe aller Diagramme eines Tabellenblatts ändern

Diagramm platzieren

Verknüpfung zwischen Diagramm und Daten lösen

Alle Diagramme eines Tabellenblatts drucken

Kapitel 9: E-Mails mit Excel versenden

Die aktive Arbeitsmappe als E-Mail-Anhang senden

Einen bestimmten Zellbereich als E-Mail-Anhang senden

Ein bestimmtes Tabellenblatt als E-Mail-Anhang senden

Mail mit einem Link zu Ihrer Arbeitsmappe senden

E-Mail an alle Adressen in einer Liste mit Kontakten senden
Alle E-Mail-Anhänge in einem Ordner speichern
Bestimmte E-Mail-Anhänge in einem Ordner speichern
Kapitel 10: Externe Daten mit Makros bearbeiten
Externe Datenverbindungen verwenden
Mit Makros dynamische Verbindungen erstellen
Alle Verbindungen in einer Arbeitsmappe in einer Schleife bearbeiten
ADO und VBA verwenden, um externe Daten abzurufen
ADO in einen Makro verwenden
Mit Textdateien arbeiten
Teil V: Der Top-Ten-Teil
Kapitel 11: Zehn nützliche Tipps für den Visual-Basic-Editor
Kommentarblöcke verwenden
Mehrere Codezeilen auf einen Rutsch kopieren
Zwischen Modulen und Prozeduren springen
Beamen Sie sich zu Ihren Funktionen
In der richtigen Prozedur bleiben
Den Code schrittweise ausführen
Zu einer bestimmten Zeile in Ihrem Code springen
Die Ausführung Ihres Codes an einer bestimmten Stelle unterbrechen
Den Anfang und das Ende von Variablenwerten anzeigen
Die automatische Syntaxüberprüfung ausschalten
Kapitel 12: Zehn Orte, wo Sie Hilfe bei der Makro-Programmierung erhalten
Lassen Sie Excel die Makros für Sie schreiben
Verwenden Sie die VBA-Hilfedateien
Stibitzen Sie Code im Internet
Userforen optimal nutzen
Besuchen Sie Experten-Blogs

Suchen Sie auf YouTube nach Schulungsvideos
Nehmen Sie an Online-Kursen teil
Vom Office Dev Center lernen
Analysieren Sie andere Excel-Dateien in Ihrem Unternehmen
Fragen Sie Ihre Excel-Experten vor Ort

Kapitel 13: Zehn Methoden zur Beschleunigung von Makros

Automatische Neuberechnung abschalten
Die Bildschirmaktualisierung deaktivieren
Aktualisierung der Statusleiste abschalten
Excel anweisen, Ereignisse zu ignorieren
Seitenwechsel ausblenden
Die Aktualisierung von Pivot-Tabellen unterdrücken
Kopieren und Einfügen vermeiden
Die With-Anweisung verwenden
Die Methode Select vermeiden
Zugriffe auf das Tabellenblatt optimieren

Stichwortverzeichnis

End User License Agreement

Illustrationsverzeichnis

Kapitel 1

Abbildung 1.1: Das Dialogfeld »Makro aufzeichnen«
Abbildung 1.2: Dieses Tabellenblatt enthält zwei nebeneinander steh...
Abbildung 1.3: Das Tabellenblatt mit der neuen Zeile »Anzahl«
Abbildung 1.4: Das Excel-Dialogfeld »Makro«
Abbildung 1.5: Aufzeichnung eines Makros mit relativen Verweisen
Abbildung 1.6: Fügen Sie hier Speicherorte ein, die als vertrauensw...

Abbildung 1.7: Sie finden die Formularsteuerelemente auf der Registerkarte »Entwi...

Abbildung 1.8: Der neu hinzugefügten Schaltfläche ein Makro zuweisen

Abbildung 1.9: Ein Makro in die Symbolleiste für den Schnellzugriff einfügen

Abbildung 1.10: Verwenden Sie Makros, um Schaltflächen zu erstellen, mit denen di...

Abbildung 1.11: In diesem Bericht können die Benutzer die Perspektive des Diagram...

Abbildung 1.12: Die Makros hinter diesen Schaltflächen ordnen die Datenfelder der...

Abbildung 1.13: Vorher aufgezeichnete Ansichten zur Verfügung zu stellen, führt n...

Abbildung 1.14: Sie können Ihren Anwendern die Entscheidung darüber überlassen, w...

Kapitel 2

Abbildung 2.1: Die wichtigsten Elemente der Entwicklungsumgebung Visual-Basic-Edi...

Abbildung 2.2: Der Projekt-Explorer zeigt zwei Projekte an. Beide Projekte sind e...

Abbildung 2.3: Code-Module sind im Knoten »Module« des Projekt-Explorers sichtbar...

Abbildung 2.4: Die Registerkarte »Editor« des Dialogfelds »Optionen«

Abbildung 2.5: So legen Sie das Aussehen von VBE auf der Registerkarte »Editorfor...

Abbildung 2.6: Die Registerkarte »Allgemein« des Dialogfelds »Optionen«

Abbildung 2.7: Die Registerkarte »Verankern« des Dialogfelds »Optionen«

Kapitel 3

Abbildung 3.1: Das integrierte Code-Modul eines Tabellenblatts öffnen

Abbildung 3.2: Das Standardereignis »SelectionChange«

Abbildung 3.3: Wählen Sie das passende Ereignis aus.

Abbildung 3.4: Das integrierte Modul für »eine Arbeitsmappe öffnen«

Abbildung 3.5: Das Standardereignis »Open« für Arbeitsmappen

Abbildung 3.6: Öffnen Sie die Drop-down-Liste »Ereignisse« und wählen Sie das pas...

Kapitel 4

Abbildung 4.1: Ändert sich eine beliebige Zelle im Bereich C5:C16, wird die Arbei...

Abbildung 4.2: Tippen Sie den Code in das Tabellenblattereignis `Change` ein oder k...

Abbildung 4.3: Wenn Sie versuchen, die Arbeitsmappe zu schließen, w...

Abbildung 4.4: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis `B`...

Abbildung 4.5: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis B...

Abbildung 4.6: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis O...

Abbildung 4.7: Sie möchten erreichen, dass beim Öffnen der Arbeitsmappe das Tabel...

Abbildung 4.8: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis O...

Abbildung 4.9: Das von einem Makro aktivierte Dialogfeld zum Öffnen einer Datei

Abbildung 4.10: Sie können diese nervige Meldung vermeiden, die an...

Abbildung 4.11: Eine deutliche und gut lesbare Meldung

Abbildung 4.12: Sie können das Schließen einer Arbeitsmappe verhindern, solange e...

Abbildung 4.13: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis

Kapitel 5

Abbildung 5.1: Entfernen Sie im Makro die Anweisung `Application.DisplayAlerts = F...`

Abbildung 5.2: Ohne Makro sind Sie auf das Excel-Dialogfeld »Einblenden« angewies...

Abbildung 5.3: Es ist häufig nützlich, wenn die Tabellenblätter alphabetisch sort...

Abbildung 5.4: Sie können im Blattregister ein Tabellenblatt mit der rechten Maus...

Abbildung 5.5: Der Befehl »Blatt schützen« ist deaktiviert, wenn Sie mehrere Tabe...

Abbildung 5.6: Der Befehl »Blattschutz aufheben« ist deaktiviert, wenn Sie mehrer...

Abbildung 5.7: Mit einem Inhaltsverzeichnis lässt sich sehr schnell auch in einer...

Abbildung 5.8: Tippen Sie den Code in das Tabellenblattereignis `BeforeDoubleClick`

Abbildung 5.9: Dank der hervorgehobenen Zeile und Spalte ist es einfacher, die an...

Abbildung 5.10: Geben Sie den Code in das Tabellenblattereignis BeforeDoubleClick...

Kapitel 6

Abbildung 6.1: Klicken Sie auf den Befehl »Namen definieren«, um den ausgewählten...

Abbildung 6.2: Hier legen Sie den Namen für den Bereich fest.

Abbildung 6.3: Das Dialogfeld »Namens-Manager« zeigt alle benannten Bereiche an.

Abbildung 6.4: Tippen Sie den Code in die Prozedur für das Arbeitsm...

Abbildung 6.5: Das Dialogfeld »Inhalte auswählen«

Abbildung 6.6: Aktivieren Sie im Dialogfeld »Inhalte auswählen« das Kontrollkästc...

Abbildung 6.7: Sie können ein Makro verwenden, um dynamisch die erste freie Zelle...

Kapitel 7

Abbildung 7.1: Importierte Zahlen sind manchmal als Text formatiert.

Abbildung 7.2: Wählen Sie den Befehl »Text in Spalten«.

Abbildung 7.3: Klicken Sie auf »Fertig stellen«, um falsch formatierte Zahlen zu ...

Abbildung 7.4: Dieses Makro sucht Duplikate in einem Bereich und heben sie hervor...

Abbildung 7.5: Nur Zeilen, die Duplikate enthalten, sind noch zu sehen.

Abbildung 7.6: Der Standard-AutoFilter fügt in jede Spalte des Datenbereichs eine...

Abbildung 7.7: Mit ein wenig VBA-Code erreichen Sie, dass nur die Spalten »Vertri...

Abbildung 7.8: Dieses Makro zeigt alle gefilterten Spalten in der Statusleiste an...

Abbildung 7.9: Tippen Sie den Code in das Tabellenblattereignis `Calculate` ein ode...

Kapitel 8

Abbildung 8.1: Verzeichnis aller PivotTable-Berichte

Abbildung 8.2: Excel zeigt eine Liste der Funktionen an, die Sie für die Zusammen...

Abbildung 8.3: Tippen Sie den Code in das Tabellenblattereignis `BeforeDoubleClick`

Abbildung 8.4: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis `B`...

Abbildung 8.5: Im Auswahlbereich werden alle Diagrammobjekte sowie deren Namen an...

Kapitel 10

Abbildung 10.1: Wählen Sie die Datenbankquelle aus, die die Daten enthält, die Si...

Abbildung 10.2: Wählen Sie das Access-Objekt aus, das Sie importieren wollen.

Abbildung 10.3: Legen Sie fest, wo und wie Sie die Access-Daten sehen wollen.

Abbildung 10.4: Daten, die aus Access importiert wurden.

Abbildung 10.5: Solange eine Verbindung mit Ihrer Datenbank verfügbar ist, können...

Abbildung 10.6: Im Excel 2019-Aufgabenbereich ABFRAGEN UND VERBINDUNGEN werden au...

Abbildung 10.7: Legen Sie auf der Registerkarte DEFINITION den Befehlstyp auf »SQ...

Abbildung 10.8: Legen Sie die Zelle fest, in der das Auswahlkriterium eingegeben ...

Abbildung 10.9: Merken Sie sich den Namen der Verbindung, in diesem Fall *Facility*...

Abbildung 10.10: Sie haben nun ein einfach zu verwendendes Verfahren, um die exte...

Abbildung 10.11: Wählen Sie die aktuelle Version der Microsoft ActiveX Data Objec...

Kapitel 11

Abbildung 11.1: Ein Hochkomma am Anfang einer Zeile wandelt diese Zeile in einen ...

Abbildung 11.2: Verwenden Sie die Symbolleiste BEARBEITEN, wenn Sie einen Block a...

Abbildung 11.3: Wenn Sie beim Ziehen die Strg -Taste gedrückt halten, erstellen Sie ...

Abbildung 11.4: Drücken Sie ⇧+F2, wenn der Cursor auf dem Namen einer Variablen od...

Abbildung 11.5: Verwenden Sie die Schaltfläche »Prozeduransicht«, um den Bildlauf...

Abbildung 11.6: Drücken Sie auf Ihrer Tastatur F8, um das Makro schrittweise ausfü...

Abbildung 11.7: Ziehen Sie den Pfeil, während Ihr Code schrittweise ausgeführt wi...

Abbildung 11.8: Ein Haltepunkt wird durch einen roten Punkt in der Markierungslei...

Abbildung 11.9: Anzeige der ersten und der letzten Zeichen einer String-Variablen...

Abbildung 11.10: Eine unvollendete Codezeile führt zu diesem irritierenden Warnhi...

Abbildung 11.11: So unterdrücken Sie während der Programmierung die Syntax-Warnhi...

Einleitung

Im weitesten Sinne ist ein Excel-Makro eine Reihe von Anweisungen, die irgendeine Funktion von Excel automatisieren. Sie können Excel somit effizienter einsetzen und machen weniger Fehler. Sie können beispielsweise ein Makro programmieren, um einen monatlichen Umsatzbericht zu drucken. Nachdem Sie das Makro entwickelt haben, können Sie es ausführen und die ganze ansonsten zeitaufwändige Aktion in einem Rutsch durchführen.

Makros werden in VBA geschrieben. VBA ist die Abkürzung für *Visual Basic for Applications* (Visual Basic für Anwendungen). Visual Basic for Applications ist eine von Microsoft entwickelte Programmiersprache und ein Werkzeug, um Programme zu entwickeln, mit denen Sie Excel steuern können.

Die Begrifflichkeit der Excel-Programmierung kann ein wenig verwirrend sein. So ist beispielsweise VBA sowohl eine Programmiersprache als auch eine Makrosprache. Wie nennen Sie etwas, das in VBA geschrieben wurde und in Excel ausgeführt wird? Ist es ein Makro oder ist es ein Programm? Das Excel-Hilfesystem verwendet für VBA-Prozeduren den Begriff Makro; daher wird in diesem Buch die gleiche Terminologie verwendet.

Im Verlauf dieses Buchs werden Sie häufig auf den Begriff *automatisieren* stoßen. Automatisierung bedeutet, dass eine Reihe von Schritten automatisch durchgeführt wird. Wenn Sie beispielsweise ein Makro schreiben, das die Hintergrundfarbe von Zellen ändert, das Tabellenblatt druckt und die Einfärbung wieder entfernt, dann haben Sie diese drei Schritte automatisiert.

Sie wissen bestimmt, dass Excel-Anwender das Programm für Tausende von unterschiedlichen Aufgaben einsetzen. Hier nur ein paar Beispiele:

- ✔ Verwaltung von Listen oder Tabellen, beispielsweise für Kundendaten oder Transaktionen
- ✔ Budgetplanung und Umsatzprognosen

- ✔ Analyse wissenschaftlicher Daten
- ✔ Erstellung von Rechnungen und anderen Formularen
- ✔ Daten in Form von Diagrammen darstellen

Diese Liste könnte noch stundenlang fortgesetzt werden. Der entscheidende Punkt ist, dass Excel für unglaublich viele verschiedene Aufgaben verwendet wird, und alle, die dieses Buch lesen, haben im Hinblick auf Excel unterschiedliche Anforderungen und Erwartungen. Etwas, was wohl die meisten Leser bewegt, ist, dass jeder irgendeine Funktion von Excel automatisieren will. Und genau darum geht es bei Makros und in diesem Buch.

Über dieses Buch

Diesem Buch über Excel-Makros liegt die Erkenntnis zugrunde, dass die Programmierung in VBA Zeit und Übung erfordert – Zeit, die Sie im Moment vielleicht gar nicht haben. Tatsächlich können sich viele Datenanalytiker nicht den Luxus leisten, sich ein paar Wochen Zeit zu nehmen, um zu VBA-Experten zu werden. Das Buch trägt diesem Sachverhalt Rechnung. Sie erhalten daher nicht einen allgemeinen Überblick zu VBA, sondern finden im Buch praxisnahe Beispielmakros zu den Einsatzgebieten, die am häufigsten vorkommen.

Jeder Abschnitt des Buchs beschreibt eine Aufgabe, die häufig vorkommt, und liefert dann ein Excel-Makro, mit dem diese Aufgabe erledigt werden kann. Außerdem erhalten Sie detaillierte Informationen dazu, wie das Makro funktioniert und wie Sie es einsetzen können.

Jeder Abschnitt enthält die folgenden Informationen:

- ✔ das Problem,
- ✔ die Makrolösung,
- ✔ wie das Makro funktioniert.

Nachdem Sie einen Abschnitt gelesen haben, sind Sie in der Lage,

- ✔ das beschriebene Makro sofort zu implementieren,
- ✔ die Funktionsweise des Makros zu verstehen,
- ✔ das Makro in anderen Arbeitsmappen zu verwenden,
- ✔ die beschriebene Technik in anderen Makros einzusetzen.

Die Makros in diesem Buch wurden mit der Zielsetzung erstellt, Ihnen einen schnellen und einfachen Einstieg in VBA zu ermöglichen. Jedes Makro behandelt eine regelmäßig vorkommende Aufgabe, bei der sich eine Automatisierung lohnt. Die Idee ist, dass Sie die Verwendung von Makros erlernen, indem Sie Makros benutzen. Dieses Buch wurde so geschrieben, dass Sie die einzelnen Makros implementieren können und sozusagen ganz nebenbei lernen, was das Makro wie macht.

Törichte Annahmen über die Leser

Ich setze nicht viel voraus, aber die folgenden Vermutungen habe ich schon angestellt, als ich mir Sie, meine Leser, vorgestellt habe:

- ✔ Sie haben Microsoft Excel 2007 oder eine neuere Version installiert.
- ✔ Sie haben grundlegende Kenntnisse über die Datenanalyse mit Excel, zum Beispiel über das Arbeiten mit Tabellen, die Zusammenfassung von Daten, das Erstellen von Formeln, das Verweisen auf Zellen und das Filtern und Sortieren.
- ✔ Sie können auf das Internet zugreifen, um die Beispieldateien herunterzuladen, die Sie auf https://www.wiley-vch.de/de/dummies/downloads finden.

Icons in diesem Buch

Was wäre ein »… für Dummies«-Buch ohne die vertrauten Icons, die Sie auf wichtige Informationen hinweisen und Sie auf Ihrem Weg unterstützen. In diesem Abschnitt stelle ich kurz die in diesem Buch verwendeten Icons vor.

 Dieses Tipp-Icon weist Sie auf hilfreiche Empfehlungen und andere nützliche Informationen hin.

 Beim Symbol »Nicht vergessen« finden Sie allgemeine, interessante und nützliche Hinweise, die Sie sich für den späteren Einsatz merken sollten.

 Das Icon »Warnung« macht Sie auf drohende Gefahren aufmerksam. Sie finden hier Informationen, auf die Sie achten sollten, und es rät Ihnen, hier besonders behutsam vorzugehen.

Die Beispieldateien zu diesem Buch

Im Web finden Sie Beispieldateien zu diesem Buch.

Für jedes Makro in diesem Buch gibt es eine Beispieldatei, die zeigt, wie das Makro funktioniert und in der Sie sich den Makro-Code ansehen können. Sie können die Beispieldateien auch verwenden, um den Code in Ihre eigenen Arbeitsmappen einzufügen: Sie ersparen sich so Tipparbeit. Laden Sie die Beispieldateien von folgender Webseite herunter:

https://www.wiley-vch.de/de/dummies/downloads

Suchen Sie auf dieser Seite nach *Excel Makros für Dummies* und klicken Sie den gefundenen Eintrag an, um die ZIP-Datei mit allen Beispielen herunterzuladen.

Sie finden bei jedem Beispielmakro ausführliche Informationen dazu, wo Sie den kopierten Code einfügen müssen. In den meisten Fällen öffnen Sie die Beispieldatei mit dem betreffenden Makro, gehen zum Visual-Basic-Editor (indem Sie Alt+F11 drücken) und kopieren den Code. Gehen Sie dann zu Ihrer eigenen Arbeitsmappe, öffnen Sie den Visual-Basic-Editor und fügen Sie den Code an der gewünschten Stelle ein.

Beachten Sie, dass Sie bei einigen Makros Änderungen vornehmen müssen, damit sie zu Ihrer Umgebung passen. So gibt es beispielsweise in Kapitel 4 ein Makro, das alle Arbeitsmappen im Verzeichnis `c:\Temp\` druckt. Bevor Sie dieses Makro verwenden, müssen Sie den Ordner angeben, in dem sich auf Ihrem Computer die zu druckenden Arbeitsmappen befinden.

Falls ein bestimmtes Makro bei Ihnen nicht funktioniert, müssen Sie in den meisten Fällen nicht viel mehr als eine bestimmte Komponente ändern. Achten Sie insbesondere auf Adressen von Zellbereichen, Namen von Verzeichnissen und andere »hart-kodierte« Namen.

Wie es weitergeht

Falls das Thema Excel-Makros für Sie vollkommenes Neuland ist, beginnen Sie am besten mit den Kapiteln 1 bis 3. Sie finden dort das Grundlagenwissen, auf dem die Makros in diesem Buch basieren. Mit diesen kurzen und prägnanten Informationen werden Sie verstehen, wie Makros und VBA funktionieren. Außerdem wird dort das Fundament geschaffen, das Sie brauchen, um die Makros dieses Buchs zu implementieren.

Falls Sie bereits Erfahrungen mit Excel-Makros besitzen und direkt in die Makrobeispiele eintauchen wollen, schauen Sie sich die Kapitel 4 bis 9 an und suchen Sie sich die Makros heraus, die für Sie besonders interessant sind. Jedes Makrobeispiel steht für sich allein. Jeder Abschnitt enthält also alles, was Sie brauchen, um den jeweiligen Code zu verstehen und um ihn in Ihren eigenen Arbeitsmappen zu verwenden.

In Teil II finden Sie Makros, die regelmäßig wiederkehrende Arbeitsmappen- und Tabellenblattaufgaben automatisieren und die Ihnen helfen, Zeit zu sparen und die Aufgaben effizienter zu erledigen.

Durchforsten Sie Teil III, wenn Sie Makros suchen, mit denen Sie in Zellbereichen navigieren, Zellen formatieren und die Daten Ihrer Arbeitsmappen bearbeiten können.

Wenn Sie auf der Suche nach Makros für die Automatisierung von redundanten Schritten beim Einsatz von PivotTable-Berichten und Diagrammen sind wenn Sie Makros benötigen, mit denen Sie E-Mails mit Excel-Dateien als Anhängen versenden können, oder wenn Sie Verbindungen mit externen Datenquellen herstellen wollen, dann blättern Sie durch Teil IV.

Vergessen Sie nicht, sich auch die Kapitel in Teil V anzusehen, dem immer wieder beliebten »… für Dummies«-Top-Ten-Teil. Sie finden dort nützliche und hilfreiche Tipps und Hinweise, mit denen Sie noch mehr aus Ihren Makro-Fähigkeiten herausholen können.

Hier noch ein paar abschließende Bemerkungen, die die Verwendung der Beispielmakros vereinfachen:

- **Alle Excel-Dateien, die Makros enthalten, müssen die Dateinamenserweiterung .xlsm besitzen.** In Kapitel 1 finden Sie einen eigenen Abschnitt mit wichtigen Informationen zu den Dateierweiterungen für Arbeitsmappen mit Makros.
- **Excel führt ein Makro erst dann aus, wenn es aktiviert wird.** Wenn Sie die Makros implementieren, müssen Sie und Ihre Kunden die Sicherheitsvorkehrungen von Excel beachten.

Im Abschnitt »Makrosicherheit ab Excel 2010« in Kapitel 1 finden Sie weiterführende Informationen dazu.

- **Sie können Makro-Aktionen nicht rückgängig machen.** Während Sie in Excel arbeiten, können Sie oft die Schritte, die Sie gerade durchgeführt haben, wieder rückgängig machen, da Excel die letzten 100 Aktionen in einem Protokoll festhält. Wenn Sie ein Makro ausführen, werden diese Aktionen nicht protokolliert und sie können daher auch nicht rückgängig gemacht werden.
- **Sie müssen die Makros so anpassen, dass sie zu Ihren Arbeitsmappen passen.** Viele der Makrobeispiele verwenden Zelladressen oder Namen von Tabellenblättern, die möglicherweise so in Ihren Arbeitsmappen nicht vorkommen. Passen Sie daher alle verwendeten Namen von Tabellenblättern (wie *Tabelle1*) und die Zellbereiche wie in `Range("A1")` so an, dass sie zu den Zellbereichen und den Tabellenblättern passen, die Sie in Ihren Arbeitsmappen verwenden.

Teil I

Heiliges Makro, Batman!

IN DIESEM TEIL …

- ✔ Legen Sie das Fundament für Ihr Makro-Können und lernen Sie die grundlegenden Konzepte der Makroaufzeichnung kennen
- ✔ Erlangen Sie solides Grundlagenwissen zur Verwendung und Verteilung von Makros in Excel
- ✔ Erforschen Sie die Excel-Programmierung und tauchen Sie in die Welt des Visual-Basic-Editors ein
- ✔ Verwenden Sie das Excel-Objektmodell, um Ihre eigenen Makros komplett selbst zu schreiben
- ✔ Verstehen Sie, welche Rollen Variablen, Ereignisse und die Fehlerbehandlung bei der Makroprogrammierung spielen

Kapitel 1

Makrogrundlagen

IN DIESEM KAPITEL

- Warum Sie Makros verwenden sollten
- Makros aufzeichnen
- Makrosicherheit verstehen
- Beispiele für Makros in der Praxis

Ein Makro ist im Grunde eine Reihe von Anweisungen oder Code, mit dem Sie Excel anweisen, bestimmte Aktionen auszuführen. Sie können Excel-Makros schreiben/programmieren oder aufzeichnen. Das Schlüsselwort für dieses Kapitel lautet *Aufzeichnung*.

Das Aufzeichnen eines Makros ähnelt dem Speichern einer Telefonnummer in Ihrem Handy. Auf Ihrem Handy wählen Sie die Nummer zuerst von Hand, und anschließend speichern Sie sie. Danach können Sie die gespeicherte Nummer mit einem einzigen Tastendruck erneut wählen. In Excel starten Sie die Makroaufzeichnung und führen dann die gewünschten Aktionen aus. Während der Aufzeichnung übersetzt Excel im Hintergrund Ihre Tastenanschläge und Mausklicks in Code, und zwar in einer Programmiersprache mit dem Namen *Visual Basic for Applications* (VBA, Visual Basic für Anwendungen). Nachdem ein Makro aufgezeichnet ist, können Sie es so oft Sie wollen starten und so die aufgezeichneten Aktionen immer wieder neu ausführen lassen.

In diesem Kapitel erkunden Sie Makros und Sie werden dabei sehen, wie Sie mit Makros sich wiederholende Aufgaben

automatisieren und sich so Ihr Leben (zumindest Ihr Leben mit Excel) einfacher machen können.

Warum sollten Sie Makros verwenden?

Der erste Schritt bei der Verwendung von Makros ist die Erkenntnis, dass Sie ein Problem haben. Genau genommen haben Sie mehrere Probleme:

- **Problem 1: Sich wiederholende Aufgaben**An jedem Monatsanfang müssen Sie wieder und wieder diesen einen Bericht erstellen. Sie müssen Daten importieren. Sie müssen diese Pivot-Tabellen aktualisieren. Sie müssen diese Spalten löschen und so weiter. Wäre es nicht fantastisch, wenn Sie einfach ein Makro starten könnten, das die redundanten Schritte beim Erstellen Ihres Dashboards automatisch durchführt?
- **Problem 2: Sie machen Fehler** Wenn man mit Excel in den Nahkampf geht, macht man zwangsläufig Fehler. Wenn Sie öfters Formeln verwenden, Tabellen sortieren oder Daten von Hand verschieben, besteht immer die Gefahr einer Katastrophe. Kommen dann noch enge Abgabetermine und permanent neue Änderungswünsche hinzu, steigt die Wahrscheinlichkeit von Fehlern noch weiter. Warum zeichnen Sie nicht einfach ein Makro auf, kontrollieren noch schnell, dass es auch korrekt funktioniert, und vergessen dann einfach die einzelnen Schritte der Aufgabe? Das Makro führt bei jedem Start alle Aktionen immer gleich durch, und Sie reduzieren so die Fehlerwahrscheinlichkeit.
- **Problem 3: Schwierige Navigation** Sie erstellen Berichte häufig für Anwender, die nur geringe Excel-Kenntnisse haben. Daher ist es immer hilfreich, Ihre Berichte so zu erstellen, dass diese einfach zu verwenden sind.. Makros können verwendet werden, um Tabellenblätter dynamisch zu formatieren und zu

drucken, um bestimmte Tabellenblätter in der Arbeitsmappe zu öffnen und sogar um das geöffnete Dokument in einem bestimmten Ordner zu speichern. Ihre Kundschaft wird diesen Komfort, durch den die Durchsicht Ihrer Arbeitsmappen angenehmer wird, sehr zu schätzen wissen.

Grundlagen der Makroaufzeichnung

Damit Sie Ihr erstes Makro aufzeichnen können, müssen Sie zuerst den Makrorekorder finden: Dieser befindet sich auf der Registerkarte ENTWICKLERTOOLS. Leider ist diese Registerkarte in der Standardkonfiguration von Excel nicht sichtbar und es kann daher sein, dass Sie diese Registerkarte auch in Ihrer Excel-Version nicht sehen. Da Sie mit Excel-Makros arbeiten wollen, schalten Sie die Registerkarte ENTWICKLERTOOLS ein. Das geht so:

1. **Wählen Sie DATEI | OPTIONEN.**
2. **Wählen Sie im Dialogfeld EXCEL-OPTIONEN die Kategorie MENüBAND ANPASSEN .**
3. **Schalten Sie in der Liste auf der rechten Seite das Kontrollkästchen neben ENTWICKLERTOOLS ein.**
4. **Klicken Sie auf OK, um zu Excel zurückzukehren.**

Da die Registerkarte ENTWICKLERTOOLS nun sichtbar ist, können Sie den Makrorekorder starten, indem Sie auf der Registerkarte ENTWICKLERTOOLS in der Gruppe CODE auf MAKRO AUFZEICHNEN klicken. Hierdurch wird das Dialogfeld MAKRO AUFZEICHNEN aktiviert, das Sie in Abbildung 1.1 sehen.

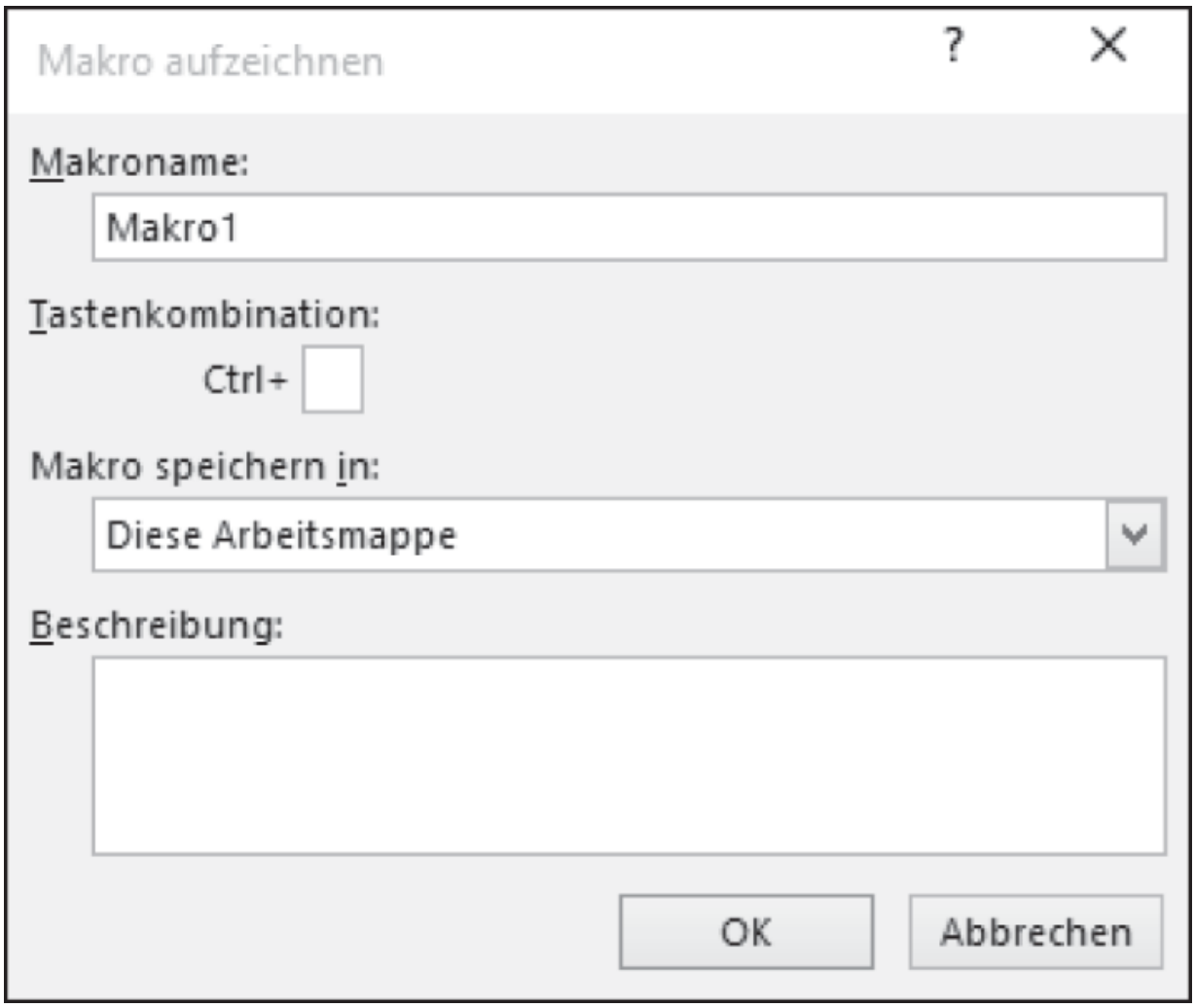


Abbildung 1.1: Das Dialogfeld »Makro aufzeichnen«

In diesem Dialogfeld können Sie die folgenden vier Eingaben vornehmen:

- **Makroname:** Excel weist Ihrem Makro einen allgemeinen Namen zu, wie hier `Makro1`. Sie sollten Ihrem Makro jedoch einen Namen zuweisen, der besser beschreibt, was das eigentlich Makro macht. Wenn Sie beispielsweise ein Makro erstellen, das eine allgemeine Tabelle formatiert, dann nennen Sie das Makro `TabelleFormatieren`.
- **Tastenkombination:** Jedes Makro benötigt ein Ereignis, also irgendetwas, das passiert, damit es gestartet wird. Dies kann der Klick auf eine Schaltfläche sein, das Öffnen einer

Arbeitsmappe oder wie in diesem Fall eine Tastenkombination. Wenn Sie dem Makro eine Tastenkombination zuweisen, können Sie das Makro mit dieser Tastenkombination starten. Dieses Feld ist optional.

- ✔ **Makro speichern in:** Die Standardeinstellung dieser Liste ist DIESE ARBEITSMAPPE. Wenn Sie das Makro in DIESE(R) ARBEITSMAPPE speichern, dann befindet es sich nach der Aufzeichnung in der gleichen Datei wie die aktuelle Arbeitsmappe. Beim nächsten Öffnen dieser Arbeitsmappe steht Ihnen das Makro automatisch wieder zur Verfügung. Sie können die Arbeitsmappe an einen anderen Excel-Anwender senden und auch er kann die enthaltenen Makros starten (vorausgesetzt, die Makrosicherheitseinstellungen sind richtig konfiguriert – mehr hierzu weiter hinten in diesem Kapitel).
- ✔ **Beschreibung:** Verwenden Sie dieses optionale Feld, wenn sich in einer Arbeitsmappe mehrere Makros befinden oder wenn Sie andere Benutzer detailliert darüber informieren wollen, was genau das Makro macht.

Führen Sie bei geöffnetem Dialogfeld MAKRO AUFZEICHNEN folgende Schritte durch. Sie erstellen in diesem Beispiel ein einfaches Makros, das Ihren Namen in eine Zelle eines Tabellenblatts einfügt:

1. **Geben Sie in das Feld MAKRONAME einen aus einem Wort bestehenden Namen für das Makro ein und ersetzen Sie so den vorgegebenen Namen »Makro1«.**

   Ein guter Name für dieses Makro ist *MeinName*.

2. **Weisen Sie dem Makro die Tastenkombination** ⇧ + Strg + N **zu.**

   Drücken Sie dazu die ⇧ -Taste gedrückt und geben Sie in das Feld TASTENKOMBINATION ein großes N ein.

3. **Klicken Sie auf OK.**

   Hierdurch wird das Dialogfeld MAKRO AUFZEICHNEN geschlossen und die Aufzeichnung Ihrer Aktionen beginnt.

4. **Markieren Sie eine Zelle in Ihrem Excel-Tabellenblatt, geben Sie dort Ihren Namen ein und drücken Sie ↵.**
5. **Wählen Sie ENTWICKLERTOOLS | CODE | AUFZEICHNUNG BEENDEN (oder klicken Sie in der Excel-Statusleiste auf die Schaltfläche AUFZEICHNUNG BEENDEN).**

Das Makro untersuchen

Das Makro wurde in einem Modul mit dem Namen *Modul1* abgelegt. Damit Sie den Code dieses Moduls sehen können, müssen Sie den Visual-Basic-Editor (VBE) aktivieren. Sie können dies auf zwei Arten tun:

- ✔ Drücken Sie Alt + F11 .
- ✔ Wählen Sie ENTWICKLERTOOLS | CODE | VISUAL BASIC.

Der Projekt-Explorer des Visual-Basic-Editors enthält eine Liste aller derzeit geöffneten Arbeitsmappen und Add-Ins. Diese Liste wird als Baumdiagramm dargestellt; Sie können die einzelnen Knoten dieser Liste erweitern oder reduzieren. Der soeben von Ihnen aufgezeichnete Code befindet sich in MODUL1 unterhalb des Knotens MODULE, und zwar in der aktuellen Arbeitsmappe. Wenn Sie auf MODUL1 doppelklicken, wird der Code dieses Moduls im Codefenster angezeigt.

Das aufgezeichnete Makro sollte ungefähr so aussehen (sollten Sie nicht Michael Alexander heißen, sieht es an einer Stelle etwas anders aus; ahnen Sie, wo?):

```
Sub MeinName()
'
' MeinName Makro
'
' Tastenkombination: Strg+Umschalt+N
'
 ActiveCell.FormulaR1C1 = "Michael Alexander"

End Sub
```

Das aufgezeichnete Makro ist eine Sub-Prozedur mit dem Namen `MeinName`. Die Anweisungen informieren Excel darüber, was bei der Ausführung des Makros gemacht werden soll.

Beachten Sie, dass Excel am Anfang der Prozedur ein paar Kommentare eingefügt hat; ein Teil dieser Informationen wurde im Dialogfeld MAKRO AUFZEICHNEN eingegeben. Die Kommentarzeilen (sie werden mit einem Hochkomma eingeleitet) sind nicht zwingend erforderlich; falls Sie sie löschen, wird das Makro weiterhin funktionieren. Wenn Sie die Kommentare ignorieren, werden Sie sehen, dass die Prozedur nur eine einzige VBA-Anweisung enthält:

```
ActiveCell.FormulaR1C1 = "Michael Alexander"
```

Diese Anweisung gibt den Namen in die aktive Zelle ein, den Sie während der Makroaufzeichnung eingetippt haben.

Das Makro testen

Bevor Sie mit der Makroaufzeichnung begannen, haben Sie dem Makro die Tastenkombination ⇧ + Strg + N zugewiesen. Um das Makro zu testen, kehren Sie zu Excel zurück. Zwei Methoden stehen Ihnen zur Verfügung:

- ✔ Drücken Sie Alt + F11 .
- ✔ Klicken Sie in der Symbolleiste des Visual-Basic-Editors auf ANSICHT MICROSOFT EXCEL.

Aktivieren Sie ein Tabellenblatt, sobald Sie zurück in Excel sind. (Das Tabellenblatt kann sich in der Arbeitsmappe befinden, in der sich das VBA-Modul befindet, oder in einer anderen Arbeitsmappe). Markieren Sie eine Zelle und drücken Sie ⇧ + Strg + N. Das Makro fügt Ihren Namen in die ausgewählte Zelle ein.

Beachten Sie, dass Sie in obigem Beispiel zuerst die Zelle, die geändert werden soll, markiert und erst dann die Makroaufzeichnung gestartet haben. Dieser Schritt ist wichtig. Wenn Sie eine Zelle markieren, während die Makroaufzeichnung bereits läuft, wird das Auswählen der Zelle ebenfalls in das Makro eingefügt. In diesem Fall würde der Text immer in die aufgezeichnete Zelle eingefügt und Sie erhalten dann kein allgemein verwendbares Makro.

Das Makro bearbeiten

Nachdem Sie ein Makro aufgezeichnet haben, können Sie es bearbeiten oder erweitern (vorausgesetzt, Sie wissen, was Sie tun). Angenommen, Sie wollen, dass Ihr Name in Fett ausgegeben wird. Sie können das Makro erneut aufzeichnen. Da es sich nur um eine kleine Anpassung handelt, ist es jedoch effizienter, das bereits aufgezeichnete Makro zu bearbeiten. Drücken Sie Alt + F11, um zum Visual-Basic-Editor zu wechseln. Aktivieren Sie dann MODUL1 und fügen Sie vor der Anweisung, die Ihren Namen einfügt, die folgende Zeile ein:

```
ActiveCell.Font.Bold = True
```

Das bearbeitete Makro sieht dann so aus:

```
Sub MeinName()
'
' MeinName Makro
'
' Tastenkombination: Strg+Umschalt+N
'
 ActiveCell.Font.Bold = True

 ActiveCell.FormulaR1C1 = "Michael Alexander"

End Sub
```

Testen Sie das neue Makro, um zu sehen, ob es wie geplant funktioniert.

Absolute und relative Makroaufzeichnung im Vergleich

Nachdem Sie nun die Grundlagen der Makroaufzeichnung kennen, ist es an der Zeit, etwas tiefer in die Materie einzusteigen. Das Erste, was Sie verstehen müssen, ist, dass Excel zwei unterschiedliche Modi für die Makroaufzeichnung kennt: einen Modus mit absoluten Verweisen und einen weiteren mit relativen Verweisen.

Makros mit absoluten Verweisen aufzeichnen

Excel verwendet bei der Makroaufzeichnung standardmäßig absolute Verweise. Wie Sie vielleicht wissen, wird der Begriff absoluter Verweis oft im Zusammenhang mit Zellbezügen in Formeln verwendet. Wenn ein Zellverweis in einer Formel ein absoluter Verweis ist, dann passt Excel den Verweis nicht automatisch an, wenn die Formel in eine andere Zelle kopiert wird.

Wie sich dieses Konzept auf Makros auswirkt, lässt sich am besten verstehen, indem man es ausprobiert. Öffnen Sie die Beispieldatei *1.1 Beispiele.xlsx* und zeichnen Sie ein Makro auf, das die Zeilen des Tabellenblatts *Filialen* zählt (Abbildung 1.2).

Sie finden das Beispieldatenset für dieses Kapitel in den Begleitdateien, die Sie von der Website des Buchs herunterladen können. Weitere Informationen hierzu finden Sie in der Einleitung des Buchs.

Führen Sie folgende Schritte durch, um das Makro aufzuzeichnen:

1. **Achten Sie darauf, dass vor Aufzeichnung des Makros Zelle A1 markiert ist.**
2. **Klicken Sie auf der Registerkarte ENTWICKLERTOOLS in der Gruppe CODE auf MAKRO AUFZEICHNEN.**
3. **Nennen Sie das Makro `AddTotal`.**
4. **Legen Sie als Speicherort DIESE ARBEITSMAPPE fest.**

	A	B	C	D	E	F	G	H	I	J
1		Region	Absatzmarkt	Filiale			Region	Absatzmarkt	Filiale	
2		NORD	BUFFALO	601419			SÜD	CHARLOTTE	173901	
3		NORD	BUFFALO	701407			SÜD	CHARLOTTE	301301	
4		NORD	BUFFALO	802202			SÜD	CHARLOTTE	302301	
5		NORD	CANADA	910181			SÜD	CHARLOTTE	601306	
6		NORD	CANADA	920681			SÜD	DALLAS	202600	
7		NORD	MICHIGAN	101419			SÜD	DALLAS	490260	
8		NORD	MICHIGAN	501405			SÜD	DALLAS	490360	
9		NORD	MICHIGAN	503405			SÜD	DALLAS	490460	
10		NORD	MICHIGAN	590140			SÜD	FLORIDA	301316	
11		NORD	NEWYORK	801211			SÜD	FLORIDA	701309	
12		NORD	NEWYORK	802211			SÜD	FLORIDA	702309	
13		NORD	NEWYORK	804211			SÜD	NEWORLEANS	601310	
14		NORD	NEWYORK	805211			SÜD	NEWORLEANS	602310	
15		NORD	NEWYORK	806211			SÜD	NEWORLEANS	801607	
16										
17										

Abbildung 1.2: Dieses Tabellenblatt enthält zwei nebeneinander stehende Tabellen, die gleich aufgebaut sind.

5. **Klicken Sie auf OK, um die Aufzeichnung zu starten.**

   Von diesem Punkt an zeichnet Excel Ihre Aktionen auf. Während die Makroaufzeichnung läuft, führen Sie diese Schritte durch:

   1. Markieren Sie Zelle A16 und geben dort den Text `Anzahl` ein.
   2. Markieren Sie die erste leere Zelle in Spalte D (Zelle D16) und geben Sie dort `=ANZAHL2(D2:D15)` ein.
   3. Diese Formel zeigt unten in Spalte D die Anzahl der Filialnummern an. Sie verwenden ANZAHL2, da die Filialnummern als Text eingegeben wurden.

4. Klicken Sie auf der Registerkarte ENTWICKLERTOOLS in der Gruppe CODE auf AUFZEICHNUNG BEENDEN, um die Makroaufzeichnung zu beenden.

Das formatierte Tabellenblatt sollte nun ungefähr so aussehen, wie Abbildung 1.3.

Probieren Sie das Makro aus. Löschen Sie dazu die Zeile 16 und starten Sie das Makro, indem Sie folgende Schritte durchführen:

1. **Klicken Sie auf der Registerkarte ENTWICKLERTOOLS auf MAKROS.**
2. **Suchen Sie in der Liste nach dem Makro ADDTOTAL, das Sie soeben aufgezeichnet haben.**
3. **Klicken Sie die Schaltfläche AUSFüHREN an.**

Wenn alles gut läuft, führt das Makro die aufgezeichneten Aktionen aus und Sie erhalten unterhalb der Tabelle die Zeile *Anzahl*. Ganz egal, wie sehr Sie es auch versuchen, es ist nicht möglich, das Makro für die zweite Tabelle (G1:I15 in Abbildung 1.3) des Tabellenblatts zu verwenden. Warum? Weil Sie das Makro als absolutes Makro aufgezeichnet haben.

	A	B	C	D	E	F	G	H	I	J
1		Region	Absatzmarkt	Filiale			Region	Absatzmarkt	Filiale	
2		NORD	BUFFALO	601419			SÜD	CHARLOTTE	173901	
3		NORD	BUFFALO	701407			SÜD	CHARLOTTE	301301	
4		NORD	BUFFALO	802202			SÜD	CHARLOTTE	302301	
5		NORD	CANADA	910181			SÜD	CHARLOTTE	601306	
6		NORD	CANADA	920681			SÜD	DALLAS	202600	
7		NORD	MICHIGAN	101419			SÜD	DALLAS	490260	
8		NORD	MICHIGAN	501405			SÜD	DALLAS	490360	
9		NORD	MICHIGAN	503405			SÜD	DALLAS	490460	
10		NORD	MICHIGAN	590140			SÜD	FLORIDA	301316	
11		NORD	NEWYORK	801211			SÜD	FLORIDA	701309	
12		NORD	NEWYORK	802211			SÜD	FLORIDA	702309	
13		NORD	NEWYORK	804211			SÜD	NEWORLEANS	601310	
14		NORD	NEWYORK	805211			SÜD	NEWORLEANS	602310	
15		NORD	NEWYORK	806211			SÜD	NEWORLEANS	801607	
16	Anzahl			14						
17										

Abbildung 1.3: Das Tabellenblatt mit der neuen Zeile »Anzahl«

Lassen Sie uns das Makro untersuchen, um besser zu verstehen, was das genau bedeutet. Um den Code zu untersuchen, klicken Sie auf der Registerkarte ENTWICKLERTOOLS auf MAKROS. Sie sehen dann das Dialogfeld MAKRO, wie es Abbildung 1.4 zeigt.

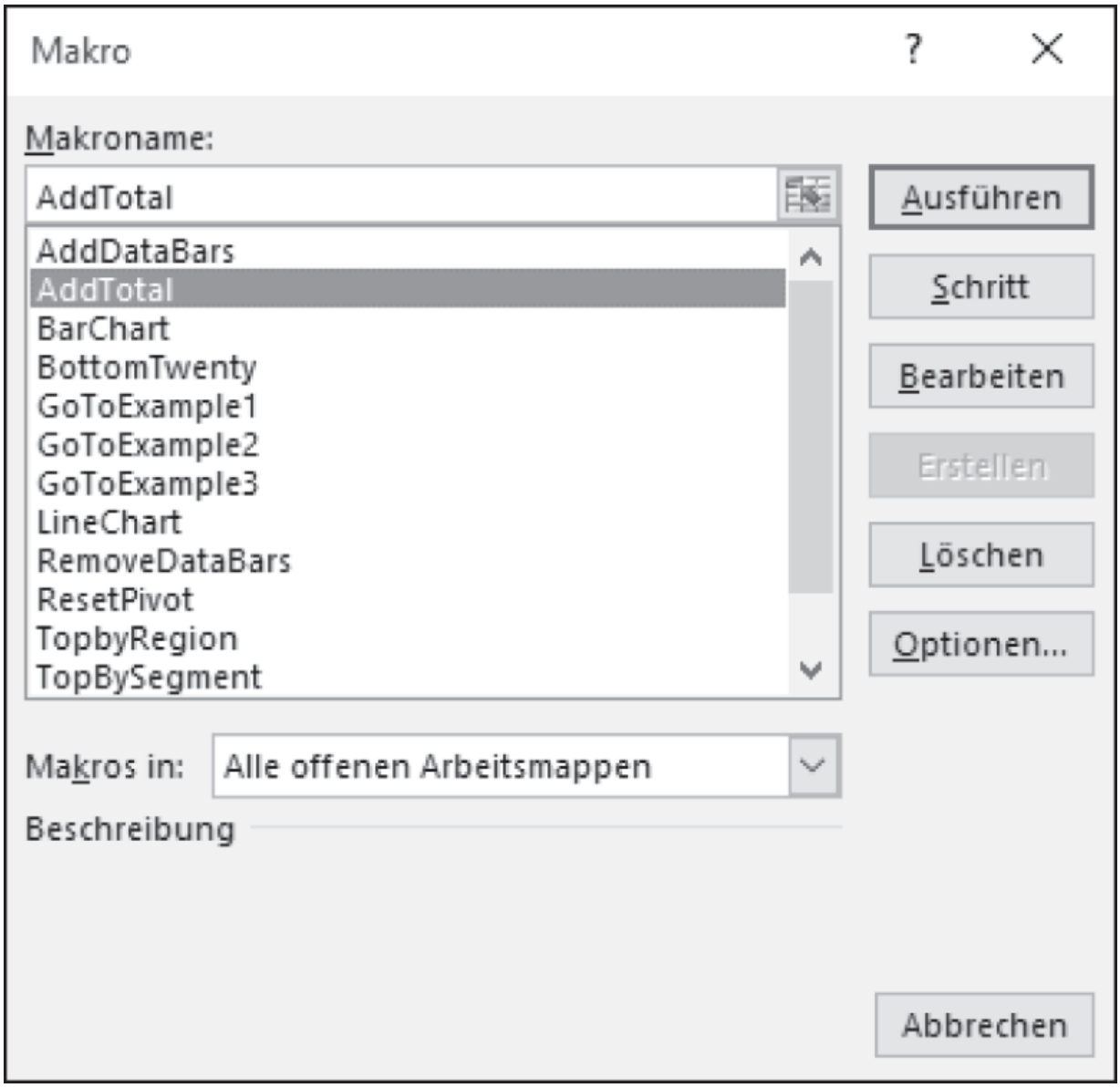


Abbildung 1.4: Das Excel-Dialogfeld »Makro«

Markieren Sie das Makro ADDTOTAL und klicken Sie auf BEARBEITEN. So wird der Visual-Basic-Editor geöffnet und zeigt den Code an, der bei der Makroaufzeichnung erstellt wurde:

```
Sub AddTotal()
 Range("A16").Select
 ActiveCell.FormulaR1C1 = "Anzahl"
 Range("D16").Select
 ActiveCell.FormulaR1C1 = "=COUNTA(R[-14]C:R[-1]C)"
End Sub
```

Bitte beachten Sie, dass Sie bei der Makroaufzeichnung die deutschen Namen der Formeln verwenden müssen. Während der Aufnahme konvertiert Excel die deutschen Formelnamen automatisch in die englischen. Daher wurde aus der Eingabe *ANZAHL2* im aufgezeichneten Makro *COUNTA*. Eine Übersicht mit den deutschen und englischen Funktionsnamen finden Sie unter anderem hier: http://excelnova.org/excel-ressourcen/excel-formeln-ubersetzt-deutsch-englisch/

Achten Sie besonders auf die Zeilen 2 und 4 des Makros: Wenn Sie Excel auffordern, die Zellen A16 und D16 auszuwählen, werden exakt diese Zellen ausgewählt. Da das Makro im Modus *absolute Verweise* aufgezeichnet wurde, interpretiert Excel die Zell- und Bereichsangaben als absolut. Anders ausgedrückt: Egal, welche Zelle Sie vor dem Starten des Makros auswählen, Excel wird im Makro immer die Zellen A16 und D16 verwenden. Im nächsten Abschnitt werden Sie sehen, wie das gleiche Makro aussieht, wenn es im relativen Verweismodus aufgezeichnet wird.

Makros mit relativen Verweisen aufzeichnen

Im Zusammenhang mit Excel-Makros bedeutet »relativ« relativ zur jeweils aktuellen Zelle. Achten Sie daher darauf, welche Zelle Sie auswählen – dies gilt sowohl für die Aufzeichnung des Makros als auch für seine Ausführung.

Öffnen Sie die Beispieldatei *1.1 Beispiele.xlsm*, falls sie nicht bereits geöffnet ist. Führen Sie dann folgende Schritte durch, um ein Makro im relativen Verweismodus aufzuzeichnen:

Weitere Informationen über den Download der Beispieldateien finden Sie in der Einleitung dieses Buchs.

1. **Klicken Sie auf der Registerkarte ENTWICKLERTOOLS in der Gruppe CODE auf RELATIVE VERWEISE VERWENDEN (siehe Abbildung 1.5).**
2. **Achten Sie darauf, dass Zelle A1 markiert ist.**
3. **Klicken Sie auf der Registerkarte ENTWICKLERTOOLS auf MAKRO AUFZEICHNEN.**
4. **Nennen Sie das Makro `AddTotalRelative`.**
5. **Legen Sie als Speicherort DIESE ARBEITSMAPPE fest.**
6. **Klicken Sie auf OK, um die Aufzeichnung zu starten.**
   Von diesem Punkt an zeichnet Excel Ihre Aktionen auf.

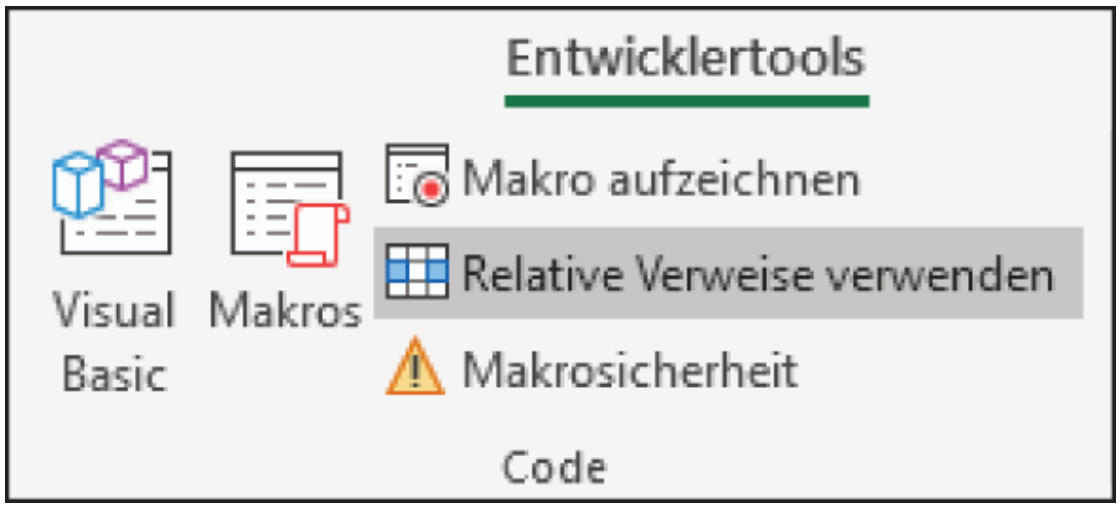


Abbildung 1.5: Aufzeichnung eines Makros mit relativen Verweisen

7. **Markieren Sie Zelle A16 und geben Sie dort `Anzahl` ein.**
8. **Markieren Sie die erste leere Zelle in Spalte D (D16) und geben Sie `=ANZAHL2(D2:D15)` ein.**
9. **Klicken Sie auf der Registerkarte ENTWICKLERTOOLS auf AUFZEICHNUNG BEENDEN, um die Makroaufzeichnung zu beenden.**

Sie haben nun zwei verschiedene Makros aufgezeichnet. Schauen Sie sich den Code des zweiten Makros an.

Klicken Sie auf der Registerkarte ENTWICKLERTOOLS auf MAKROS, um das Dialogfeld MAKRO zu öffnen. Wählen Sie hier das Makro ADDTOTALRELATIVE aus und klicken Sie auf BEARBEITEN.

Wiederum wird der Visual-Basic-Editor geöffnet und Sie sehen den Code, der während der Makroaufzeichnung geschrieben wurde. Diesmal sieht der VBA-Code folgendermaßen aus:

```
Sub AddTotalRelative()
 ActiveCell.Offset(15, 0).Range("A1").Select
 ActiveCell.FormulaR1C1 = "Total"
 ActiveCell.Offset(0, 3).Range("A1").Select
 ActiveCell.FormulaR1C1 = "=COUNTA(R[-14]C:R[-1]C)"
End Sub
```

Beachten Sie, dass der Code keine absoluten Zellbezüge aufweist — bis auf die Zelle A1, den Startpunkt des Makros. Lassen Sie uns eine kurzen Blick darauf werfen, was die relevanten Teile des Codes genau bedeuten.

Beachten Sie, dass Excel in Zeile 2 die Eigenschaft `Offset` der aktiven Zelle verwendet. Diese Eigenschaft weist die Markierung an, sich von der aktiven Zelle aus eine bestimmte Anzahl von Zellen nach oben beziehungsweise unten oder nach links beziehungsweise rechts zu bewegen.

Die `Offset`-Eigenschaft weist Excel an, die Markierung von der aktiven Zelle (in diesem Fall A1) aus um 15 Zeilen nach unten zu bewegen und bei der Spalte keine Änderung vorzunehmen (0). Es ist nicht erforderlich, dass Excel eine Zelle auswählt, da wir ein Makro mit relativen Verweisen aufgezeichnet haben.

Um das Makro auszuprobieren, löschen Sie die Zeile mit der Anzahl für beide Tabellen und machen dann Folgendes:

1. **Markieren Sie Zelle A1.**
2. **Klicken Sie auf der Registerkarte ENTWICKLERTOOLS auf MAKROS.**

3. **Suchen Sie nach dem Makro ADDTOTALRELATIVE.**
4. **Klicken Sie auf AUSFüHREN.**
5. **Markieren Sie nun Zelle F1.**
6. **Klicken Sie auf der Registerkarte ENTWICKLERTOOLS auf MAKROS.**
7. **Suchen Sie nach dem Makro ADDTOTALRELATIVE.**
8. **Klicken Sie auf AUSFüHREN.**

Beachten Sie, dass dieses Makro, anders als das vorige, mit beiden Datensätzen funktioniert. Da das Makro die Anzahl relativ zur aktuellen Zelle berechnet, liefert es für beide Tabellen das richtige Ergebnis.

Damit das Makro funktioniert, müssen Sie lediglich Folgendes gewährleisten:

- ✔ Vor der Ausführung des Makros müssen Sie die richtige Startzelle auswählen.
- ✔ Der Datenblock muss die gleiche Anzahl von Zeilen und Spalten aufweisen wie die Daten, mit denen Sie das Makro aufgezeichnet haben.

Hoffentlich hat dieses einfache Beispiel die Unterschiede zwischen der Makroaufzeichnung mit absoluten Zellbezügen und der mit relativen Verweisen deutlich gemacht.

Weitere Konzepte für die Makroaufzeichnung

Mit den Informationen aus den vorigen Abschnitten können Sie nun Ihre ersten eigenen Makros aufzeichnen. In diesem Abschnitt werden weitere wichtige Konzepte vorgestellt, die Sie beim Arbeiten mit Makros beachten sollten.

Dateierweiterungen für Arbeitsmappen mit Makros

Mit der Einführung von Excel 2007 hat Microsoft das Format von Excel-Dateien und deren Standard-Dateinamenserweiterung von .xls in .xlsx geändert. Dateien mit der Erweiterung .xlsx können allerdings keine Makros enthalten. Wenn Ihre Arbeitsmappe Makros enthält und Sie die Mappe als .xslx-Datei abspeichern, werden die Makros automatisch entfernt. Excel weist Sie darauf hin, dass die Makroinhalte entfernt werden, wenn Sie eine Arbeitsmappe mit Makros als .xlsx-Datei speichern.

Um die Makros zu sichern, müssen Sie die Datei im Format *Arbeitsmappe mit Makros* speichern. Hierdurch erhält die Datei die Erweiterung .xslm. Die Idee dahinter ist, dass alle Arbeitsmappen mit der Erweiterung .xlsx als sicher vor potenziell schädlichen Makros eingestuft werden können. Dateien mit der Erweiterung .xlsm hingegen stellen wegen möglicher Makro-Viren eine potenzielle Gefahr dar.

Makrosicherheit ab Excel 2010

Mit der Veröffentlichung von Office 2010 hat Microsoft erhebliche Änderungen am Office-Sicherheitsmodell vorgenommen. Eine der wesentlichen Änderungen ist das Konzept der vertrauenswürdigen Dokumente. Ohne in die technischen Details eintauchen zu wollen, ist ein vertrauenswürdiges Dokument in Excel eine Arbeitsmappe, die Sie als sicher eingestuft haben, indem Sie die Makros aktiviert haben.

Wenn Sie in Excel 2010 oder neuer eine Arbeitsmappe öffnen, die Makros enthält, sehen Sie unter dem Menüband eine gelbe Sicherheitswarnung, die Sie darauf hinweist, dass aktive Inhalte (Makros) zunächst einmal deaktiviert wurden.

Wenn Sie auf INHALT AKTIVIEREN klicken, wird diese Arbeitsmappe automatisch zu einem vertrauenswürdigen Dokument. Das bedeutet, dass Sie die Sicherheitswarnung nicht mehr sehen, wenn Sie die gleiche Datei erneut auf Ihrem

Computer öffnen. Die Idee dahinter ist, dass wenn Sie Excel einmal anweisen, einer bestimmten Arbeitsmappe zu vertrauen und die Ausführung von Makros zulassen, es sehr wahrscheinlich ist, dass Sie die Makroausführung bei jedem weiteren Öffnen dieser Arbeitsmappe wieder zulassen werden. Excel merkt sich also, dass Sie die Makros für diese Arbeitsmappe zugelassen haben und unterdrückt für sie in der Folge die Warnung zu den Makros.

Dies ist für Sie und Ihre Kunden eine gute Nachricht. Nachdem die Makroausführung lediglich einmal zugelassen wurde, werden Ihre Kunden nicht immer wieder erneut von der Sicherheitswarnung belästigt. Und Sie brauchen sich keine Sorgen darüber zu machen, dass Ihr mit Makros ausgestattetes Excel-Dashboard nicht funktioniert, weil die Makroausführung deaktiviert wurde.

Vertrauenswürdige Speicherorte

Falls Sie der Meinung sind, dass die auftauchenden Makro-Sicherheitswarnungen Sie nerven (auch wenn sie nur einmal angezeigt werden), können Sie für Ihre Dateien einen vertrauenswürdigen Speicherort festlegen. Ein *vertrauenswürdiger Speicherort* ist ein Ordner, der als sichere Zone eingestuft wird, weil dort lediglich vertrauenswürdige Arbeitsmappen gespeichert sind. Ein vertrauenswürdiger Speicherort erlaubt es Ihnen und Ihren Kunden, Arbeitsmappen mit Makros zu öffnen, die keinen Sicherheitsbeschränkungen unterliegen, solange sich die Arbeitsmappen in diesem Ordner befinden.

Um einen vertrauenswürdigen Speicherort festzulegen, führen Sie folgende Schritte durch:

1. **Klicken Sie auf der Registerkarte ENTWICKLERTOOLS auf MAKROSICHERHEIT.**

   Das Dialogfeld TRUST CENTER wird geöffnet.

2. **Klicken Sie an der linken Seite auf VERTRAUENSWüRDIGE SPEICHERORTE.**

An der rechten Seite sehen Sie die Einstellungen für die vertrauenswürdigen Speicherorte (siehe Abbildung 1.6). Die obere Liste enthält alle Ordner, die bereits als vertrauenswürdig eingestuft wurden.

3. **Klicken Sie auf NEUEN SPEICHERORT HINZUFüGEN.**
4. **Klicken Sie auf DURCHSUCHEN und öffnen Sie den Ordner, der als vertrauenswürdiger Speicherort gelten soll.**

Nachdem Sie einen vertrauenswürdigen Speicherort festgelegt haben, werden die Makros der Arbeitsmappen, die Excel aus diesem Ordner öffnet, automatisch aktiviert.

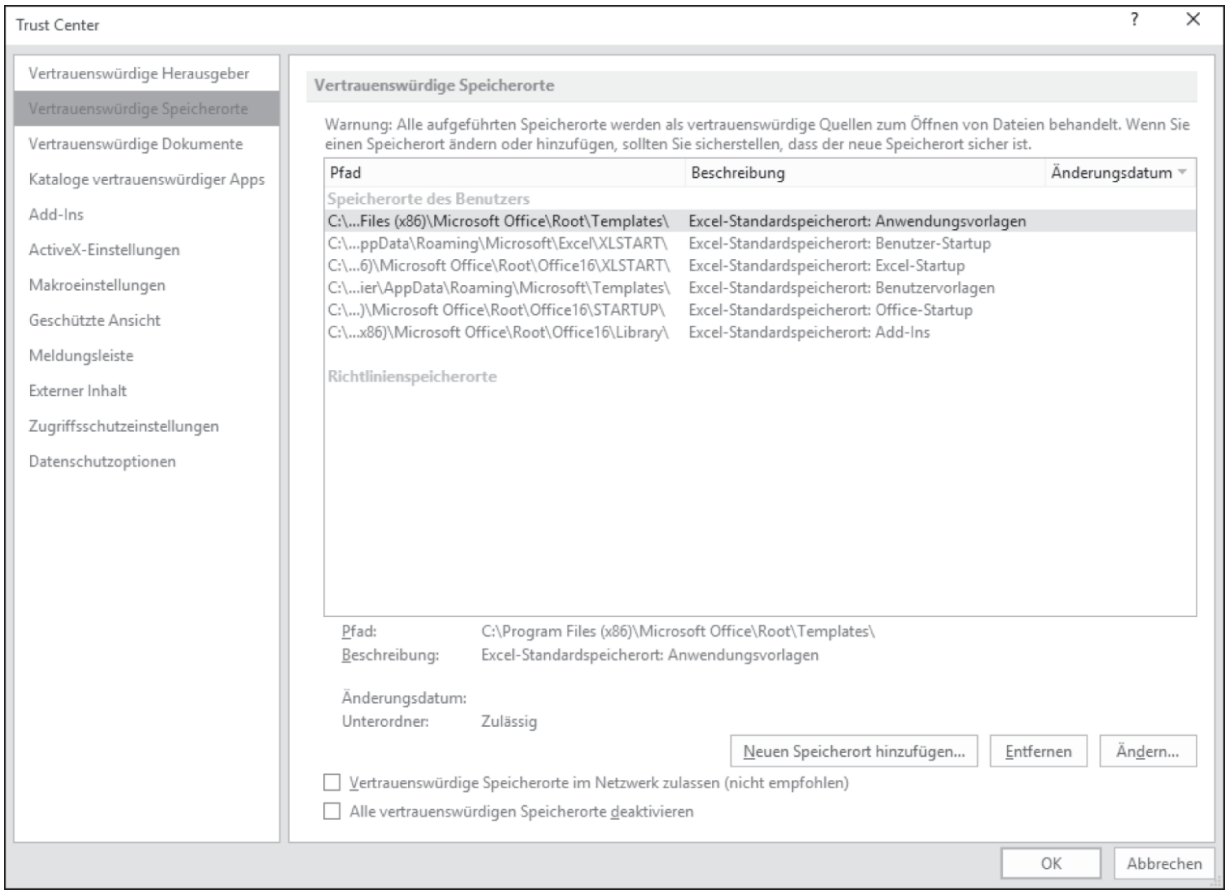


Abbildung 1.6: Fügen Sie hier Speicherorte ein, die als vertrauenswürdig gelten sollen.

Makros in Ihrer persönlichen Makroarbeitsmappe abspeichern

Die meisten benutzerdefinierten Makros sind für die Verwendung in einer bestimmten Arbeitsmappe vorgesehen. Es kann aber auch Makros geben, die Sie in mehreren oder sogar in allen Arbeitsmappen einsetzen wollen. Sie können diese Makros in Ihrer persönlichen Makroarbeitsmappe speichern, damit sie Ihnen immer zur Verfügung stehen. Die persönliche Makroarbeitsmappe wird beim Start von Excel automatisch geöffnet. Diese Datei mit dem Namen *personal.xlsb* wird automatisch erstellt, wenn Sie das erste Mal ein Makro in dieser speziellen Arbeitsmappe aufzeichnen.

Um ein Makro in der persönlichen Makroarbeitsmappe abzuspeichern, wählen Sie im Dialogfeld MAKRO AUFZEICHNEN in der Liste MAKRO SPEICHERN IN die Option PERSöNLICHE MAKROARBEITSMAPPE aus (siehe Abbildung 1.1).

Wenn Sie Makros in der persönlichen Makroarbeitsmappe speichern, brauchen Sie nicht daran zu denken, diese zu öffnen, um die dort enthaltenen Makros in einer »normalen« Arbeitsmappe einzusetzen: Excel öffnet die persönliche Makroarbeitsmappe automatisch beim Start des Programms. Wenn Sie Excel beenden, werden Sie gefragt, ob die Änderungen in der persönlichen Makroarbeitsmappe gespeichert werden sollen.


Die persönliche Makroarbeitsmappe befindet sich in einem Fenster, das ausgeblendet ist, damit es nicht im Weg ist.

Ein Makro einer Schaltfläche oder einem anderen Formularsteuerelement zuweisen

Nachdem Sie Makros erstellt haben, brauchen Sie noch einen offensichtlichen und einfachen Weg, um die verschiedenen Makros zu starten. Eine Schaltfläche auf einem Tabellenblatt bietet eine solche einfache und gleichzeitig effektive Benutzeroberfläche.

Wie es der Zufall will, stellt Ihnen Excel eine Reihe von Formularsteuerelementen zur Verfügung, mit denen Sie direkt auf einem Tabellenblatt Benutzeroberflächen erstellen können. Ihnen stehen verschiedene Steuerelemente zur Verfügung, angefangen von Schaltflächen (die am häufigsten verwendet werden) bis hin zu Bildlaufleisten.

Das Konzept für die Verwendung von Formularsteuerelementen ist einfach. Sie fügen ein Formularsteuerelement in ein Tabellenblatt ein und weisen ihm dann ein Makro zu – beispielsweise das Makro, das Sie vorhin aufgezeichnet haben. Wenn Sie es einem Steuerelement zugewiesen haben und das Steuerelement dann anklicken, wird das zugeordnete Makro gestartet/ausgeführt.

Erstellen Sie nun eine Schaltfläche für das Makro `AddTotalRelative`, das Sie weiter oben erstellt haben. Das geht so:

1. **Klicken Sie auf der Registerkarte ENTWICKLERTOOLS auf EINFüGEN (siehe Abbildung 1.7).**
2. **Klicken Sie im Bereich FORMULARSTEUERELEMENTE auf SCHALTFLäCHE.**
3. **Klicken Sie im Tabellenblatt die Stelle an, an der Sie die Schaltfläche platzieren wollen.**

   Nachdem Sie im Tabellenblatt eine Stelle angeklickt haben, wird das Dialogfeld MAKRO ZUWEISEN angezeigt (siehe Abbildung 1.8).
4. **Wählen Sie das Makro aus, das Sie der Schaltfläche zuweisen wollen, und klicken Sie auf OK.**

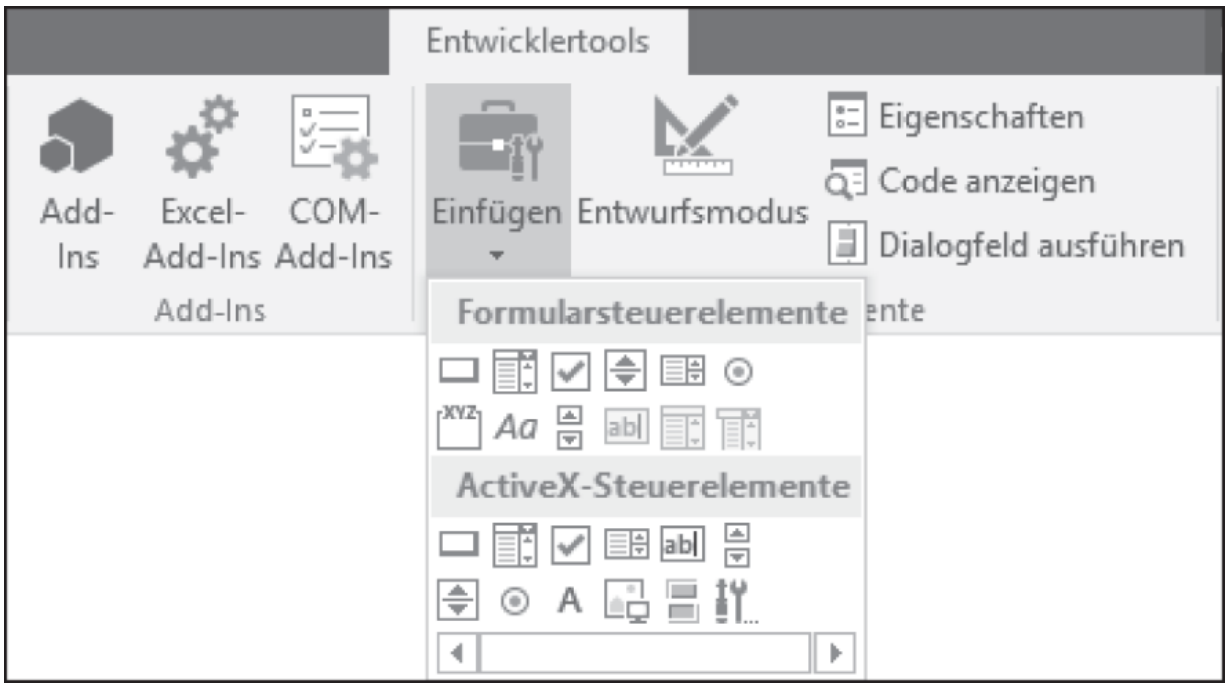


Abbildung 1.7: Sie finden die Formularsteuerelemente auf der Registerkarte »Entwicklertools«.

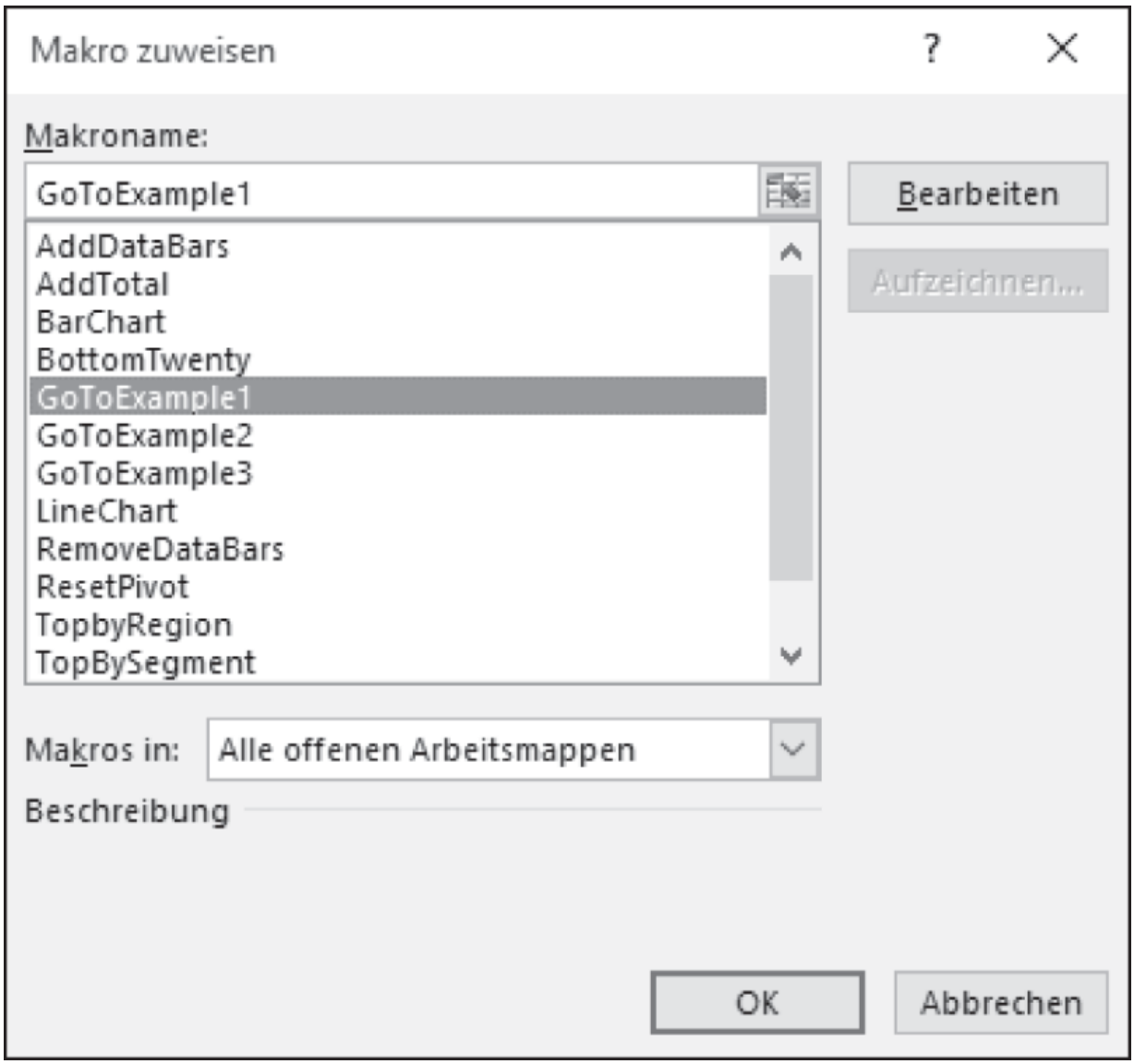


Abbildung 1.8: Der neu hinzugefügten Schaltfläche ein Makro zuweisen

Sie haben nun eine Schaltfläche erstellt, die Ihr Makro ausführt, wenn Sie sie anklicken. Beachten Sie, dass alle Steuerelemente im Bereich FORMULARSTEUERELEMENTE auf die gleiche Weise funktionieren wie die Schaltfläche, die wir im Beispiel verwendet haben. Sie können also ein Makro einem Steuerelement zuweisen; das Makro wird gestartet, wenn das Steuerelement angewählt wird.

Formularsteuerelemente vs. ActiveX-Steuerelemente

Werfen Sie in Abbildung 1.7 einen Blick auf die Bereiche FORMULARSTEUERELEMENTE und ACTIVEX-STEUERELEMENTE. Auch wenn die Steuerelemente beider Gruppen sehr ähnlich aussehen, so funktionieren sie doch sehr unterschiedlich. Formularsteuerelemente sind für den Einsatz auf einem Tabellenblatt konzipiert; ActiveX-Steuerelemente werden üblicherweise in sogenannten Excel-User-Forms verwendet. Allgemein gilt: Verwenden Sie immer Formularsteuerelemente, wenn Sie Steuerelemente auf einem Tabellenblatt einsetzen wollen. Formularsteuerelemente erfordern weniger Overhead, werden schneller ausgeführt und sie lassen sich viel einfacher konfigurieren als die entsprechenden ActiveX-Steuerelemente.

Ein Makro in die Symbolleiste für den Schnellzugriff einfügen

Sie können ein Makro auch einer Schaltfläche in der Symbolleiste für den Schnellzugriff zuweisen. Die Symbolleiste für den Schnellzugriff befindet sich entweder oberhalb oder unterhalb des Menübands. Führen Sie folgende Schritte durch, um dort für ein Makro eine benutzerdefinierte Schaltfläche zu erstellen:

1. **Klicken Sie die Symbolleiste für den Schnellzugriff mit der rechten Maustaste an und wählen Sie PASSEN SIE DIE SYMBOLLEISTE FüR DEN SCHNELLZUGRIFF AN.**

   Das Dialogfeld aus Abbildung 1.9 wird angezeigt.

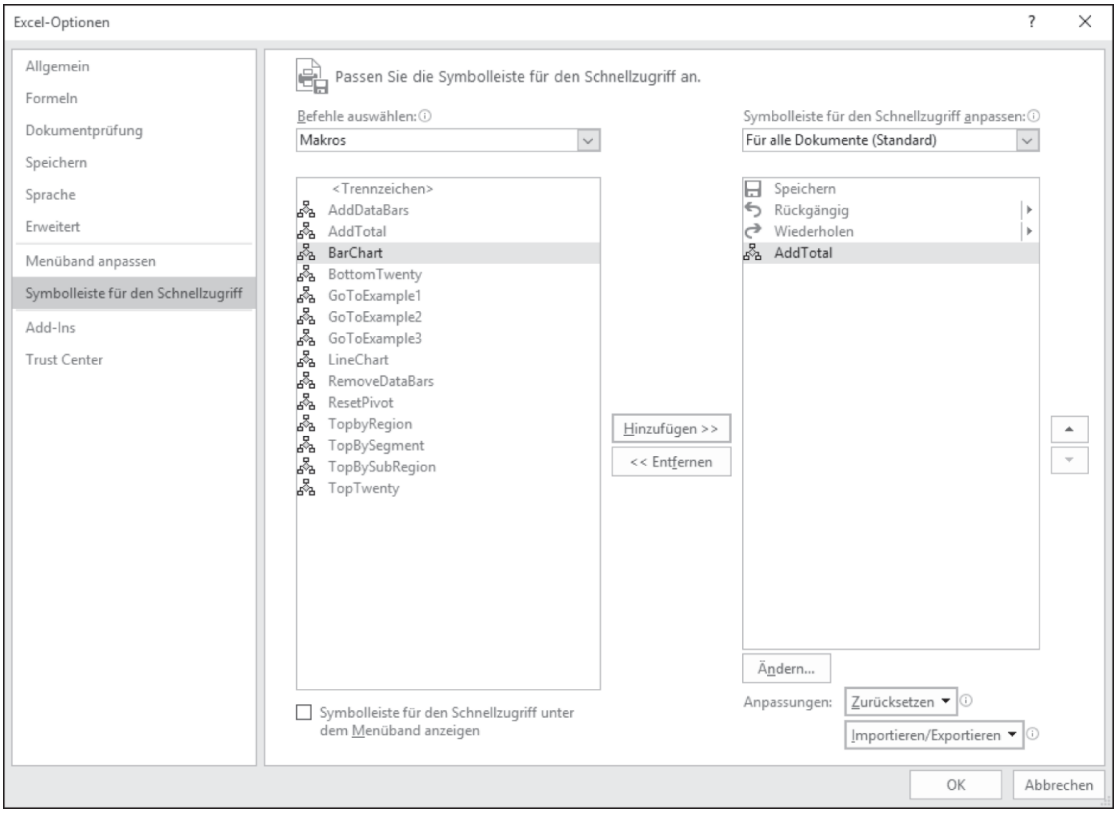


Abbildung 1.9: Ein Makro in die Symbolleiste für den Schnellzugriff einfügen

2. **Klicken Sie auf der linken Seite des Dialogfelds EXCEL-OPTIONEN auf die Kategorie SYMBOLLEISTE FüR DEN SCHNELLZUGRIFF.**
3. **Wählen Sie in der Drop-down-Liste BEFEHLE AUSWäHLEN den Eintrag MAKROS aus.**
4. **Wählen Sie das Makro aus, das Sie in die Symbolleiste für den Schnellzugriff einfügen wollen, und klicken Sie auf HINZUFüGEN.**
5. **Klicken Sie auf ÄNDERN, um das Symbol der Schaltfläche festzulegen.**

Makros im Einsatz – Beispiele

Die Grundlagen für das Erstellen und die Verwendung von Makros zu beschreiben, ist leider nur die halbe der Miete. Genauso wichtig ist es, gute Verfahren zu finden, wie Sie die

Makros beispielsweise beim Erstellen von Geschäftsberichten einsetzen können. Nehmen Sie sich einen Moment Zeit und schauen Sie sich ein paar Beispiele dafür an, wie sich mit Makros das Erstellen von Geschäftsberichten vereinfachen lässt.

Um die Beispiele dieses Abschnitts in der Praxis zu sehen, laden Sie die Beispieldateien zu diesem Buch herunter (siehe Einleitung) und öffnen die Datei *1.1 Beispiele.xlsm*.

Navigationsschaltflächen erstellen

Makros werden sehr häufig für die Navigation in Arbeitsmappen eingesetzt. Es kann etwas frustrierend sein, sich in Arbeitsmappen mit vielen Tabellenblättern beziehungsweise Registerkarten zu bewegen. Um die Benutzer zu unterstützen, können Sie so etwas wie ein Hauptmenü erstellen, ähnlich wie es Abbildung 1.10 zeigt. Klickt der Benutzer die Schaltfläche BEISPIEL 1 an, wird das entsprechende Tabellenblatt geöffnet.

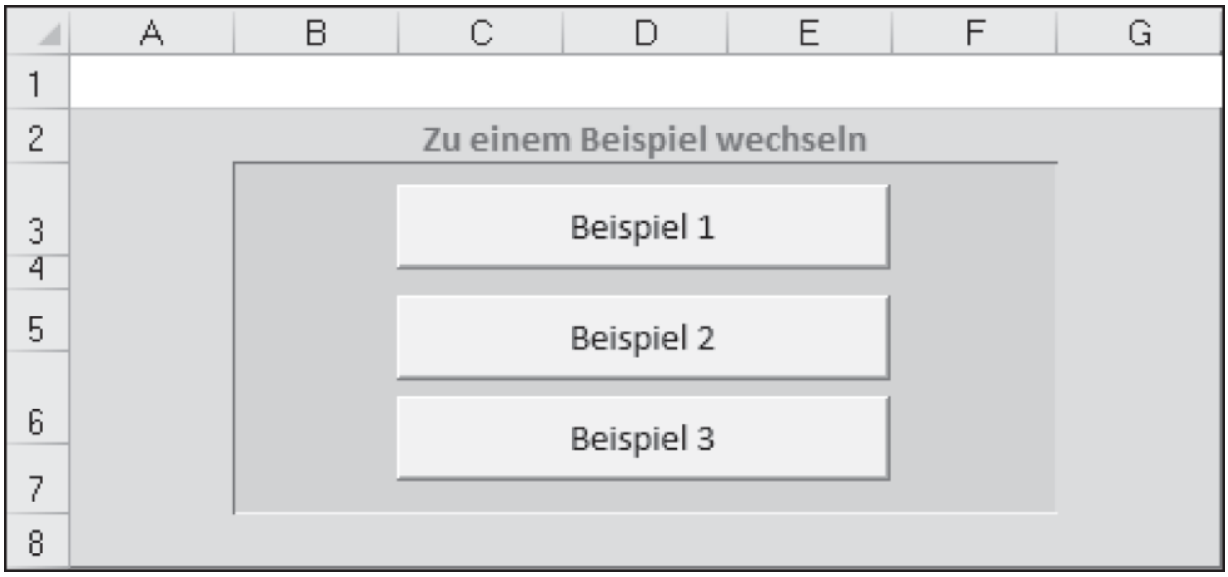


Abbildung 1.10: Verwenden Sie Makros, um Schaltflächen zu erstellen, mit denen die Benutzer zwischen Berichten navigieren können.

Das Erstellen eines Makros, mit dem Sie zu einem bestimmten Tabellenblatt navigieren können, ist ganz einfach:

1. **Beginnen Sie auf dem Tabellenblatt, das als eine Art Hauptmenü dienen soll.**
2. **Starten Sie die Makroaufzeichnung.**
3. **Klicken Sie während der Aufzeichnung das Tabellenblatt an, das mit dem Makro geöffnet werden soll.**
4. **Nachdem Sie das Zieltabellenblatt angeklickt haben, beenden Sie die Aufzeichnung.**
5. **Weisen Sie das Makro einer Formularschaltfläche zu.**

Es ist nützlich zu wissen, dass Excel ein integriertes Hyperlink-Feature besitzt, mit dem Sie den Inhalt einer Zelle in einen Hyperlink konvertieren können, der auf eine andere Stelle zeigt. Dies kann eine andere Excel-Arbeitsmappe, eine Website oder auch einfach ein anderes Tabellenblatt in der aktuellen Arbeitsmappe sein. Auch wenn die Verwendung eines Hyperlinks einfacher erscheint als das Einrichten eines Makros, so ist es leider nicht möglich, einem Formularsteuerelement, wie beispielsweise einer Schaltfläche, einen Hyperlink zuzuweisen. Anstelle einer Schaltfläche verwendet ein Hyperlink einfachen Text, um den Anwender über das Linkziel zu informieren.

Pivot-Tabelle-Daten dynamisch neu anordnen

Makros können mit jedem Excel-Objekt verwendet werden, das normalerweise bei der Erstellung von Berichten verwendet wird. Beispielsweise können Sie ein Makro einsetzen, um Ihren Anwendern die Möglichkeit zu geben, eine Pivot-Tabelle dynamisch zu ändern.

Das Beispiel in Abbildung 1.11 zeigt, wie Benutzer mit Makros die Perspektive eines Diagramms ändern können, indem sie einfach eine der Schaltflächen anklicken.

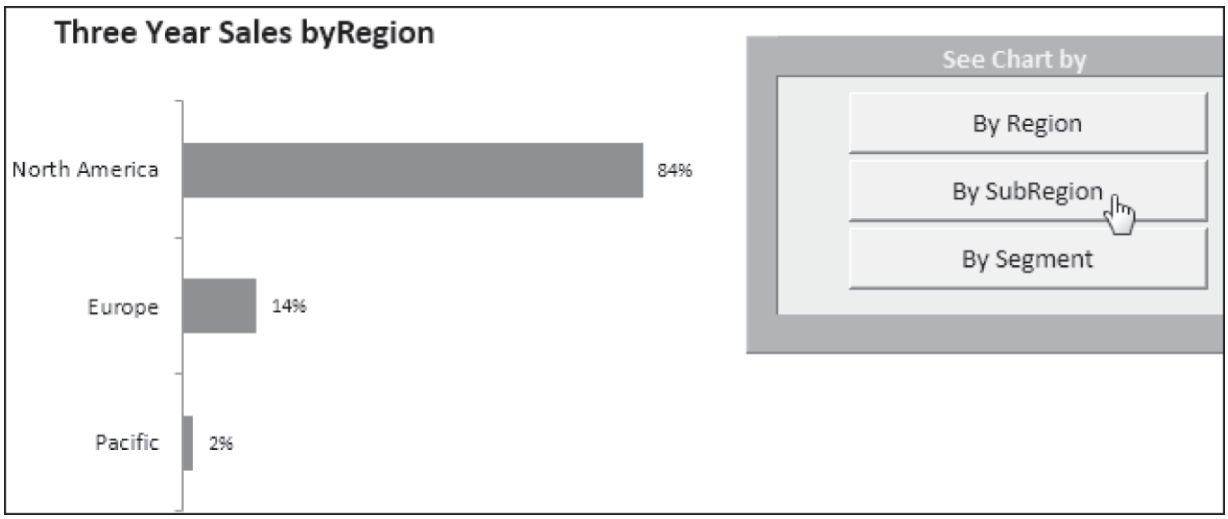


Abbildung 1.11: In diesem Bericht können die Benutzer die Perspektive des Diagramms ändern.

Abbildung 1.12 macht allerdings deutlich, dass das Diagramm eigentlich ein Pivot-Diagramm ist, dessen Daten aus einer Pivot-Tabelle stammen. Die aufgezeichneten Makros, die den Schaltflächen zugewiesen wurden, machen nichts anderes, als die Pivot-Tabelle neu anzuordnen.

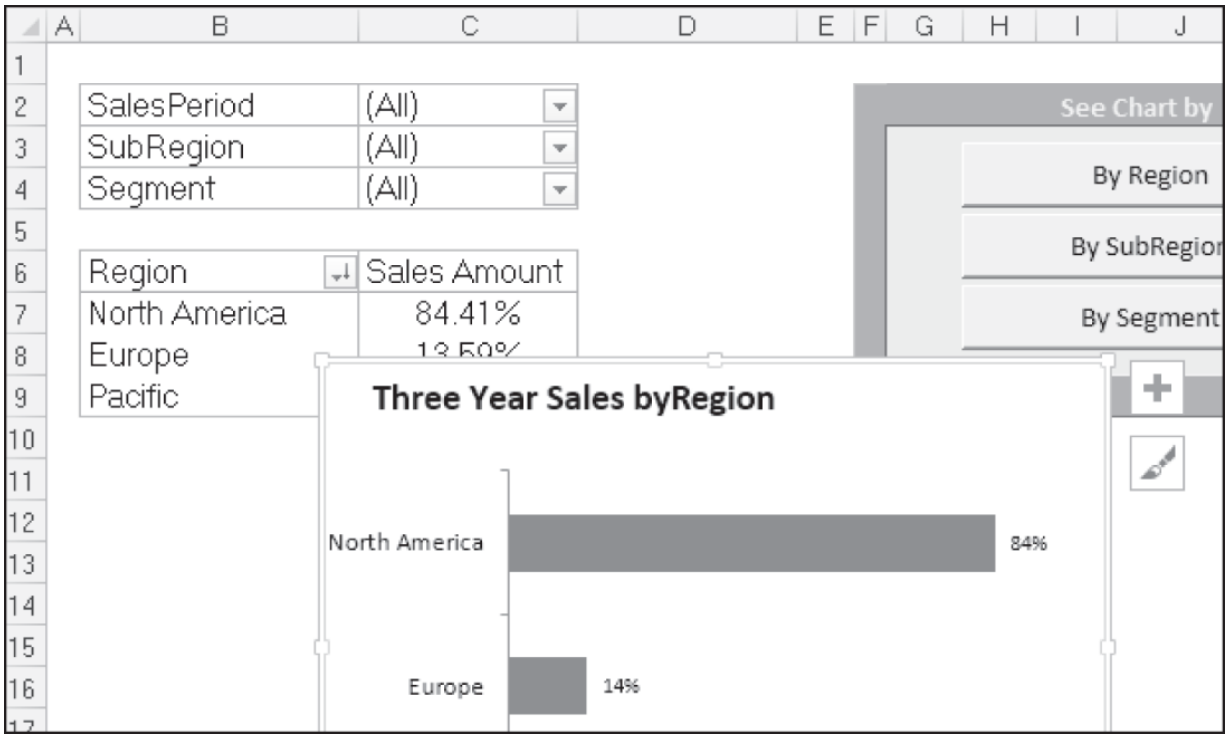


Abbildung 1.12: Die Makros hinter diesen Schaltflächen ordnen die Datenfelder der PivotTable neu an.

Dies sind die allgemeinen Schritte, mit denen Sie diese Art von Arbeitsmappe erstellen können:

1. **Erstellen Sie die PivotTable und ein Pivot-Diagramm.**
2. **Starten Sie die Makroaufzeichnung.**
3. **Verschieben Sie während der Makroaufzeichnung ein Pivot-Feld aus einem Bereich der Pivot-Tabelle in einen anderen und beenden Sie dann die Makroaufzeichnung.**
4. **Zeichnen Sie ein weiteres Makro auf, in dem das Datenfeld zurück an die Ausgangsposition verschoben wird.**
5. **Nachdem Sie die beiden Makros aufgezeichnet haben, weisen Sie sie den verschiedenen Schaltflächen zu.**

Sie können nun die aufgezeichneten Makros starten und sehen, wie das Pivot-Feld dynamisch hin- und herspringt.

Berichte mit einem Klick erstellen

Die beiden letzten Beispiele haben gezeigt, dass Sie jede Aktion aufzeichnen können, die Ihnen als Makro nützlich erscheint. Falls Sie also meinen, dass Ihre Anwender es schätzen, wenn ein bestimmtes Feature für sie automatisiert wird, warum zeichnen Sie hierfür nicht einfach ein Makro auf?

Im Beispiel in Abbildung 1.13 können Sie mit einem Mausklick eine Pivot-Tabelle filtern und die 20 umsatzstärksten und umsatzschwächsten Kunden anzeigen lassen. Da die Schritte für diesen Filter aufgezeichnet wurden, sparen die Anwender Zeit und sie können den Filter anwenden, ohne die hierfür erforderlichen Einzelschritte zu kennen. Außerdem können Sie so Fehlbedienungen vermeiden, da die Anwender auf eine sichere und getestete Weise mit Ihren Berichten interagieren.

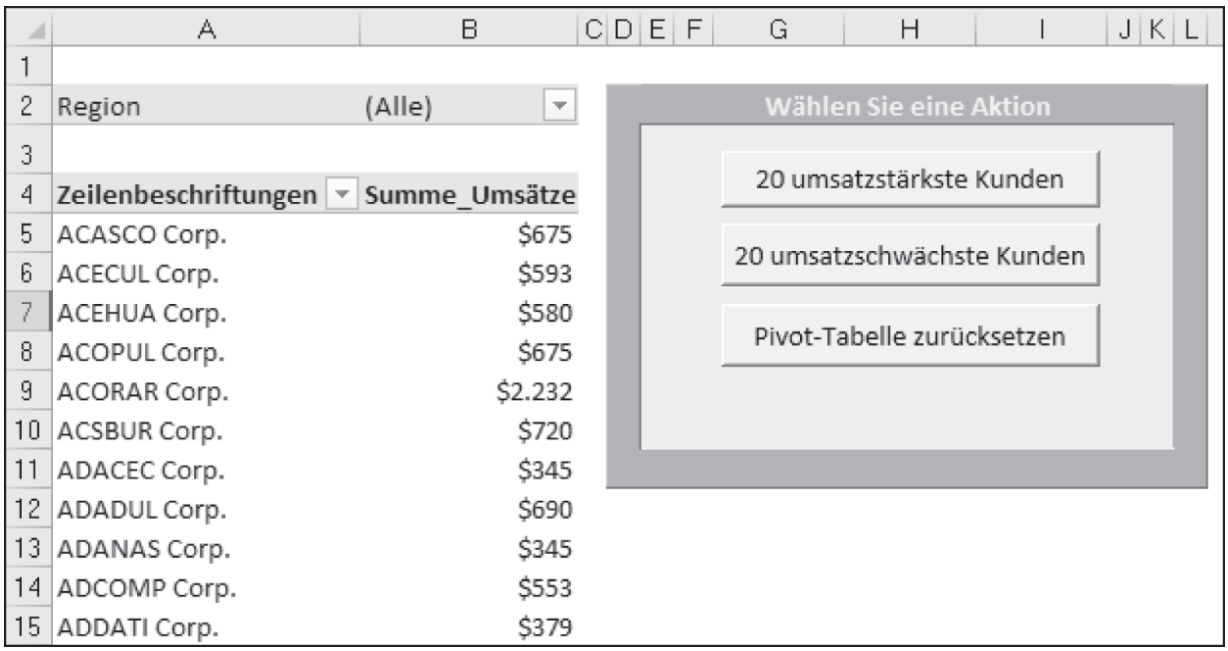


Abbildung 1.13: Vorher aufgezeichnete Ansichten zur Verfügung zu stellen, führt nicht nur zur Zeitersparnis, sondern es erlaubt es den Anwendern, fortgeschrittene Features zu verwenden, ohne alle Einzelheiten kennen zu müssen.

Dies erspart ihnen nicht nur Zeit und Mühe, sondern lässt auch Anwender davon profitieren, die nicht wissen, wie sie diese Aktionen durchführen müssen.

In Abbildung 1.14 sehen Sie ein Beispiel dafür, wie Ihre Anwender sich die gleichen Daten schnell in unterschiedlichen Diagrammtypen ansehen können. Lachen Sie nicht, weil Sie möglicherweise meinen, dass dieses Beispiel viel zu einfach und daher sinnlos ist. Es ist nicht unüblich, dass Excel-Benutzer die gleichen Daten auf verschiedene Weisen darstellen wollen.

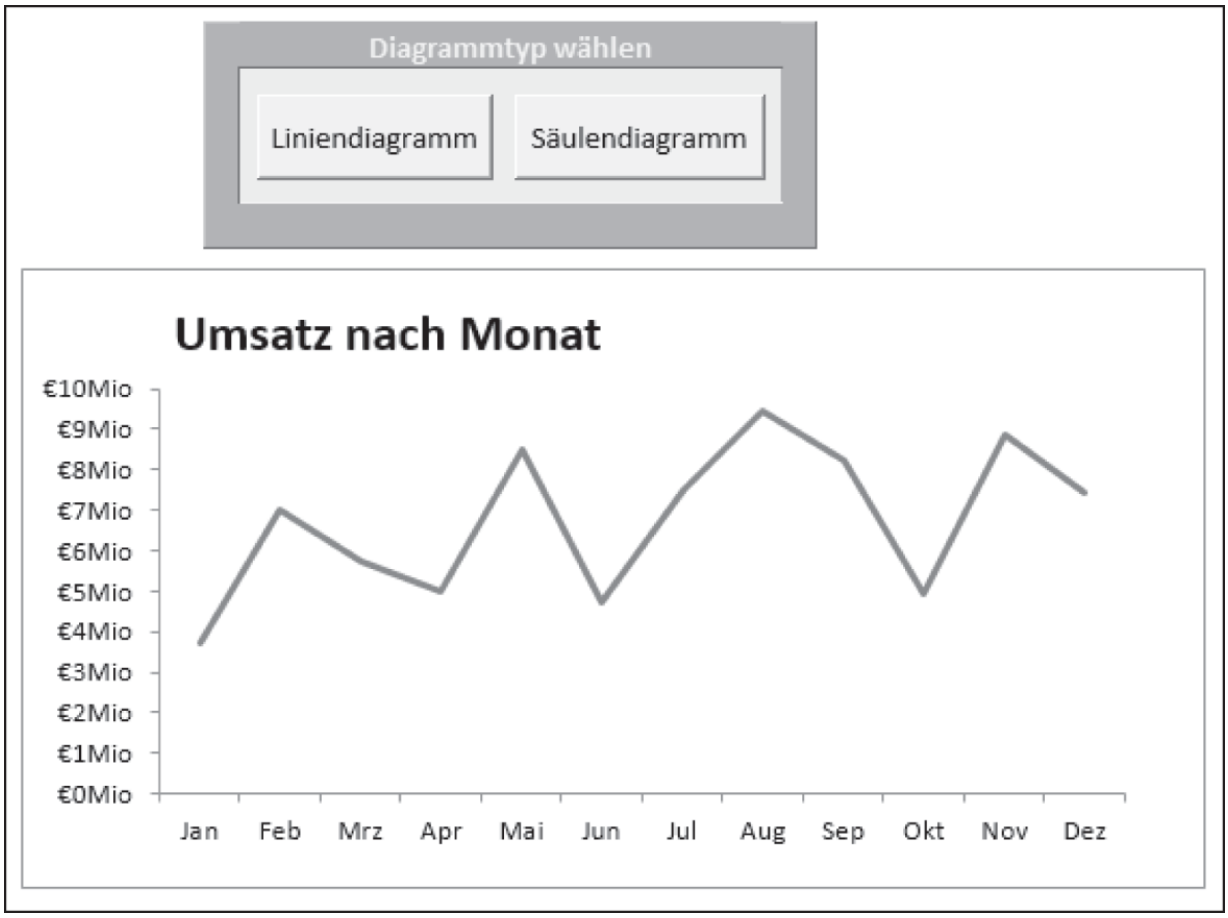


Abbildung 1.14: Sie können Ihren Anwendern die Entscheidung darüber überlassen, welcher Diagrammtyp zur Darstellung der Daten gewählt werden soll.

Anstatt viel Platz für mehrere Diagramme zu verwenden, zeichnen Sie einfach ein Makro auf, das den Diagrammtyp ändert. Ihre Kunden können dann die Daten so betrachten, wie es ihnen gerade am sinnvollsten erscheint.

Kapitel 2

Machen Sie sich mit dem Visual-Basic-Editor vertraut

IN DIESEM KAPITEL

- Die Elemente des Visual-Basic-Editors verstehen
- Mit dem Projekt-Explorer arbeiten
- Das Codefenster verwenden
- Visual-Basic-Editor anpassen

Der Visual-Basic-Editor (VBE) ist die Entwicklungsumgebung, in der alle Excel-Makros gespeichert und bearbeitet werden. Mit jeder Arbeitsmappe, die Sie erstellen, erhalten Sie automatisch und kostenlos die VBE-Umgebung, die automatisch eine Verbindung mit der jeweiligen Arbeitsmappe herstellt. Auch wenn Sie niemals ein Makro aufzeichnen, wartet der VBE im Hintergrund auf seinen Einsatz. Und wenn Sie dann ein Makro erstellen, erwacht der VBE zum Leben und kann sofort die verschiedenen Prozeduren und Makros, mit denen Sie ihn füttern, verarbeiten.

In diesem Kapitel werfen Sie einen ersten Blick hinter die Kulissen des Excel-Makro-Theaters und lernen dabei den Visual-Basic-Editor kennen.

Im Visual-Basic-Editor arbeiten

Der Visual-Basic-Editor ist genau genommen eine eigenständige Anwendung, die gestartet wird, wenn Sie Excel öffnen. Um die verborgene VBE-Umgebung zu sehen, müssen Sie sie aktivieren.

Der schnellste Weg dafür ist die Tastenkombination Alt+F11, die Sie drücken können, wenn Excel aktiv ist. Um von der VBE-Umgebung zu Excel zurückzukehren, drücken Sie ein weiteres Mal Alt+F11.

Sie können den VBE auch mit dem Befehl VISUAL BASIC aktivieren, den Sie auf der Excel-Registerkarte ENTWICKLERTOOLS finden.

Abbildung 2.1 zeigt das Hauptfenster des Visual-Basic-Editors, einige Schlüsselelemente sind gekennzeichnet. Es kann auch sein, dass das Fenster auf Ihrem Bildschirm nicht exakt so aussieht wie das in der Abbildung. VBE besteht aus mehreren Fenstern und lässt sich außerdem an Ihre Bedürfnisse anpassen. Sie können Fenster ein- und ausblenden, ihre Anordnung verändern, Fenster andocken und so weiter.

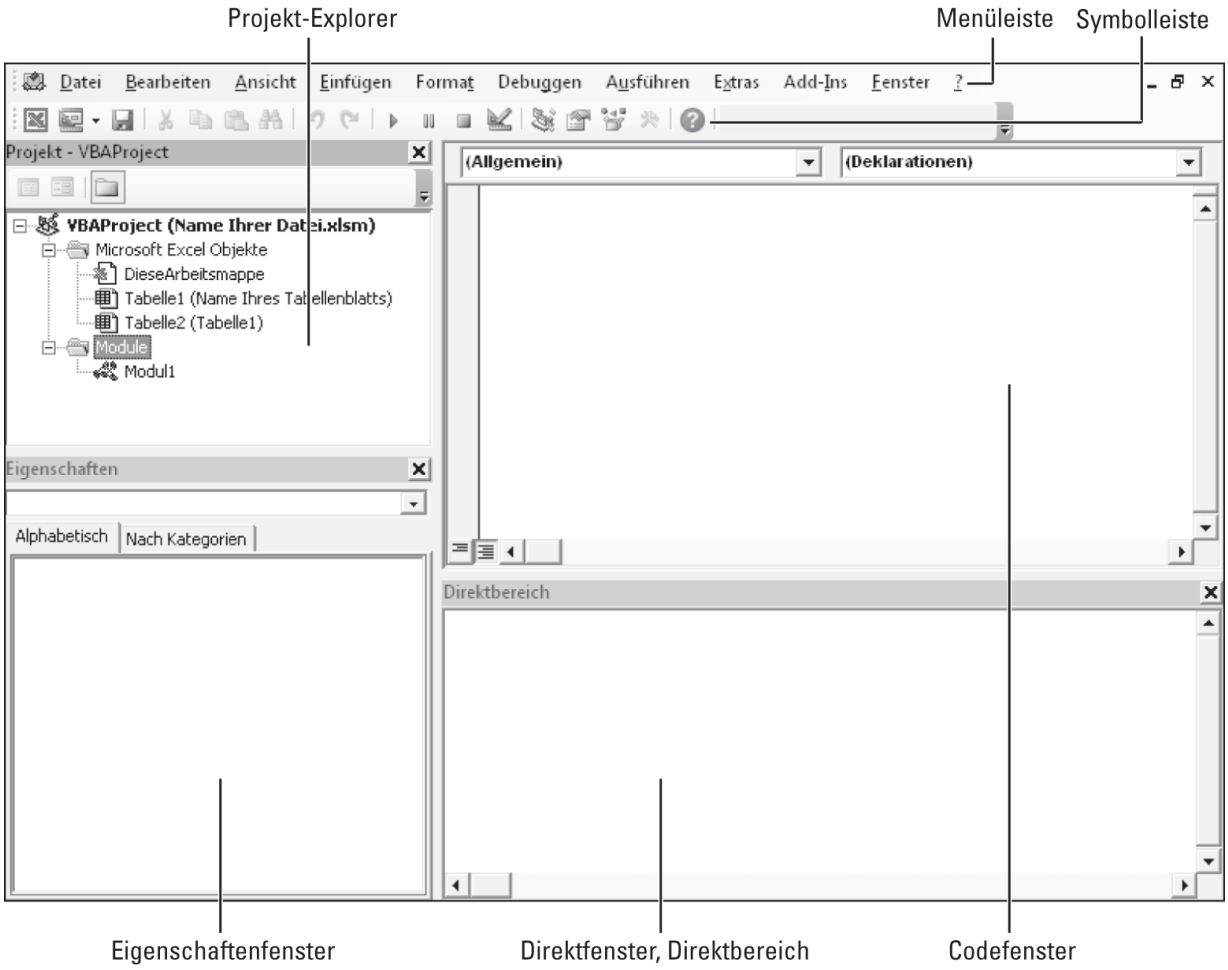


Abbildung 2.1: Die wichtigsten Elemente der Entwicklungsumgebung Visual-Basic-Editor

Die VBE-Menüleiste

Die VBE-*Menüleiste* funktioniert genauso wie die Menüleisten in anderen Programmen, die Sie bereits kennen. Im Menü finden Sie die Befehle, mit denen Sie die verschiedenen VBE-Komponenten steuern und bedienen können. Viele Menübefehle können alternativ auch mit Tastenkombinationen aufgerufen werden.

Der VBE enthält außerdem die sogenannten Kontextmenüs. Sie können fast jede beliebige Stelle im VBE mit der rechten Maustaste anklicken und damit ein Menü mit den wichtigsten Befehlen öffnen.

Die VBE-Symbolleiste

Die Standardsymbolleiste, die normalerweise direkt unterhalb der Menüleiste angezeigt wird, ist eine von vier Symbolleisten, die der VBE zur Verfügung stellt. Sie können die Symbolleisten anpassen, sie verschieben, andere Symbolleisten anzeigen lassen und so weiter. Falls Sie dies tun möchten, wählen Sie ANSICHT | MENüLEISTEN, um die betreffenden Befehle zu sehen. Die meisten Leute lassen die Symbolleisten jedoch so, wie sie sind.

Der Projekt-Explorer

Im Projekt-Explorer sehen Sie ein Baumdiagramm, das alle derzeit in Excel geöffneten Arbeitsmappen anzeigt, einschließlich der Add-ins und der ausgeblendeten Arbeitsmappen. Doppelklicken Sie auf ein Element, um es zu erweitern beziehungsweise zu reduzieren. Sie lernen den Projekt-Explorer ausführlicher im Abschnitt »Mit dem Projekt-Explorer arbeiten« weiter hinten in diesem Kapitel kennen.

Falls der Projekt-Explorer nicht sichtbar ist, drücken Sie Strg+R oder wählen Sie ANSICHT | PROJEKT-EXPLORER. Um den Projekt-Explorer zu schließen, klicken Sie in der Titelleiste die SCHLIESSEN-Schaltfläche (mit dem X) an. Alternativ klicken Sie

mit der rechten Maustaste eine beliebige Stelle im Projekt-Explorer an, und wählen Sie im Kontextmenü AUSBLENDEN.

Das Codefenster

Ein Codefenster enthält VBA-Code. Jedem Objekt eines Projekts ist ein Codefenster zugeordnet, in dem sich der VBA-Code befindet. Um den Code eines Objekts zu sehen, doppelklicken Sie im Projekt-Explorer auf das gewünschte Objekt. Um beispielsweise den Code des Objekts *Tabelle1* zu sehen, doppelklicken Sie im Projekt-Explorer auf TABELLE1. Das Codefenster ist so lange leer, bis Sie dort ein wenig VBA-Code eingegeben haben.

Im Abschnitt »Mit dem Codefenster arbeiten« weiter hinten in diesem Kapitel erfahren Sie mehr über die Codefenster.

Direktfenster/Direktbereich

Das Direktfenster, das auch Direktbereich genannt wird, ist möglicherweise gerade nicht sichtbar. Falls Sie das Direktfenster nicht sehen, drücken Sie Strg+G oder wählen Sie ANSICHT | DIREKTFENSTER. Um das Direktfenster zu schließen, klicken Sie in der Titelleiste die SCHLIESSEN-Schaltfläche (mit dem X) an oder klicken Sie mit der rechten Maustaste eine beliebige Stelle im Direktfenster an und wählen Sie AUSBLENDEN.

Das Direktfenster ist vor allem dann nützlich, wenn Sie VBA-Anweisungen direkt ausführen oder wenn Sie Ihren Code debuggen (von Fehlern befreien) wollen. Falls Sie gerade erst mit VBA anfangen, ist dieses Fenster noch nicht so wichtig. Sie können es daher schließen und den freiwerdenden Platz für andere VBE-Elemente verwenden.

Mit dem Projekt-Explorer arbeiten

Wenn Sie mit VBE arbeiten, ist jede Excel-Arbeitsmappe ein eigenes Projekt. Sie können sich ein Projekt wie eine Sammlung

von Objekten vorstellen, die ähnlich wie eine Gliederung hierarchisch organisiert ist. Um ein Projekt zu öffnen, klicken Sie das Pluszeichen an, das sich im Projekt-Explorer links neben dem Projektnamen befindet. Um ein Projekt zu reduzieren, klicken Sie das Minuszeichen links neben dem Projektnamen an. Sie können die Elemente (Knoten) im Projekt-Explorer auch doppelt anklicken, um so die Ansicht zu erweitern beziehungsweise zu reduzieren.

Abbildung 2.2 zeigt den Projekt-Explorer mit zwei Projekten: eine Arbeitsmappe mit dem Namen *Mappe1* und eine weitere mit dem Namen *Mappe2*.

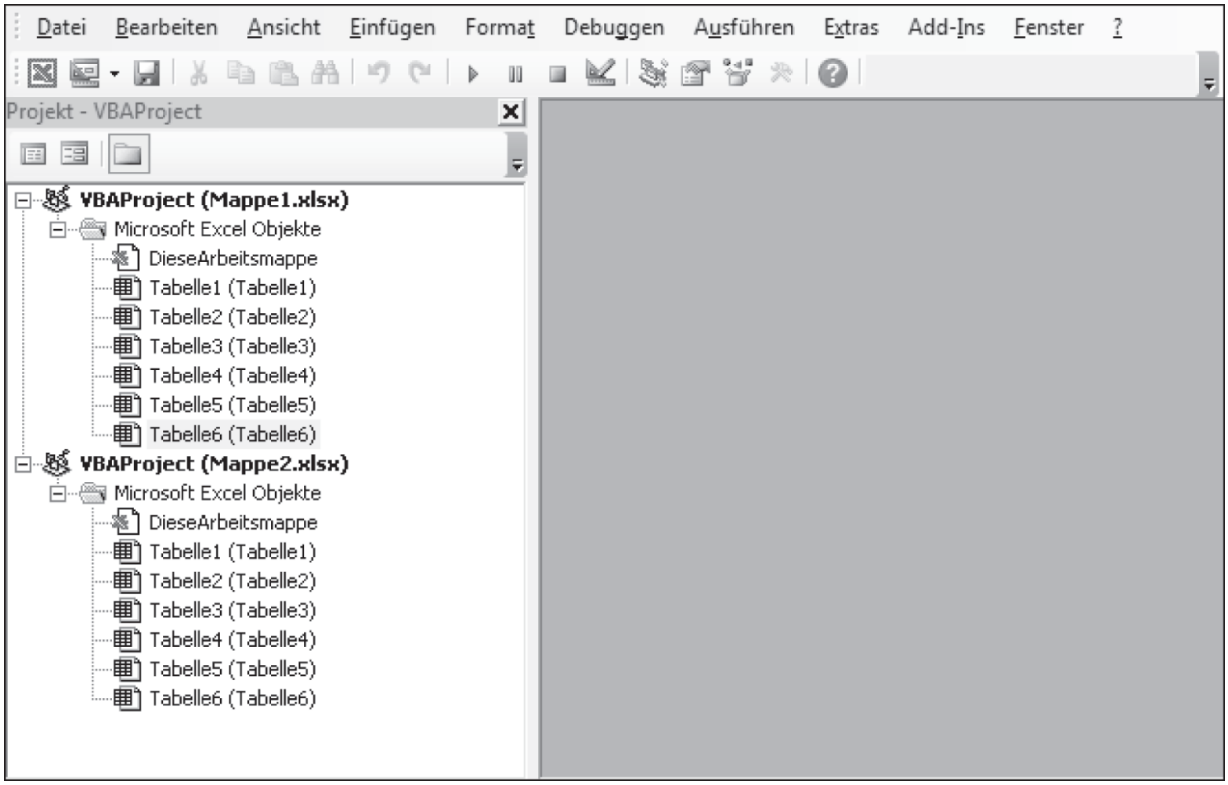


Abbildung 2.2: Der Projekt-Explorer zeigt zwei Projekte an. Beide Projekte sind erweitert und Sie sehen die jeweils darin enthaltenen Objekte.

Jedes Projekt enthält mindestens den Knoten MICROSOFT EXCEL OBJEKTE. Wenn Sie diesen Knoten erweitern, sehen Sie ein Element für jedes Tabellenblatt in der Arbeitsmappe (jedes Tabellenblatt wird als eigenständiges Objekt behandelt) und ein weiteres Objekt mit dem Namen DIESEARBEITSMAPPE (das die

Arbeitsmappe darstellt). Falls das Projekt VBA-Module enthält, wird im Projekt-Explorer zusätzlich der Knoten MODULE angezeigt.

Ein neues VBA-Modul einfügen

Wenn Sie ein Makro aufzeichnen, fügt Excel in das Projekt automatisch ein VBA-Modul ein, in dem das aufgezeichnete Makro abgespeichert wird. In welcher Arbeitsmappe das aufgezeichnete Makro abgelegt wird, hängt von Ihrer Auswahl ab, die Sie kurz vor dem Start der Makroaufzeichnung treffen.

Ein VBA-Modul kann drei verschiedene Arten von VBA-Code enthalten:

- **Deklarationen:** Eine oder mehrere Anweisungen mit informativem Charakter, die Sie VBA bereitstellen. Beispielsweise können Sie den Datentyp der Variablen festlegen, die Sie verwenden wollen, oder verschiedene, modulweite Optionen festlegen.
- **Sub-Prozeduren:** Eine Reihe von Programmierbefehlen, die eine bestimmte Aktion ausführen. Alle aufgezeichneten Makros sind Sub-Prozeduren.
- **Function-Prozeduren:** Eine Reihe von Programmierbefehlen, die genau einen Wert zurückgeben (konzeptionell ähnlich wie die Tabellenblattfunktionen wie beispielsweise *SUMME*).

Sie können in einem VBA-Modul beliebig viele Sub-Prozeduren, Function-Prozeduren und Deklarationen einfügen. Es liegt ganz an Ihnen, wie Sie ein VBA-Modul organisieren. Manche Leute bevorzugen es, den gesamten VBA-Code für eine Anwendung in einem VBA-Modul abzulegen; andere bevorzugen es, hierfür mehrere VBA-Module zu verwenden. Es ist eine Frage des persönlichen Geschmacks, ähnlich wie das Einrichten einer Wohnung.

Führen Sie diese Schritte durch, um in ein Projekt manuell ein neues VBA-Modul einzufügen:

1. **Markieren Sie im Projekt-Explorer den Namen des Projekts.**
2. **Wählen Sie EINFüGEN | MODUL.**

Sie können auch Folgendes tun:

1. **Klicken Sie den Projektnamen mit der rechten Maustaste an.**
2. **Wählen Sie im Kontextmenü EINFüGEN | MODUL.**

Das neue Modul wird im Projekt-Explorer unterhalb des Knotens MODULE angezeigt (siehe Abbildung 2.3). Jedes Modul, das Sie in einer bestimmten Arbeitsmappe erstellen, wird im Ordner MODULE abgelegt.

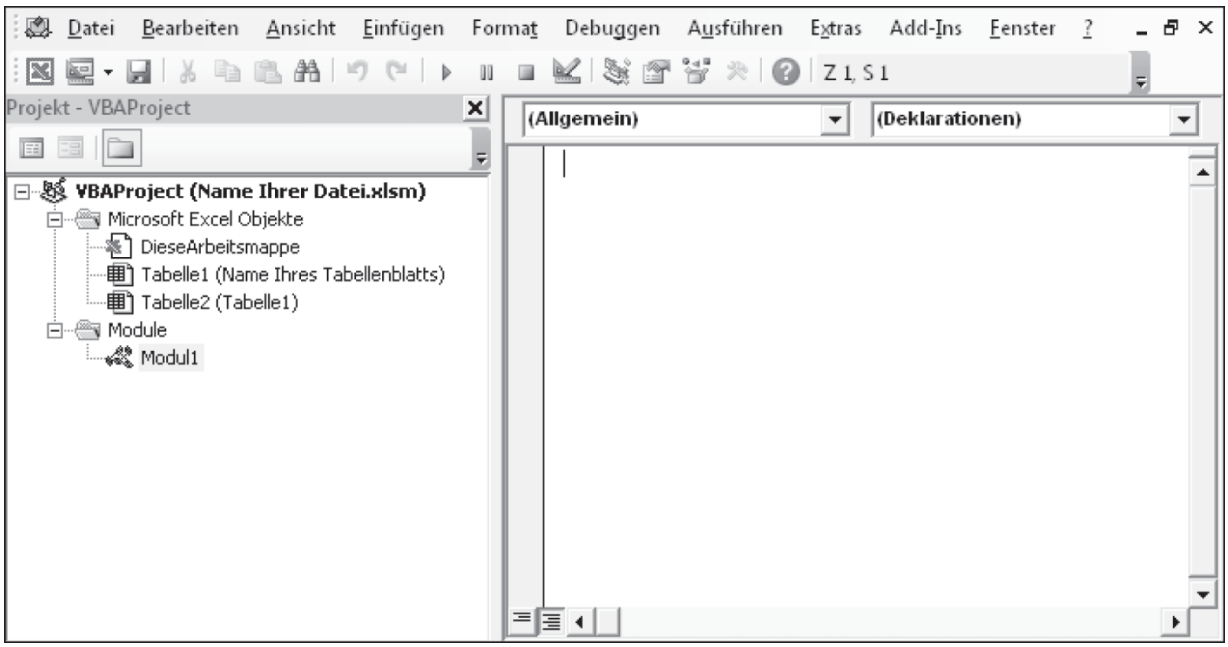


Abbildung 2.3: Code-Module sind im Knoten »Module« des Projekt-Explorers sichtbar.

Ein VBA-Modul entfernen

Ein Code-Modul, das Sie nicht mehr benötigen, können Sie entfernen. Führen Sie dazu folgende Schritte durch:

1. **Markieren Sie im Projekt-Explorer den Namen des Moduls.**

2. **Wählen Sie DATEI | ENTFERNEN VON XXX, wobei anstelle von XXX der Name des Moduls angezeigt wird.**

Oder:

1. **Klicken Sie den Modulnamen mit der rechten Maustaste an.**
2. **Wählen Sie im Kontextmenü ENTFERNEN VON XXX.**

Sie können lediglich VBA-Module entfernen; es ist nicht möglich, die anderen Code-Module zu löschen, wie die der Tabellenblattobjekte oder das der Arbeitsmappe.

Mit dem Codefenster arbeiten

Je kompetenter Sie mit VBA werden, desto mehr Zeit werden Sie im Codefenster verbringen. Aufgezeichnete Makros werden in einem Modul abgespeichert; Sie können VBA-Code aber auch direkt in ein VBA-Modul eingeben.

Fenster minimieren und maximieren

Codefenster verhalten sich so ähnlich wie die Arbeitsmappenfenster in Excel. Sie können sie minimieren, maximieren, ihre Größe ändern, sie ausblenden, anordnen und vieles mehr. Die meisten Leute maximieren das Codefenster, in dem sie gerade arbeiten. Auf diese Weise können sie mehr vom VBA-Code sehen und sind weniger abgelenkt.

Um ein Codefenster zu maximieren, klicken Sie in der Titelleiste auf die Schaltfläche MAXIMIEREN (direkt neben dem X) oder doppelklicken Sie auf die Titelleiste. Um die ursprüngliche Fenstergröße wiederherzustellen, klicken Sie auf die Schaltfläche FENSTER WIEDERHERSTELLEN, die sich nun an der rechten Seite der Menüleiste befindet und zwei kleine Fenstersymbole darstellt.

Es kann Situationen geben, in denen es sinnvoll ist, gleichzeitig zwei oder mehr Codefenster anzeigen zu lassen, beispielsweise wenn Sie den Code in zwei Modulen vergleichen oder Code von einem Modul in ein anderes kopieren wollen. Sie können die Fenster entweder manuell anordnen oder die Befehle FENSTER | NEBENEINANDER und FENSTER | UNTEREINANDER verwenden, damit sie automatisch angeordnet werden.

Sie können schnell von einem Codefenster zum nächsten wechseln, indem Sie Strg+⇥ drücken. Wenn Sie diese Tastenkombination mehrfach eingeben, werden nach und nach alle geöffneten Codefenster nach vorne gebracht. Drücken Sie ⇧ +Strg+⇥, um rückwärts durch die Fenster zu blättern.

Minimieren Sie ein Fenster, falls Sie es nicht benötigen. Sie können auch die X-Schaltfläche SCHLIESSEN oben rechts in der Titelleiste des Codefensters verwenden, wenn Sie das Fenster komplett schließen wollen. (Wenn Sie ein Codefenster schließen, wird es lediglich ausgeblendet; durch das Schließen geht kein Code verloren.) Um das Codefenster erneut zu öffnen, doppelklicken Sie im Projekt-Explorer das gewünschte Element an. Übrigens: das Arbeiten mit diesen Codefenstern hört sich komplizierter an als es in Wirklichkeit ist!

VBA-Code in ein Modul einfügen

Bevor Sie etwas Sinnvolles machen können, benötigen Sie etwas VBA-Code in Ihrem VBA-Modul. Es gibt drei Wege, um VBA-Code in ein VBA-Modul zu bekommen:

- ✔ Verwenden Sie den Excel-Makrorekorder, um Ihre Aktionen aufzuzeichnen und diese in VBA-Code zu konvertieren.
- ✔ Geben Sie den Code direkt ein.
- ✔ Kopieren Sie den Code in einem Modul und fügen Sie ihn in ein anderes ein.

Sie haben die großartige Methode zur Codeerstellung mit dem Excel-Makrorekorder bereits kennengelernt. Jedoch lassen sich nicht alle Aufgaben durch Aufzeichnung eines Makros in VBA-

Code übersetzen. Oft müssen Sie den Code direkt in ein Modul eingeben. Den Code direkt in ein Modul einzugeben, bedeutet entweder, dass Sie ihn direkt eintippen oder dass Sie Code, den Sie irgendwo gefunden haben, kopieren und einfügen.

Die Eingabe und die Bearbeitung von Text in einem VBA-Modul funktionieren so, wie Sie es vermuten. Sie können Text markieren, kopieren, ausschneiden, einfügen und vieles mehr.

Eine einzelne Zeile mit VBA-Code kann so lang sein, wie Sie wollen. Jedoch ist es sinnvoll, lange Codezeilen auf mehrere Zeilen zu umbrechen und hierzu das *Zeilenfortsetzungszeichen* zu verwenden. Um eine einzelne Codezeile, die manchmal auch Anweisung genannt wird, auf der nächsten Zeile fortzusetzen, beenden Sie die erste Zeile mit einem Leerzeichen gefolgt von einem Unterstrich. Hierdurch wird die Anweisung in der nächsten Zeile fortgesetzt. Nachfolgend sehen Sie ein Beispiel für eine Anweisung, die über drei Zeilen verteilt ist:

```
Selection.Sort Key1:=Range("A1"), _
 Order1:=xlAscending, Header:=xlGuess, _
 Orientation:=xlTopToBottom
```

Diese Anweisung wird genauso ausgeführt, als wenn sie sich (ohne Zeilenfortsetzungszeichen) in einer Zeile befinden würde. Beachten Sie, dass die zweite und dritte Zeile dieser Anweisung eingerückt sind. Die Einrückung ist optional, aber sie trägt dazu bei, die Tatsache zu verdeutlichen, dass es sich bei diesen Zeilen nicht um eigenständige Anweisungen handelt.

Der VBE unterstützt mehrere Ebenen für die Funktionen Rückgängig und Wiederholen. Falls Sie versehentlich eine Anweisung gelöscht haben, können Sie die Schaltfläche RüCKGäNGIG in der Symbolleiste verwenden (oder Strg+Z drücken), bis die gelöschte Anweisung wieder erscheint. Nach dem Rückgängigmachen verwenden Sie die Schaltfläche WIEDERHOLEN, damit die rückgängig gemachten Änderungen erneut ausgeführt werden.

Haben Sie Lust, ein wenig echten VBA-Code einzugeben? Probieren Sie folgende Schritte aus:

1. **Erstellen Sie in Excel eine neue Arbeitsmappe.**
2. **Drücken Sie Alt + F11, um den VBE zu aktivieren.**
3. **Klicken Sie im Projekt-Explorer den Namen der neuen Arbeitsmappe an.**
4. **Wählen Sie EINFüGEN | MODUL, um ein VBA-Modul in das Projekt einzufügen.**
5. **Geben Sie in dieses Modul den folgenden Code ein:**

```
Sub GuessName()
 Dim Msg as String
 Dim Ant As Long
 Msg = "Ist Ihr Name " & Application.UserName & "?"
 Antwort = MsgBox(Msg, vbYesNo)
 If Ant = vbNo Then MsgBox "Oh, macht nichts."
 If Ant = vbYes Then MsgBox "Ich muss " & _ &lt;![CDATA[
"hellseherisch sein!"
End Sub
```

6. **Setzen Sie die Einfügemarke an eine beliebige Stelle im Text, den Sie eingegeben haben. Drücken Sie F5, um die Prozedur auszuführen.**

Der VBE verfügt über einen eigenen Satz von Tastenkombinationen, mit denen Sie schnell einen Befehl über Ihre Tastatur auslösen können. F5 ist die Tastenkombination des Befehls AUSFüHREN | SUB/USERFORM AUSFüHREN. Weitere nützliche VBE-Tastenkombinationen finden Sie auf der Schummelseite dieses Buchs.

Während Sie den Code aus Schritt 5 eingeben, werden Sie bemerken, dass der VBE kleine Anpassungen an dem von Ihnen eingegebenen Text vornimmt. Beispielsweise ergänzt der VBE automatisch die Anweisung `End Sub`, sobald Sie die Anweisung `Sub` tippen. Falls Sie vor oder nach dem Gleichheitszeichen kein Leerzeichen eingeben, fügt der VBE dies für Sie ein. Außerdem werden Textfarbe und Groß-/Kleinschreibung bestimmter Begriffe geändert. Dies alles ist ganz normal und kein Grund zur Sorge.

Durch diese Anpassungen hilft der VBE dabei, den Code übersichtlich, aufgeräumt und lesbar zu halten.

Falls Sie die oben stehenden Schritte durchgeführt haben, dann haben Sie soeben eine Sub-Prozedur erstellt, die auch Makro genannt wird. Wenn Sie F5 drücken, führt Excel den Code aus und befolgt die im Code enthaltenen Anweisungen. Anders ausgedrückt: Excel wertet jede Anweisung aus und macht das, was Sie in der Anweisung festgelegt haben. Sie können das Makro beliebig oft ausführen – jedoch verliert es nach ein paar Dutzend Malen möglicherweise ein bisschen von seiner Anziehungskraft.

In diesem einfachen Makro werden folgende Konzepte verwendet:

- ✔ Definition einer Sub-Prozedur (die erste Zeile)
- ✔ Deklaration von Variablen (die `Dim`-Anweisungen)
- ✔ Zuweisung von Werten an Variablen (`Msg` und `Ant`)
- ✔ Verknüpfung von Zeichenketten (mit dem Verkettungsoperator `&`)
- ✔ Verwendung einer eingebauten VBA-Funktion (`MsgBox`)
- ✔ Verwendung von eingebauten VBA-Konstanten (`vbYesNo`, `vbNo` und `vbYes`)
- ✔ Verwendung der If-Then-Entscheidung (zweimal)
- ✔ Beenden einer Sub-Prozedur (die letzte Zeile)

Wie bereits weiter vorne erwähnt, können Sie Code in ein VBA-Modul kopieren und einfügen. So kann beispielsweise eine Sub- oder Function-Prozedur, die Sie für ein Projekt erstellt haben, auch in einem anderen Projekt nützlich sein. Statt Zeit mit dem erneuten Eingeben des Codes zu vergeuden, können Sie das Modul aktivieren und die Standardaktionen für das Kopieren und Einfügen verwenden (Strg+C für Kopieren und Strg+V für Einfügen). Nach dem Einfügen des Codes in das Zielmodul lassen sich dort noch eventuell erforderliche Änderungen

vornehmen.

Die VBA-Entwicklungsumgebung anpassen

Falls Sie ernsthaft vorhaben, ein Excel-Programmierer zu werden, werden Sie eine Menge Zeit mit den VBA-Modulen auf Ihrem Bildschirm verbringen. Um Ihnen das Arbeiten mit der Entwicklungsumgebung so bequem wie möglich zu machen, stellt Ihnen der VBE zahlreiche Optionen zur Verfügung, mit denen Sie ihn anpassen können.

Wählen Sie in der VBA-Entwicklungsumgebung den Befehl EXTRAS | OPTIONEN. Sie sehen dann ein Dialogfeld mit vier Registerkarten: EDITOR, EDITORFORMAT, ALLGEMEIN und VERANKERN. Nehmen Sie sich einen Moment Zeit, um einige der Optionen zu erkunden, die Sie auf den verschiedenen Registerkarten finden.

Die Registerkarte »Editor«

Abbildung 2.4 zeigt die Optionen, die Ihnen auf der Registerkarte EDITOR des Dialogfelds OPTIONEN zur Verfügung stehen. Mit diesen Optionen legen Sie fest, wie bestimmte Dinge im Editor funktionieren.

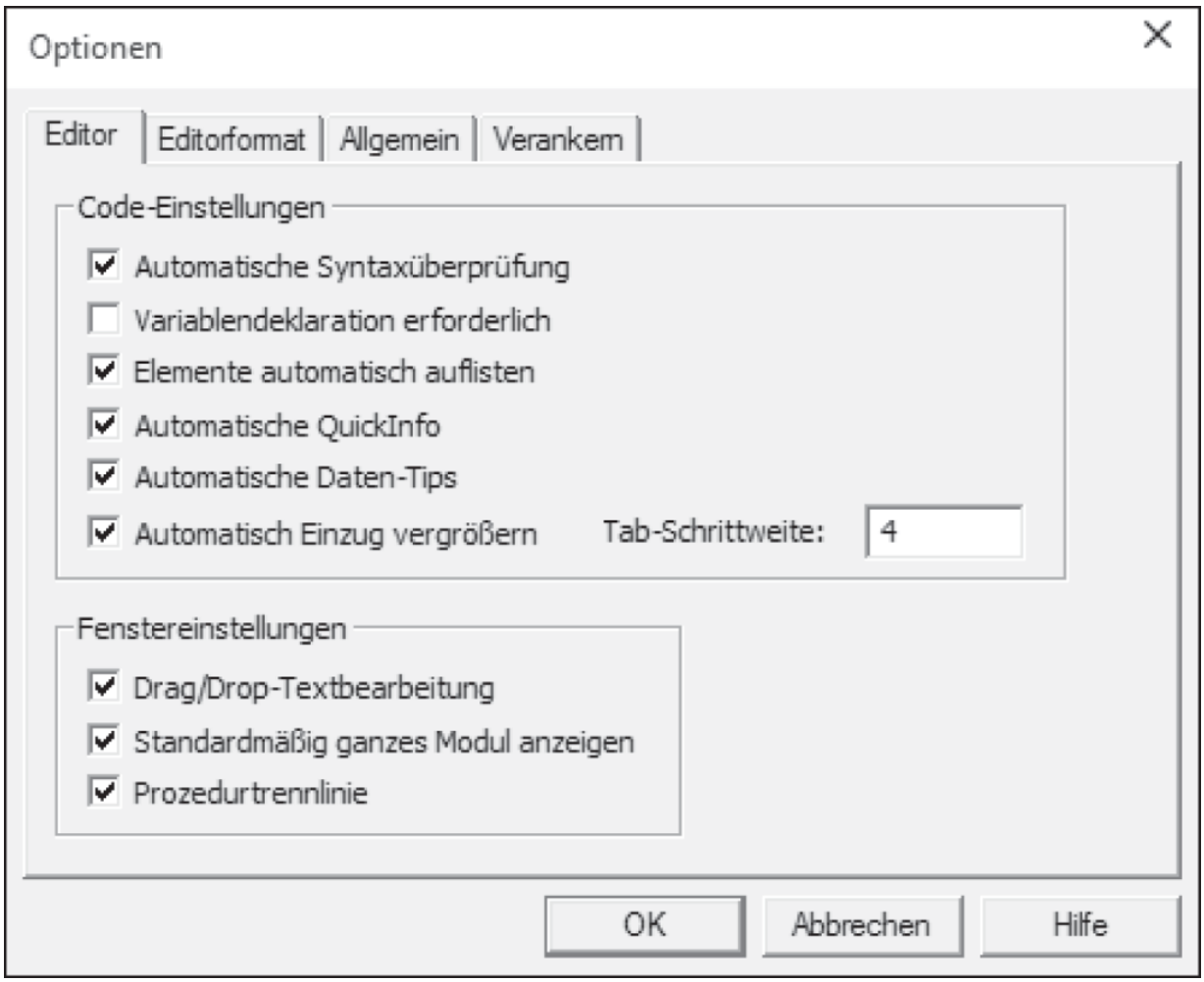


Abbildung 2.4: Die Registerkarte »Editor« des Dialogfelds »Optionen«

Option »Automatische Syntaxüberprüfung«

Mit dieser Option legen Sie fest, ob VBE ein Dialogfeld anzeigt, wenn es im von Ihnen eingegebenen VBA-Code einen Syntaxfehler erkennt. Das Dialogfeld informiert Sie grob darüber, was das Problem ist. Falls Sie diese Option nicht aktivieren, zeigt VBE erkannte Syntaxfehler mittels einer anderen Farbe direkt im Code an. Der Vorteil: Sie müssen sich dann nicht mit den auftauchenden Dialogfeldern herumschlagen.

Option »Variablendeklaration erforderlich«

Falls die Option VARIABLENDEKLARATION ERFORDERLICH eingeschaltet ist, ergänzt der VBE am Anfang jedes Moduls, das Sie einfügen, die folgende Anweisung:

```
Option Explicit
```

Diese Einstellung wirkt sich nur auf neue Module aus, nicht auf bereits vorhandene Module. Wenn diese Anweisung in Ihrem Modul erscheint, müssen Sie jede Variable, die Sie verwenden, explizit deklarieren. Die Verwendung einer `Dim`-Anweisung ist eine Möglichkeit, Variablen zu deklarieren.

Option »Elemente automatisch auflisten«

Schalten Sie die Option ELEMENTE AUTOMATISCH AUFLISTEN ein, damit VBE Sie beim Eingeben von VBA-Code unterstützt. Durch diese Option wird eine Liste mit Befehlen angezeigt, die zu der Anweisung passen, die Sie gerade eingeben. Dies ist eines der besten Features des Visual-Basic-Editors.

Option »Automatische QuickInfo«

Wenn die Option AUTOMATISCHE QUICKINFO eingeschaltet ist, zeigt der VBE, während Sie eine Anweisung eingeben, Informationen zu Funktionen und deren Argumenten an. Diese Funktion kennen Sie wahrscheinlich von Excel: Wenn Sie eine neue Formel mit einer Funktion eingeben, zeigt Excel automatisch die Parameter dieser Funktion an.

Option »Automatische Daten-Tipps«

Wenn Sie die Option AUTOMATISCHE DATEN-TIPPS einschalten, zeigt VBE während des Debuggens den Wert der Variablen an, auf der sich derzeit der Mauszeiger befindet. Diese Option ist standardmäßig eingeschaltet und ist oft sehr nützlich. Es gibt keinen Grund, diese Option zu deaktivieren.

Option »Automatisch Einzug vergrößern«

Die Option für die automatische Vergrößerung des Einzugs legt fest, ob VBE jede neue Codezeile genauso weit einrückt wie die vorherige. Die meisten Entwickler verwenden Einzüge, um den Code übersichtlicher zu machen. Daher ist diese Option üblicherweise eingeschaltet.

Verwenden Sie die ⇥ -Taste und nicht die Leertaste, um Ihren Code einzurücken. Sie können auch ⇧+⇥ eingeben, um den Einzug zu verkleinern. Wenn Sie den Einzug für mehrere Codezeilen ändern wollen, markieren Sie diese und drücken Sie dann die ⇥ -Taste.

Die Symbolleiste BEARBEITEN der VBA-Entwicklungsumgebung (die standardmäßig ausgeblendet ist), enthält zwei nützliche Schaltflächen: EINZUG VERGRöSSERN und EINZUG VERKLEINERN. Mit diesen Schaltflächen lässt sich der Einzug eines Codeblocks schnell ändern. Markieren Sie den Code und klicken Sie eine dieser Schaltflächen an, um den Einzug des Blocks zu ändern.

Option »Drag/Drop-Textbearbeitung«

Wenn die Option DRAG/DROP-TEXTBEARBEITUNG eingeschaltet ist, können Sie Text kopieren und verschieben, indem Sie ihn mit der Maus ziehen und ablegen.

Option »Standardmäßig ganzes Modul anzeigen«

Die Option STANDARDMäSSIG GANZES MODUL ANZEIGEN wirkt sich nur auf neue und nicht auf bereits vorhandene Module aus. Falls diese Option eingeschaltet ist, werden alle Prozeduren im Codefenster als scrollbare Liste angezeigt. Wenn die Option ausgeschaltet ist, können Sie immer nur eine Prozedur sehen.

Option »Prozedurtrennlinie«

Wenn die Option PROZEDURTRENNLINIE eingeschaltet ist, werden im Codefenster nach jeder Prozedur Trennlinien angezeigt. Durch diese Trennlinien kann man besser sehen, wo ein Codefragment endet und wo das nächste beginnt.

Die Registerkarte »Editorformat«

Abbildung 2.5 zeigt die Registerkarte EDITORFORMAT des Dialogfelds OPTIONEN. Auf dieser Registerkarte legen Sie die

Optionen für den VBE fest.

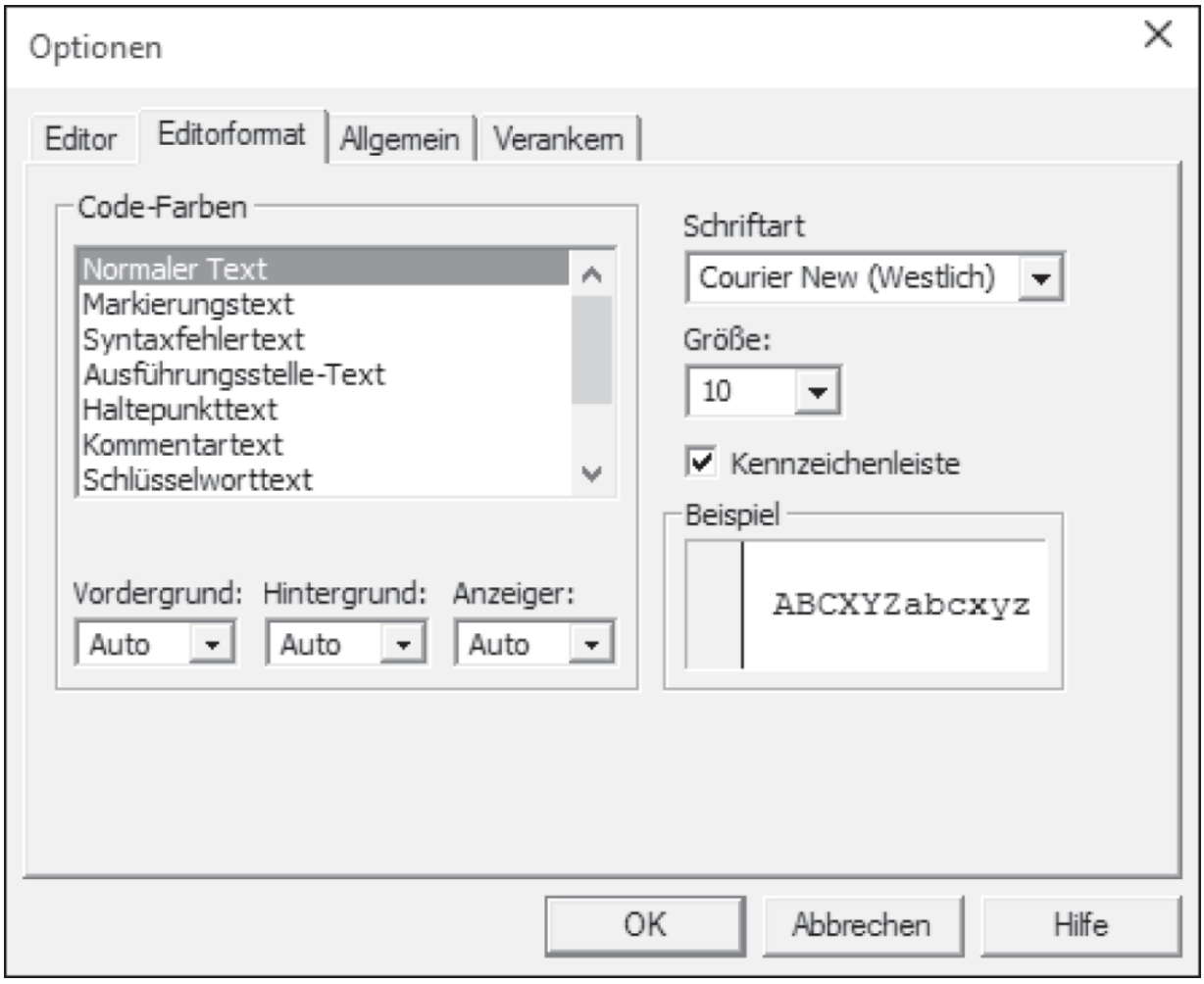


Abbildung 2.5: So legen Sie das Aussehen von VBE auf der Registerkarte »Editorformat« fest.

Option »Code-Farben«

Mit der Option CODE-FARBEN können Sie die Textfarbe und die Hintergrundfarbe für die verschiedenen Elemente des VBA-Codes einstellen. Dies ist vor allem eine Frage des persönlichen Geschmacks. Die meisten Excel-Entwickler verwenden die vorgegebenen Einstellungen. Wenn Sie mögen, können Sie mit diesen Einstellungen herumspielen und schauen, ob Ihnen andere Farbkombinationen besser gefallen.

Option »Schriftart«

Mit der Option SCHRIFTART legen Sie die Schrift für die VBA-Module fest. Um optimale Ergebnisse zu erhalten, verwenden Sie am besten eine nicht-proportionale Schrift, wie beispielsweise COURIER NEW. Bei einer nicht-proportionalen Schrift haben alle Zeichen die gleiche Breite. Ihr Code ist besser lesbar, weil die einzelnen Zeichen dann sauber vertikal ausgerichtet sind. Außerdem lassen sich auf diese Weise mehrere aufeinanderfolgende Leerzeichen gut erkennen (was in bestimmten Fällen nützlich ist).

Option »Größe«

Mit dieser Option legen Sie die Schriftgröße fest, in der Ihr Code in den VBA-Modulen angezeigt wird. Diese Einstellung hängt von Ihren persönlichen Vorlieben ab, die unter anderem von der Bildschirmauflösung und von Ihrer Sehschärfe beeinflusst werden.

Option »Kennzeichenleiste«

Diese Option steuert, ob die vertikale Kennzeichenleiste in Ihren Modulen sichtbar ist oder nicht. Am besten lassen Sie diese Option eingeschaltet; anderenfalls können Sie beim Debuggen Ihres VBA-Codes die hilfreichen grafischen Kennzeichen nicht sehen.

Die Registerkarte »Allgemein«

Abbildung 2.6 zeigt die Optionen, die Sie auf der Registerkarte ALLGEMEIN des Dialogfelds OPTIONEN finden. In den allermeisten Fällen sind die Voreinstellungen perfekt und müssen nicht geändert werden.

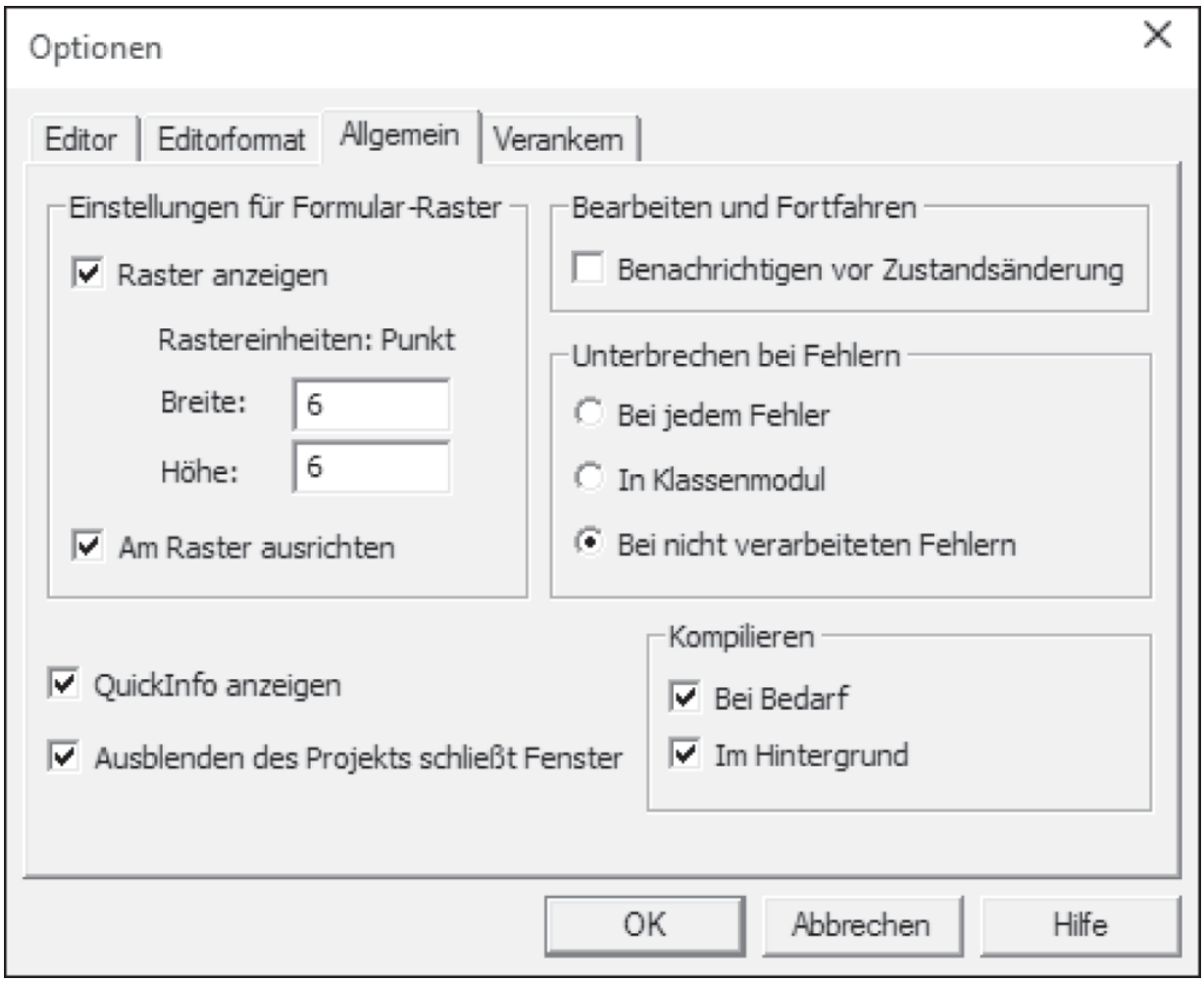


Abbildung 2.6: Die Registerkarte »Allgemein« des Dialogfelds »Optionen«

UNTERBRECHEN BEI FEHLERN ist die wichtigste Option auf der Registerkarte ALLGEMEIN. Wenn Sie gerade am Anfang Ihrer Karriere als Excel-Makroprogrammierer stehen, ist es sinnvoll, in dieser Optionsgruppe die Einstellung BEI NICHT VERARBEITETEN FEHLERN zu verwenden. Diese Einstellung sorgt dafür, dass Excel Ihre Fehler erkennen kann, während Sie den Code eingeben.

Die Registerkarte »Verankern«

Abbildung 2.7 zeigt die Registerkarte VERANKERN. Die dort verfügbaren Optionen steuern, wie sich die verschiedenen VBE-Fenster verhalten. Wenn ein Fenster verankert ist, dann befindet es sich an einem festen Platz an einem der Ränder des VBE-

Programmfensters. Das Verankern von Fenstern macht es einfacher, ein bestimmtes Fenster zu erkennen und zu verorten. Wenn Sie das Verankern für alle Fenster ausschalten, erhalten Sie auf dem Bildschirm ziemlich schnell ein ziemliches Durcheinander. Die Voreinstellungen sind für die meisten Fälle prima.

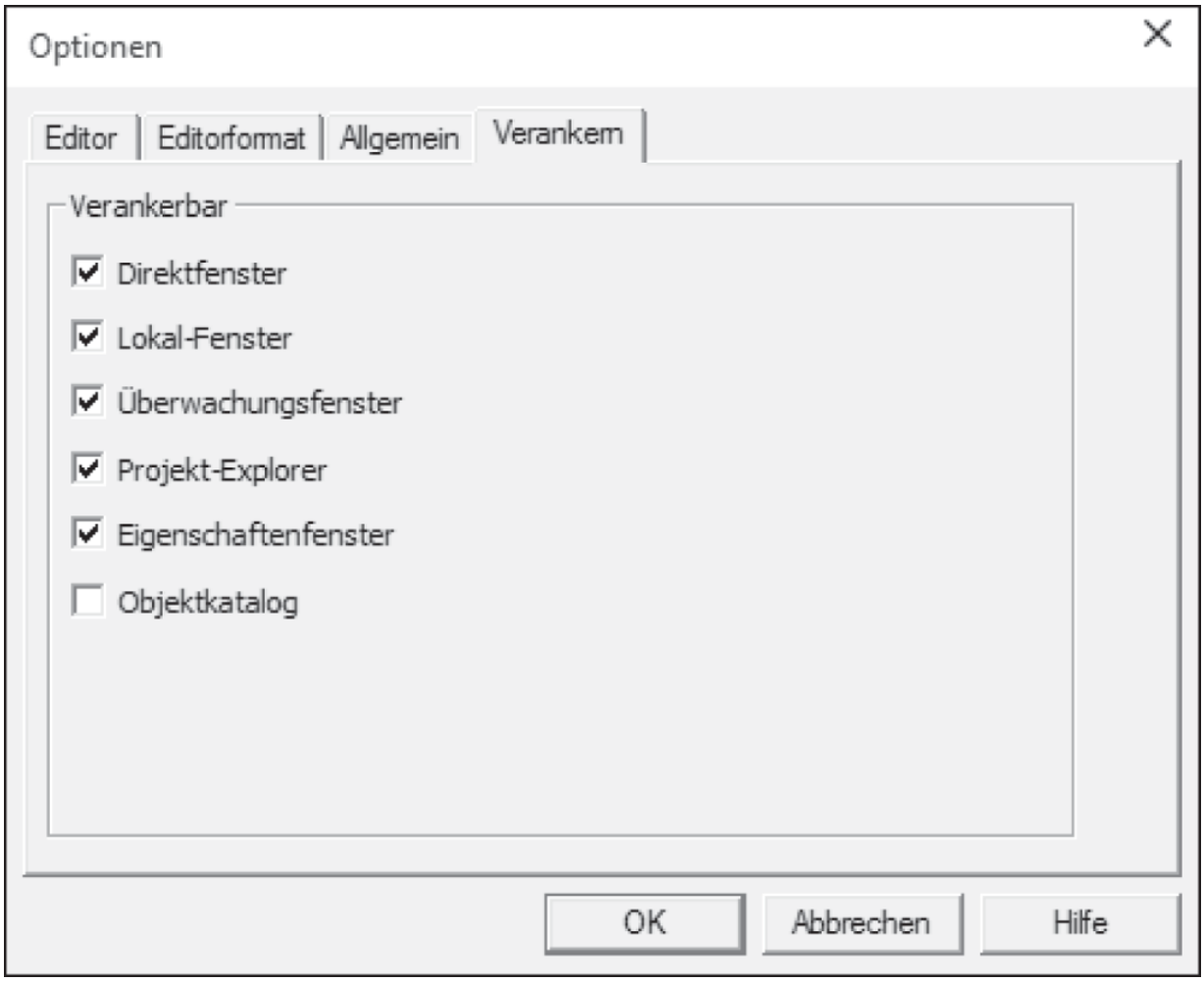


Abbildung 2.7: Die Registerkarte »Verankern« des Dialogfelds »Optionen«

Kapitel 3

Die Anatomie von Makros

IN DIESEM KAPITEL

- Das Excel-Objektmodell verstehen
- Variablen in Ihren Makros einsetzen
- Ereignisprozeduren verwenden
- Potenzielle Makrofehler behandeln

Die Maschine hinter den Makros ist VBA (Visual Basic for Applications). Wenn Sie ein Makro aufzeichnen, erstellt Excel im Hintergrund den VBA-Code, der den durchgeführten Aktionen entspricht. Damit Sie Makros richtig verstehen, ist es wichtig, die zugrundeliegende Sprache VBA zu verstehen.

Mit diesem Kapitel erhalten Sie einen guten Einstieg in die Themen Objekte, Variablen, Ereignisse und Fehlerbehandlung, die Sie in den zahlreichen Makros dieses Buchs im praktischen Einsatz erleben werden.

Ein kurzer Überblick über das Excel-Objektmodell

Visual Basic for Applications ist eine objektorientierte Programmiersprache. Das Grundkonzept bei der *objektorientierten Programmierung* ist, dass eine Softwareanwendung (in unserem Fall Excel) aus verschiedenen, einzelnen Objekten besteht, die jeweils einen eigenen Satz von Merkmalen und Einsatzzwecken bereitstellen. Eine Excel-Anwendung enthält Zellen, Tabellenblätter, Diagramme, Pivot-

Tabellen, Zeichnungsobjekte – die Liste der Excel-Objekte scheint fast endlos zu sein. Jedes Objekt besitzt bestimmte Merkmale, die *Eigenschaften* genannt werden, und einen eigenen Satz von Einsatzzwecken, die *Methoden* genannt werden.

Sie können dieses Konzept auch auf die Objekte übertragen, die Sie im realen Leben nutzen, wie den Computer in Ihrem Büro, Ihr Auto in der Garage oder den Kühlschrank in Ihrer Küche. Jedes dieser Objekte besitzt Merkmale wie Höhe, Gewicht oder Farbe. Außerdem hat jedes Objekt einen bestimmten Einsatzzweck: den Computer nutzen Sie, um mit Excel zu arbeiten, das Auto transportiert Sie über größere Entfernungen und mit dem Kühlschrank halten Sie verderbliche Lebensmittel frisch.

Auch die VBA-Objekte haben ihre speziellen Eigenschaften und Einsatzzwecke. Eine Tabellenblattzelle ist ein Objekt. Zu den beschreibbaren Merkmalen (ihren Eigenschaften) gehören ihre Adresse, die Höhe, eine Füllfarbe und so weiter. Auch eine Arbeitsmappe ist ein VBA-Objekt und zu den Einsatzzwecken (ihren Methoden) zählen öffnen, schließen und ein Diagramm oder eine Pivot-Tabelle einfügen.

In Excel nutzen Sie täglich Arbeitsmappen, Tabellenblätter und Bereiche. Vermutlich stellen Sie sich diese Objekte als Teile von Excel vor und trennen sie nicht in Ihrer Vorstellung. Für Excel sind diese Objekte jedoch unterschiedliche Bestandteile eines hierarchischen Modells, das Excel-Objektmodell genannt wird. Das *Excel-Objektmodell* ist ein Satz von sauber definierten Objekten, die aufgrund der Beziehungen, die diese Objekte miteinander haben, strukturiert sind.

Objekte verstehen

In der realen Welt können Sie alles, was Sie sehen, als Objekt beschreiben. Nehmen Sie Ihr Haus; auch dies ist ein Objekt. Ihr Haus hat verschiedene Zimmer; jedes Zimmer ist wiederum selbst ein Objekt. In den Zimmern können sich Schränke befinden. Auch diese Schränke sind Objekte. Wenn Sie sich Ihr Haus, die Zimmer und die Schränke vorstellen, können Sie hierarchische

Beziehungen zwischen diesen Objekten erkennen. So ähnlich ist dies auch in Excel.

Das *Application*-Objekt ist in Excel das allumfassende Objekt, ähnlich wie Ihr Haus. Innerhalb des *Application*-Objekts gibt es die Arbeitsmappen (englisch: *workbooks*). Eine Arbeitsmappe wiederum besitzt ein oder mehrere Tabellenblätter (englisch: *worksheets*). Auf einem Tabellenblatt gibt es Zellen und Bereiche (englisch: *range*). All diese Objekte »leben« in der großen hierarchischen Excel-Struktur.

Um in VBA auf ein bestimmtes Objekt zu verweisen, können Sie das Objektmodell durchlaufen, das heißt die genaue Position des Objekts in dieser hierarchischen Struktur angeben. Um beispielsweise Zelle A1 auf Tabelle1 auszuwählen, geben Sie folgenden Code ein:

```
Activeworkbook.Sheets("Tabelle1").Range("A1").Select
```

In den meisten Fällen wird die Hierarchie des Objektmodells implizit verstanden und Sie brauchen nicht jede Ebene anzugeben. Statt obiger Anweisung können Sie auch die folgende Codezeile verwenden, um Zelle A1 auszuwählen, da Excel davon ausgeht, dass Sie die aktive Arbeitsmappe und das aktive Tabellenblatt verwenden wollen:

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

Falls sich die Zellmarkierung bereits in Zelle A1 befindet, können Sie auch das Objekt `ActiveCell` verwenden und brauchen dann den Range (den Bereich) nicht anzugeben:

```
ActiveCell.Select
```

Sammlungen

Viele der Excel-Objekte gehören zu sogenannten Sammlungen (englisch: *collections*), bei denen es sich um Gruppen gleicher Objekte handelt. Dies entspricht in etwa Ihrem Haus, das in einem Stadtteil steht. Der Stadtteil ist in diesem Fall eine Sammlung von Häusern. Jeder Stadtteil wiederum befindet sich in einer

Sammlung von Stadtteilen, die zusammen eine Stadt sind. Sammlungen sind in Excel ebenfalls Objekttypen.

In jedem `Workbook`-Objekt (Arbeitsmappe) gibt es eine Sammlung von `Worksheets` (Tabellenblätter). Die Sammlung `Worksheets` ist ein Objekt, auf das Sie von Excel aus zugreifen können. Jedes Tabellenblatt Ihrer Arbeitsmappe befindet sich in der Sammlung `Worksheets`.

Wenn Sie auf ein einzelnes Worksheet in der Sammlung zugreifen wollen, können Sie hierfür dessen Position verwenden. Dabei handelt es sich um einen Index, der mit 1 beginnt, oder Sie verwenden den Namen des Tabellenblatts, den Sie in Anführungszeichen eingeben müssen. Wenn Sie die beiden folgenden Codezeilen in einer Arbeitsmappe ausführen, die genau ein Tabellenblatt mit dem Namen *MeinBlatt* enthält, machen die beiden Zeilen exakt das Gleiche:

```
Worksheets(1).Select
Worksheets("MeinBlatt").Select
```

Falls die aktive Arbeitsmappe zwei Tabellenblätter mit den Namen *MeinBlatt* und *DeinBlatt* enthält (und zwar in dieser Reihenfolge), können Sie mit den beiden folgenden Anweisungen auf das zweite Tabellenblatt verweisen:

```
Worksheets(2).Select
Worksheets("DeinBlatt").Select
```

Falls Sie auf ein Tabellenblatt mit dem Namen *MeinBlatt* zugreifen wollen, das sich in einer derzeit nicht aktiven Arbeitsmappe befindet, müssen Sie sowohl den Namen der Arbeitsmappe als auch den Namen des Tabellenblatts angeben:

```
Workbooks("MyData.xls").Worksheets("MeinBlatt").Select
```

Eigenschaften verstehen

Eigenschaften entsprechen den Charakteristika eines Objekts. Ihr Haus besitzt eine Farbe, eine Wohnfläche, ein Baujahr und so weiter. Einige Eigenschaften, wie die Farbe Ihres Hauses, können

sich ändern. Andere Eigenschaften, wie das Jahr, in dem das Haus gebaut wurde, sind unveränderlich.

Genauso hat ein Objekt in Excel, wie beispielsweise das Objekt `Worksheet`, einen Namen (`Name`), der geändert werden kann, und Eigenschaften wie `Rows.Count` (Anzahl der Zeilen), die Sie nicht ändern können.

Um auf eine Eigenschaft zuzugreifen, geben Sie zuerst das Objekt an und dann die Eigenschaft und verbinden beide mit einem Punkt. Sie können beispielsweise den Namen eines Tabellenblatts ändern, indem Sie dessen Eigenschaft `Name` ändern:

```
Sheets("Tabelle1").Name = "MeinBlatt"
```

Einige Eigenschaften sind schreibgeschützt; das heißt, Sie können ihnen nicht direkt einen Wert zuweisen. Ein Beispiel für eine schreibgeschützte Eigenschaft ist die Eigenschaft `Text` einer Zelle, die die formatierte Ausgabe des Werts einer Zelle enthält. Sie können diese Eigenschaft nicht überschreiben oder ändern.

Methoden

Methoden sind die Aktionen, die Sie an einem Objekt ausführen können. Am besten können Sie sich Methoden als Verben vorstellen. Sie können Ihr Haus anstreichen und daher würde sich diese Aktion ungefähr so in VBA übersetzen lassen:

```
Haus.Anstreichen
```

Ein Beispiel für eine Excel-Methode ist die Methode `Select` (auswählen) des `Range`-Objekts:

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

Ein anderes Beispiel ist die Methode `Copy` (kopieren), ebenfalls des `Range`-Objekts:

```
Range("A1").Copy
```

Einige der Methoden besitzen Parameter, mit denen Sie festlegen können, was genau die Methode machen soll. Sie können

beispielsweise die Methode `Paste` effektiver einsetzen, indem Sie den Parameter `Destination` angeben und so das Einfügeziel festlegen:

```
ActiveSheet.Paste Destination:=Range("B1")
```

Ein kurzer Blick auf Variablen

Ein weiteres Konzept, das Sie in den Makros in diesem Buch sehen werden, ist das der Variablen. Es ist wichtig, sich kurz mit Variablen zu beschäftigen, da sie in den meisten Makros, die Sie im Buch kennenlernen werden, eine wichtige Rolle spielen.

Sie können sich Variablen als Speicherbereiche vorstellen, die Sie in Ihren Prozeduren verwenden können. Es gibt unterschiedliche Typen von Variablen, die jeweils einen bestimmten Datentyp aufnehmen können.

Die gängigen Variablentypen

Hier sind einige der Datentypen für Variablen, die Sie in diesem Buch sehen werden:

- **String:** für Text
- **Integer:** für numerische Werte im Bereich von –32.768 bis 32.767
- **Long:** für numerische Daten im Bereich von –2.147.483.648 bis 2.147.483.647
- **Double:** für Fließkommazahlen
- **Variant:** kann beliebige Datentypen aufnehmen
- **Boolean:** für Wahrheitswerte (True, False = Wahr, Falsch)
- **Object:** kann jedes Objekt des Excel-Objektmodells aufnehmen

Wenn Sie in einem Makro eine Variable erzeugen, dann *deklarieren Sie eine Variable*. Hierzu verwenden Sie die Anweisung `Dim` (für das Englische dimension), gefolgt vom Namen

und abschließend dem Datentyp der Variablen. Ein paar Beispiele:

```
Dim MeinText As String
Dim MeineZahl As Integer
Dim MeinTabellenblatt As Worksheet
```

Nachdem Sie eine Variable deklariert haben, können Sie ihr einen Wert zuweisen. Hier ein paar einfache Beispiele, in denen Variablen erzeugt und dann mit einem Wert versehen werden:

```
Dim MeinText As String
MeinText = Range("A1").Value

Dim MeineZahl As Integer
MeineZahl = Range("B1").Value * 25

Dim MeinTabellenblatt As WorksheetSet
MeinTabellenblatt = Sheets("Tabelle1")
```

Die Werte, die Sie Variablen zuweisen, entstammen häufig den Daten, die sich in den Zellen befinden. Die Werte können jedoch auch Informationen sein, die Sie selbst im Laufe der Ausführung des Makros erzeugen. Dies hängt immer davon ab, welche Aufgabe Sie mit dem Makro erledigen wollen. Dies wird im Verlauf des Buchs alles noch deutlicher, wenn Sie die Makros dieses Buchs näher kennenlernen.

Obwohl es möglich ist, in Excel VBA-Code zu erstellen, in dem keine Variablen verwendet werden, werden Sie viele Beispiele sehen, in denen sie benutzt werden. Hierfür gibt es zwei Gründe:

Erstens weiß Excel nicht, wofür die Daten verwendet werden. Excel sieht keine Zahlen, Symbole oder Buchstaben. Es sieht nur Daten. Indem Sie Variablen mit bestimmten Datentypen deklarieren, informieren Sie Excel darüber, wie es mit bestimmten Daten umgehen soll, und Ihre Makros liefern so die erwarteten Ergebnisse.

Zweitens helfen Variablen dabei, Ihren Code effizienter und verständlicher zu machen. Angenommen, in Zelle A1 befindet sich eine Zahl, die Sie im Makro an mehreren Stellen benötigen.

Sie können dann natürlich den Wert aus Zelle A1 immer wieder abrufen, wenn Sie ihn benötigen:

```
Sub Makro1()

 Range("B1").Value = Range("A1").Value * 5
 Range("C1").Value = Range("A1").Value * 10
 Range("D1").Value = Range("A1").Value * 15

End Sub
```

Dieser Code vergeudet jedoch Rechenzeit, da Excel hier gezwungen wird, den gleichen Wert (in Zelle A1) dreimal zu ermitteln. Falls Sie außerdem die Arbeitsmappe ändern müssen und sich dann der benötigte Wert nicht mehr in Zelle A1, sondern in A2 befindet, müssen Sie in Ihrem Code alle Verweise auf die Zelle A1 auf die Zelle A2 abändern.

Es ist besser, den Wert der Zelle A1 nur einmal abzurufen und ihn in einer Variablen zu speichern. Im folgenden Codefragment wird der Wert der Zelle A1 in einer Variablen des Typs `Integer` gespeichert, die den Namen `meinWert` besitzt:

```
Sub MitVariable()

 Dim meinWert As Integer
 meinWert = Range("A1").Value

 Range("C3").Value = meinWert * 5
 Range("D5").Value = meinWert * 10
 Range("E7").Value = meinWert * 15

End Sub
```

Dieser Ansatz verbessert nicht nur die Effizienz Ihres Codes (weil er dafür sorgt, dass Excel den Wert in der Zelle nur einmal liest), sondern er stellt auch sicher, dass Sie lediglich eine einzige Codezeile bearbeiten müssen, sollte sich der Aufbau Ihrer Arbeitsmappe ändern.

Ereignisprozeduren verstehen

Viele Beispiele in diesem Buch implementiert den Code als Ereignisprozeduren. Damit Sie richtig verstehen, warum diese Beispiele Ereignisprozeduren verwenden, ist es wichtig, sich mit Ereignissen vertraut zu machen.

Ein *Ereignis* ist nichts anderes als eine Aktion, die während einer Excel-Sitzung eintritt. Alles, was sich in Excel ereignet, ereignet sich in einem Objekt und wird durch ein Ereignis ausgelöst. Ein paar Beispiele für Ereignisse sind das Öffnen einer Arbeitsmappe, das Einfügen eines Tabellenblatts, die Änderung eines Zellwerts, das Speichern einer Arbeitsmappe oder ein Doppelklick auf eine Zelle – und damit ist das Ende der Liste noch lange nicht erreicht.

Das Gute ist, dass Sie Excel anweisen können, ein bestimmtes Makro beziehungsweise ein Stück Code dann auszuführen, wenn ein bestimmtes Ereignis passiert. Angenommen, Sie wollen sicherstellen, dass eine Arbeitsmappe beim Schließen automatisch gespeichert wird. Sie können dann in das Arbeitsmappenereignis `BeforeClose` Code eingeben, der die Arbeitsmappe vor dem Schließen speichert.

In Kapitel 2 haben Sie im Abschnitt über das Einfügen eines neuen VBA-Moduls gesehen, wie Sie ein Standard-VBA-Modul erstellen, in dem der von Ihnen erstellte Code gespeichert wird. Das Besondere an Ereignisprozeduren ist, dass sie nicht in Standardmodulen gespeichert werden. Wie Sie in den folgenden Abschnitten sehen, werden Ereignisprozeduren direkt in den integrierten Modulen des Objekts selbst gespeichert.

Tabellenblattereignisse

Tabellenblattereignisse treten auf, wenn irgendetwas mit einem bestimmten Tabellenblatt passiert, beispielsweise wenn das Tabellenblatt ausgewählt wird, wenn eine Zelle des Tabellenblatts bearbeitet wird oder wenn im Tabellenblatt eine Formel berechnet wird. Jedes Excel-Tabellenblatt besitzt automatisch ein eigenes

Modul, in dem Sie die Ereignisprozeduren für dieses Tabellenblatt abspeichern.

Um dieses integrierte Modul zu öffnen, klicken Sie die Registerkarte des Tabellenblatts mit der rechten Maustaste an und wählen CODE ANZEIGEN (siehe Abbildung 3.1).

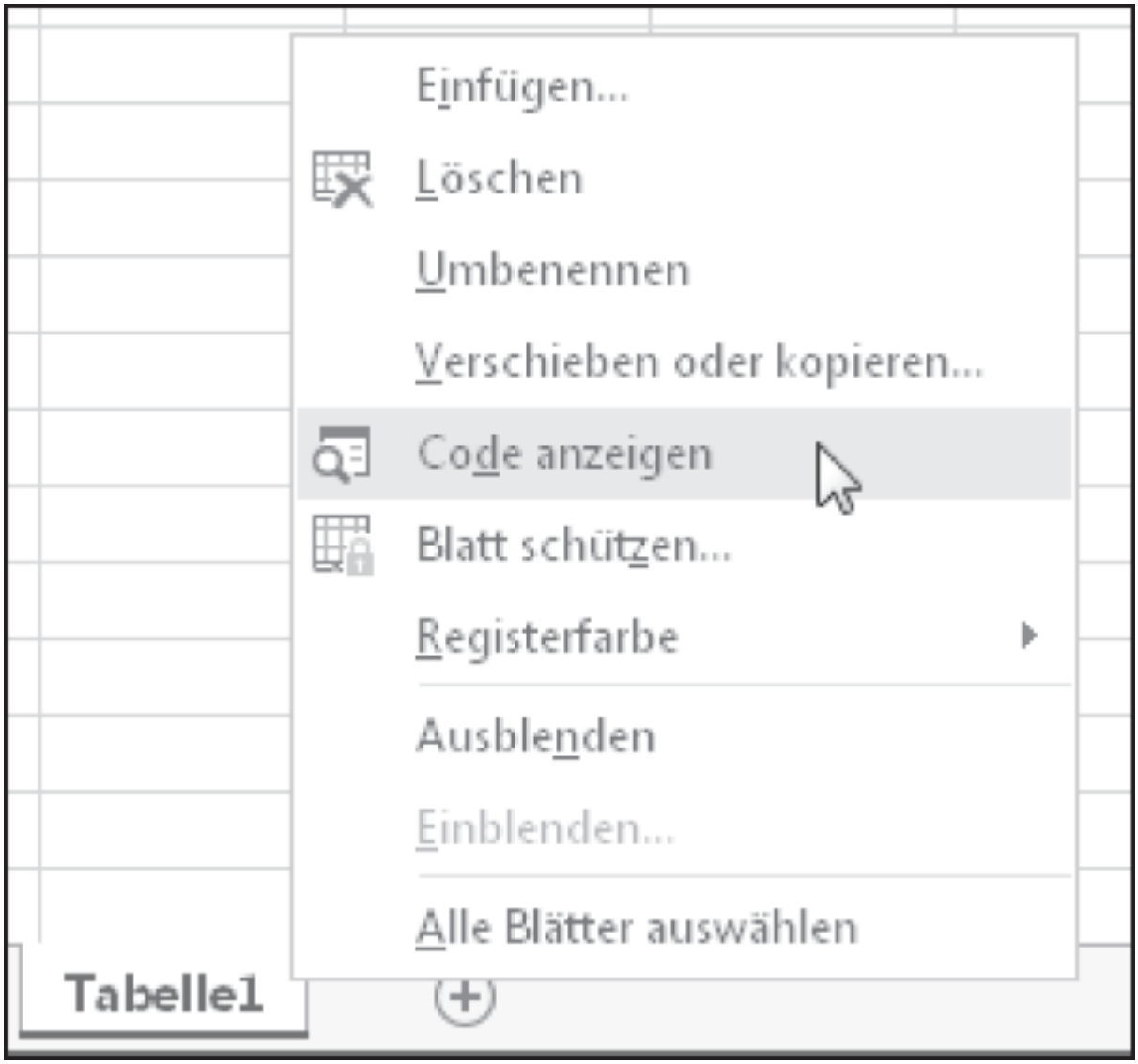


Abbildung 3.1: Das integrierte Code-Modul eines Tabellenblatts öffnen

Der Visual-Basic-Editor öffnet automatisch das integrierte Modul dieses Tabellenblatts. Im oberen Bereich des Codefensters befinden sich zwei Drop-down-Listen.

Wählen Sie in der linken Drop-down-Liste die Option WORKSHEET aus. In der Drop-down-Liste rechts davon wird automatisch das Ereignis `SelectionChange` ausgewählt. Außerdem werden die Anweisungen `Sub` und `End Sub` für die Prozedur eingefügt (siehe Abbildung 3.2), zwischen die Sie Ihren Code eingeben oder einfügen dürfen.

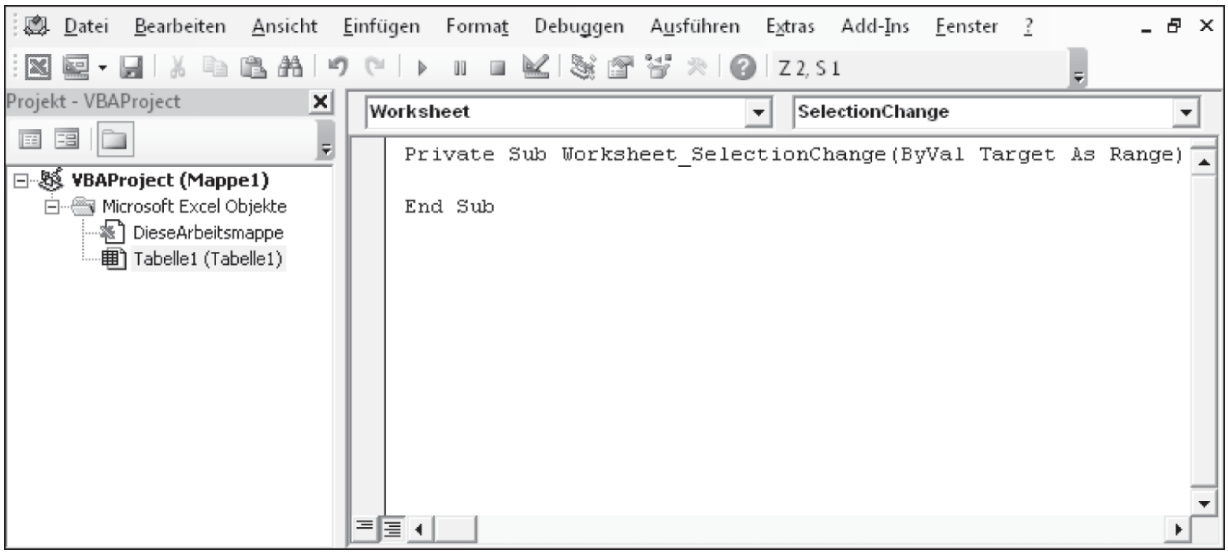


Abbildung 3.2: Das Standardereignis »SelectionChange«

In der Drop-down-Lliste EREIGNISSE können Sie das zu der jeweiligen Aufgabe passende Ereignis auswählen. Abbildung 3.3 zeigt einige der Ereignisse, die Ihnen zur Verfügung stehen.

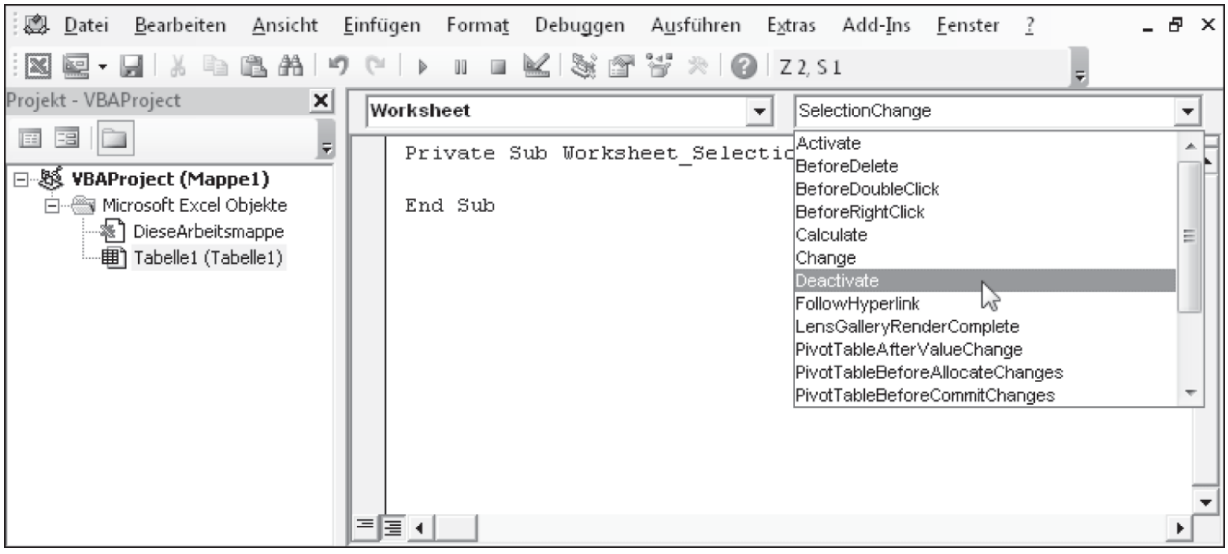


Abbildung 3.3: Wählen Sie das passende Ereignis aus.

Hier sind die für ein Tabellenblatt am häufigsten verwendeten Ereignisse:

- **Worksheet_Change:** Wird ausgelöst, wenn sich im Tabellenblatt Daten ändern.
- **Worksheet_SelectionChange:** Wird ausgelöst, wenn auf dem Tabellenblatt eine andere Zelle beziehungsweise ein anderes Objekt ausgewählt wird.
- **Worksheet_BeforeDoubleClick:** Wird ausgelöst, bevor Excel auf einen Doppelklick auf das Tabellenblatt reagiert.
- **Worksheet_BeforeRightClick:** Wird ausgelöst, bevor Excel auf einen Rechtsklick auf das Tabellenblatt reagiert.
- **Worksheet_Activate:** Wird ausgelöst, wenn der Benutzer von einem anderen auf dieses Tabellenblatt wechselt.
- **Worksheet_Deactivate:** Wird ausgelöst, wenn der Benutzer von diesem auf ein anderes Tabellenblatt wechselt.
- **Worksheet_Calculate:** Wird ausgelöst, wenn Excel nach einer Änderung im Tabellenblatt die Formeln neu berechnet.

Arbeitsmappenereignisse

Arbeitsmappenereignisse treten auf, wenn etwas mit einer bestimmten Arbeitsmappe passiert, etwa wenn eine Arbeitsmappe geöffnet oder geschlossen wird, wenn ein neues Tabellenblatt eingefügt oder wenn die Arbeitsmappe gespeichert wird. Jede Arbeitsmappe besitzt ein eigenes, integriertes Modul, in dem Sie ihre Ereignisprozeduren speichern können.

Um zu diesem integrierten Modul zu gelangen, aktivieren Sie zuerst den Visual-Basic-Editor (drücken Sie Alt+F11). Klicken Sie dann im Projekt-Explorer mit der rechten Maustaste auf DIESEARBEITSMAPPE und wählen Sie den Befehl CODE ANZEIGEN (siehe Abbildung 3.4).

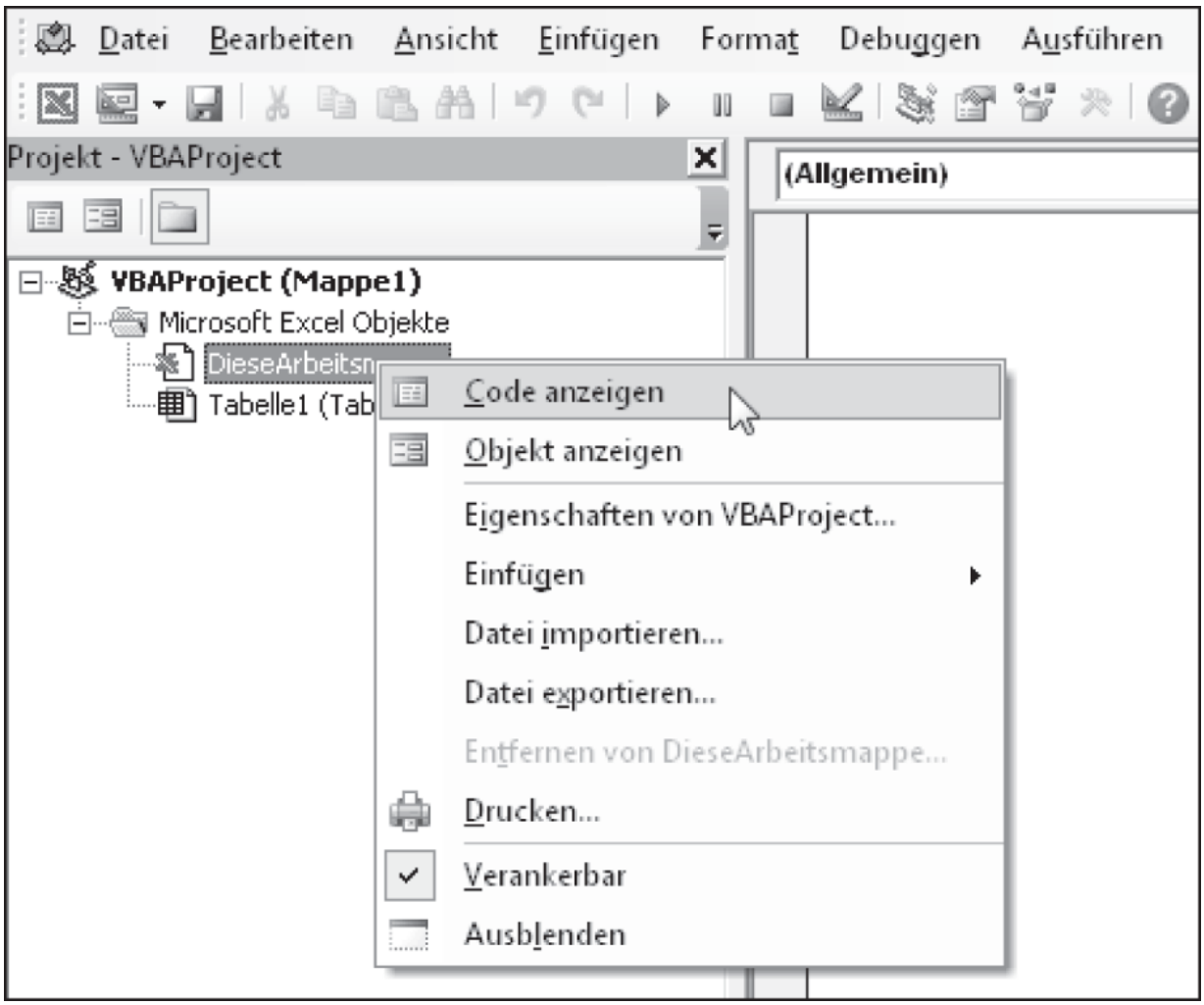


Abbildung 3.4: Das integrierte Modul für »eine Arbeitsmappe öffnen«

Im Codefenster sehen Sie oben zwei Drop-down-Listen. Öffnen Sie die linke Drop-down-Liste und wählen Sie dort WORKBOOK. Hierdurch wird in der Liste mit den Ereignissen (rechts davon) automatisch das Ereignis `OPEN` ausgewählt. Wie Sie in Abbildung 3.5 sehen können, werden wieder gleichzeitig die Anweisungen `Sub` und `End Sub` für die Prozedur eingefügt, zwischen die Sie Ihren Code eingeben oder einfügen dürfen.

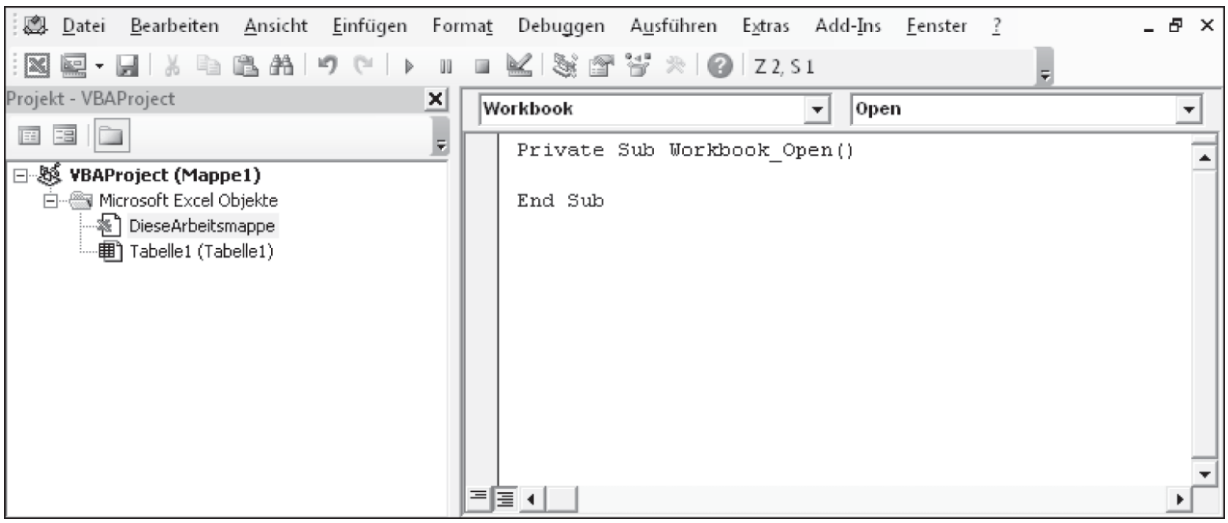


Abbildung 3.5: Das Standardereignis »Open« für Arbeitsmappen

In der Drop-down-Liste EREIGNISSE können Sie das zu der jeweiligen Aufgabe passende Ereignis auswählen. Abbildung 3.6 zeigt einige der Ereignisse, die Ihnen für Arbeitsmappen zur Verfügung stehen.

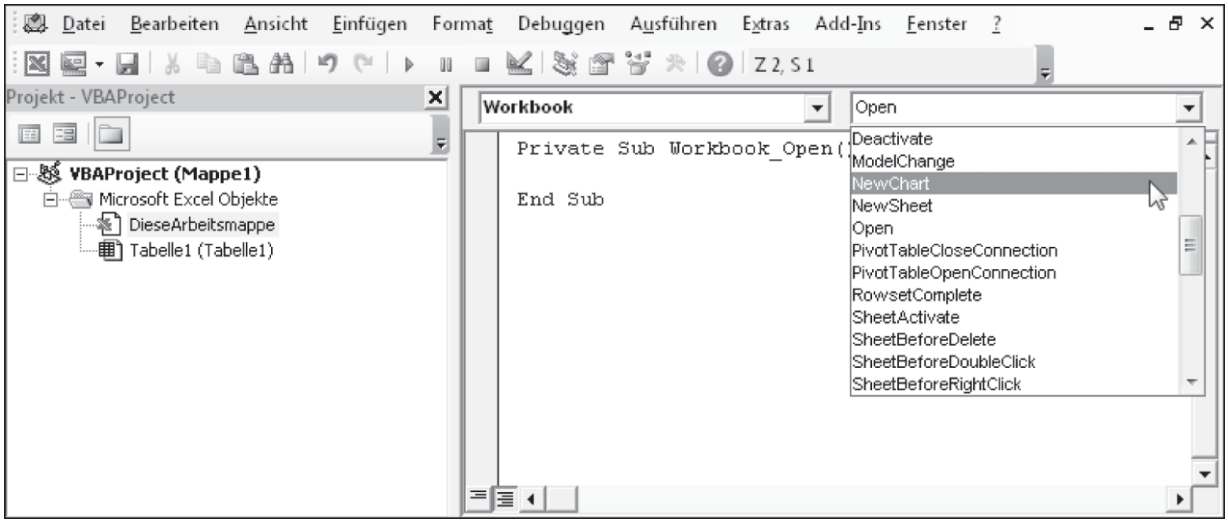


Abbildung 3.6: Öffnen Sie die Drop-down-Liste »Ereignisse« und wählen Sie das passende Ereignis aus.

Hier sind die für eine Arbeitsmappe am häufigsten verwendeten Ereignisse:

- ✔ **Workbook_Open:** Wird ausgelöst, wenn eine Arbeitsmappe geöffnet wird.

- ✔ **Workbook_BeforeSave:** Wird ausgelöst, bevor eine Arbeitsmappe gespeichert wird.
- ✔ **Workbook_BeforeClose:** Wird ausgelöst, bevor Excel die Arbeitsmappe schließt.
- ✔ **Workbook_SheetChange:** Wird ausgelöst, wenn der Anwender zwischen Tabellenblättern wechselt.

Fehlerbehandlung kurz und bündig

In einigen Makros in diesem Buch sehen Sie eine Zeile wie die folgende:

```
On Error GoTo MyError
```

Diese Zeile wird *Fehlerhandler* genannt. Mit Fehlerhandlern geben Sie an, was passieren soll, wenn während der Ausführung Ihres Codes ein Fehler auftritt.

Ohne Fehlerhandler zeigt Excel beim Auftreten eines Fehlers eine nicht sehr hilfreiche Meldung an, an der man kaum erkennen kann, was genau passiert ist. Dank der Fehlerhandler können Sie jedoch genau festlegen, was in so einem Fall passiert: etwa den Fehler einfach ignorieren oder den Code sauber beenden und dem Anwender eine an die Situation bestens angepasste Meldung anzeigen.

Die Anweisung On Error gibt es in drei Varianten:

- ✔ **On Error GoTo IrgendeineSprungmarke:** Der Code springt zu der angegebenen Sprungmarke.
- ✔ **On Error Resume Next:** Der Fehler wird ignoriert und der Code weiter ausgeführt.
- ✔ **On Error GoTo 0:** Deaktiviert den eigenen Fehlerhandler und reaktiviert die Standardfehlerbehandlung von VBA.

On Error GoTo IrgendeineSprungmarke

Bei bestimmten Fehlern in Ihrem Code kann es sinnvoll sein, die Prozedur sauber zu beenden und den Anwender hierüber mit einer verständlichen Meldung zu informieren. In diesen Fällen können Sie die Anweisung `On Error GoTo` verwenden, die Excel anweist, zu einer bestimmten Codezeile zu springen.

In folgendem kleinen Beispiel weisen Sie Excel an, den Inhalt von Zelle A1 durch den Inhalt der Zelle A2 zu dividieren und das Ergebnis in Zelle A3 abzulegen. Einfache Aufgabe. Was kann schiefgehen?

```
Sub Makro1()

 Range("A3").Value = Range("A1").Value / Range("A2").Value

End Sub
```

Zwei Fehler können in diesem Code auftreten. Wenn Zelle A2 den Wert 0 enthält, erhalten Sie einen Division-durch-null-Fehler. Wenn Zelle A2 einen nicht-numerischen Wert enthält, wird ein Fehler wegen nicht passender Datentypen erzeugt.

Um eine für den Anwender unverständliche Fehlermeldung zu vermeiden, können Sie Excel anweisen, dass der Code im Fehlerfall ab der Sprungmarke mit dem Namen `MyExit` weiter ausgeführt werden soll.

Im nachfolgenden Code wird bei der Sprungmarke `MyExit` eine Meldung angezeigt, die die Anwender darüber informiert, was den Fehler ausgelöst hat. Beachten Sie die Anweisung `Exit Sub`, die sich direkt vor der Zeile mit der Sprungmarke befindet. Diese Anweisung stellt sicher, dass die Prozedur beendet wird, wenn kein Fehler auftrat.

```
Sub Makro1 ()

 On Error GoTo MyExit
 Range("A3").Value = Range("A1").Value / Range("A2").Value
 Exit Sub
```

```
MyExit:
 MsgBox "Bitte verwenden Sie nur Zahlen größer als 0"

End Sub
```

On Error Resume Next

Manchmal wollen Sie, dass Excel den Fehler ignoriert und einfach mit der Codeausführung weitermacht. In diesen Situationen verwenden Sie die Anweisung `On Error Resume Next`.

Im folgenden Code soll beispielsweise eine Datei mit dem Namen *GhostFile.exe* aus dem Ordner *C:\Temp* gelöscht werden. Nach dem erfolgreichen Löschen informiert ein Meldungsfeld den Anwender darüber, dass die Datei gelöscht wurde:

```
Sub Makro1 ()

 Kill "C:\Temp\GhostFile.exe"
 MsgBox "Die Datei wurde gelöscht."

End Sub
```

Dies funktioniert perfekt, wenn die zu löschende Datei existiert. Falls die Datei *GhostFile.exe* jedoch aus irgendeinem Grund nicht mehr im Ordner *C:\Temp* vorhanden ist, wird von Excel standardmäßig ein Fehler ausgelöst.

In unserem Beispiel macht es Ihnen aber nichts aus, wenn die Datei nicht existiert, da sie ja doch gelöscht werden soll. Sie können daher den Fehler einfach ignorieren und mit der Codeausführung weitermachen.

Indem Sie die Anweisung `On Error Resume Next` verwenden, nimmt der Code seinen Lauf unabhängig davon, ob die Datei existiert oder nicht.

```
Sub Makro1 ()

 On Error Resume Next
 Kill "C:\Temp\GhostFile.exe"
 MsgBox "Die Datei wurde gelöscht."

End Sub
```

On Error GoTo 0

Nachdem Sie bestimmte Error-Anweisungen verwendet haben, kann es erforderlich sein, das Fehlerbehandlungsverhalten von Excel zurückzusetzen. Das folgende Beispiel verdeutlicht, was damit gemeint ist.

Auch hier wollen Sie eine Datei mit dem Namen *GhostFile.exe* aus dem Ordner *C:\Temp* löschen. Um Fehler zu vermeiden, die aus der Tatsache herrühren, dass die Datei nicht vorhanden ist, verwenden Sie die Anweisung `On Error Resume Next`. Danach führen Sie eine verdächtige mathematische Berechnung durch, in der 100 durch Mike dividiert werden soll:

```
Sub Makro1 ()

 On Error Resume Next
 Kill "C:\Temp\GhostFile.exe"
 Range("A3").Value = 100 / "Mike"

End Sub
```

Die Ausführung dieses Codes sollte wegen der falschen Berechnung eigentlich einen Fehler auslösen, jedoch passiert dies nicht. Warum? Der letzte Fehlerhandler, den Sie im Code verwendet haben, war `On Error Resume Next`. Hierdurch werden *alle* Fehler, die nach dieser Anweisung ausgelöst werden, von Excel ignoriert.

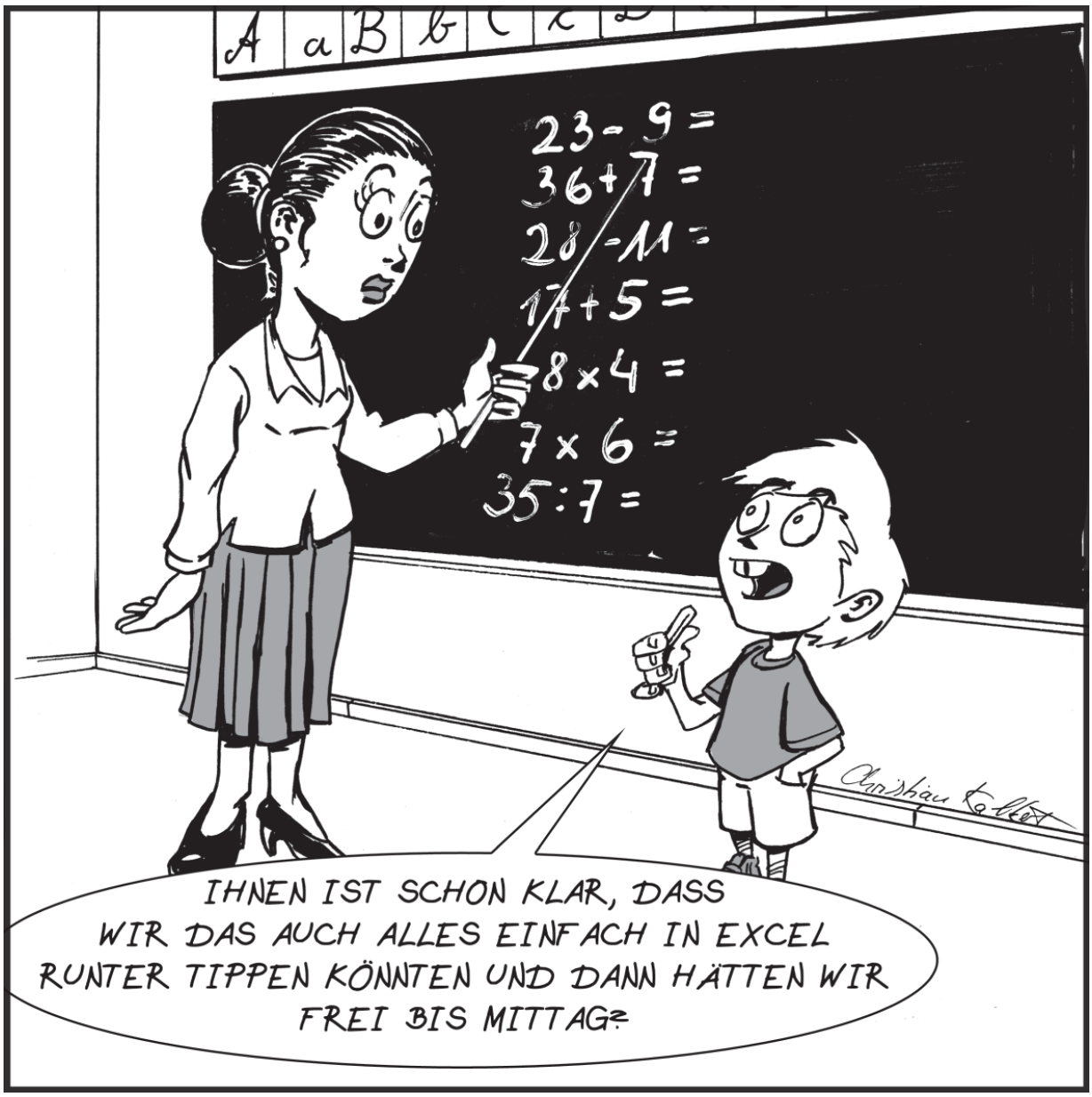
Um dieses Problem zu lösen, verwenden Sie die Anweisung `On Error GoTo 0`. Hierdurch wird die Fehlerbehandlung von Excel wieder auf das Standardverhalten zurückgesetzt:

```
Sub Makro1 ()

 On Error Resume Next
 Kill "C:\Temp\GhostFile.exe"
 On Error GoTo 0
 Range("A3").Value = 100 / "Mike"

End Sub
```

Dieser Code ignoriert nur diejenigen Fehler, die zwischen den Anweisungen `On Error Resume Next` und `On Error GoTo 0` auftreten. Nach dieser Zeile findet wieder die normale Fehlerbehandlung statt, und es wird die durch die falsche Berechnung verursachte und erwartete Fehlermeldung von Excel angezeigt.

Teil II

Aufgaben in Arbeitsmappen mit wenig Arbeit erledigen



IN DIESEM TEIL …

- ✔ Schauen Sie sich verschiedene Techniken zum Bearbeiten Ihrer Arbeitsmappen an
- ✔ Lernen Sie, wie Sie mit Excel-Makros das Erstellen und Duplizieren von Excel-Dateien erledigen
- ✔ Entdecken Sie Makros, mit denen Sie wiederkehrende Tabellenblattaufgaben automatisieren können
- ✔ Setzen Sie Makros ein, um Ihre Excel-Arbeitsmappen zu schützen und zu sichern

Kapitel 4

Makros für Arbeitsmappen

IN DIESEM KAPITEL

- Eine neue Arbeitsmappe neu erstellen
- Eine Arbeitsmappe speichern, wenn eine bestimmte Zelle geändert wird
- Eine Arbeitsmappe vor dem Schließen speichern
- Eine Arbeitsmappe schützen und den Schutz aufheben
- Beim Öffnen eine Arbeitsmappe ein bestimmtes Tabellenblatt aktvieren
- Eine vom Benutzer ausgewählte Arbeitsmappe öffnen
- Erkennen, ob eine Arbeitsmappe bereits geöffnet ist oder überhaupt existiert

Eine Arbeitsmappe ist nicht nur eine Excel-Datei; sie ist auch ein Objekt im Objektmodell von Excel (einer Objekthierarchie, mit der Sie von VBA-Code aus mit Excel interagieren können).

Dies bewirkt, dass Sie mit VBA-Code auf Arbeitsmappen zugreifen können, etwa um automatisch neue Arbeitsmappen zu erstellen, Benutzer daran zu hindern, Arbeitsmappen zu schließen, oder automatische Sicherungskopien von Arbeitsmappen zu erstellen. In diesem Kapitel lernen Sie einige der nützlichsten Makros für Arbeitsmappen kennen.

Eine neue Arbeitsmappe aus dem Nichts erstellen

Es kann Situationen geben, in denen Sie automatisiert eine neue Arbeitsmappe erstellen wollen oder müssen, beispielsweise wenn Sie Daten von einem Tabellenblatt kopieren und diese in eine neue Arbeitsmappe einfügen wollen. Das folgende Makro kopiert einen Zellbereich aus dem aktiven Tabellenblatt und fügt die Daten in eine neue Arbeitsmappe ein.

Wie es funktioniert

Beim Lesen der folgenden Codezeilen werden Sie feststellen, dass das Makro relativ intuitiv ist:

```
Sub Makro1()

'Schritt 1: Kopiere die Daten
 Sheets("Beispiel 1").Range("B4:C15").Copy

'Schritt 2: Erstelle eine neue Arbeitsmappe
 Workbooks.Add


'Schritt 3: Füge die Daten ein
 ActiveSheet.Paste

'Schritt 4: Warnmeldungen der Anwendung ausschalten
 Application.DisplayAlerts = False

'Schritt 5: Die neu erstellte Arbeitsmappe speichern
 ActiveWorkbook.SaveAs _
 Filename:="C:\MeineNeueArbeitsmappe.xlsx"

'Schritt 6: Warnmeldungen der Anwendung wieder einschalten
 Application.DisplayAlerts = True

End Sub
```

1. In Schritt 1 werden ganz einfach die Daten des Zellbereichs B4:C15 aus dem Tabellenblatt »Beispiel 1« kopiert.

   Beachten Sie, dass Sie sowohl den Namen des Tabellenblatts als auch den Zellbereich angeben müssen. Dieser Ansatz ist dann gut geeignet, wenn mehrere Arbeitsmappen gleichzeitig geöffnet sind.

2. Sie verwenden die Methode `Add` des `Workbook`-Objekts, um eine neue Arbeitsmappe zu erstellen. Dies entspricht der Verwendung des Befehls DATEI | NEU | LEERE ARBEITSMAPPE im Menüband von Excel.
3. In Schritt 3 verwenden Sie die Methode `Paste`, um die kopierten Daten in Zelle A1 der neuen Arbeitsmappe einzufügen.

   Bitte beachten Sie, dass im Code das `ActiveSheet`-Objekt verwendet wird. Wenn Sie eine neue Arbeitsmappe erstellen, dann wird diese automatisch aktiviert. (Dies ist das gleiche Verhalten, das Sie sehen, wenn Sie eine Arbeitsmappe manuell erstellen.)
4. In Schritt 4 des VBA-Codes setzen Sie die Eigenschaft `DisplayAlerts` auf `False` und schalten damit alle Excel-Warnungen ab. Sie machen das, weil die neue Arbeitsmappe im nächsten Schritt gespeichert werden soll. Vielleicht lassen Sie das Makro mehrfach laufen, was dann dazu führt, dass Excel mehrfach versucht, die Datei zu speichern.

   Was passiert, wenn Sie versuchen, eine Arbeitsmappe mehrmals zu speichern? Richtig, Excel zeigt einen Warnhinweis an, der Sie darüber informiert, dass eine Datei mit dem Namen bereits existiert, und Sie fragt, ob Sie die vorhandene Datei überschreiben wollen. Da unser Ziel darin besteht, das Erstellen einer neuen Arbeitsmappe zu automatisieren, schalten Sie diese Warnung am besten ab.
5. In Schritt 5 verwenden Sie die Methode `SaveAs`, um die Arbeitsmappe zu speichern. Beachten Sie, dass Sie hier den kompletten Pfad zum Speicherort angeben müssen, einschließlich des Dateinamens. Außerdem muss der angegebene Ordner vorhanden sein, damit die Datei erfolgreich gespeichert werden kann.
6. Da Sie in Schritt 4 alle Warnmeldungen abgeschaltet haben, müssen Sie diese am Ende wieder einschalten (siehe Schritt 6). Falls Sie das nicht machen, zeigt Excel in der laufenden Excel-Sitzung keine Warnungen mehr an.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

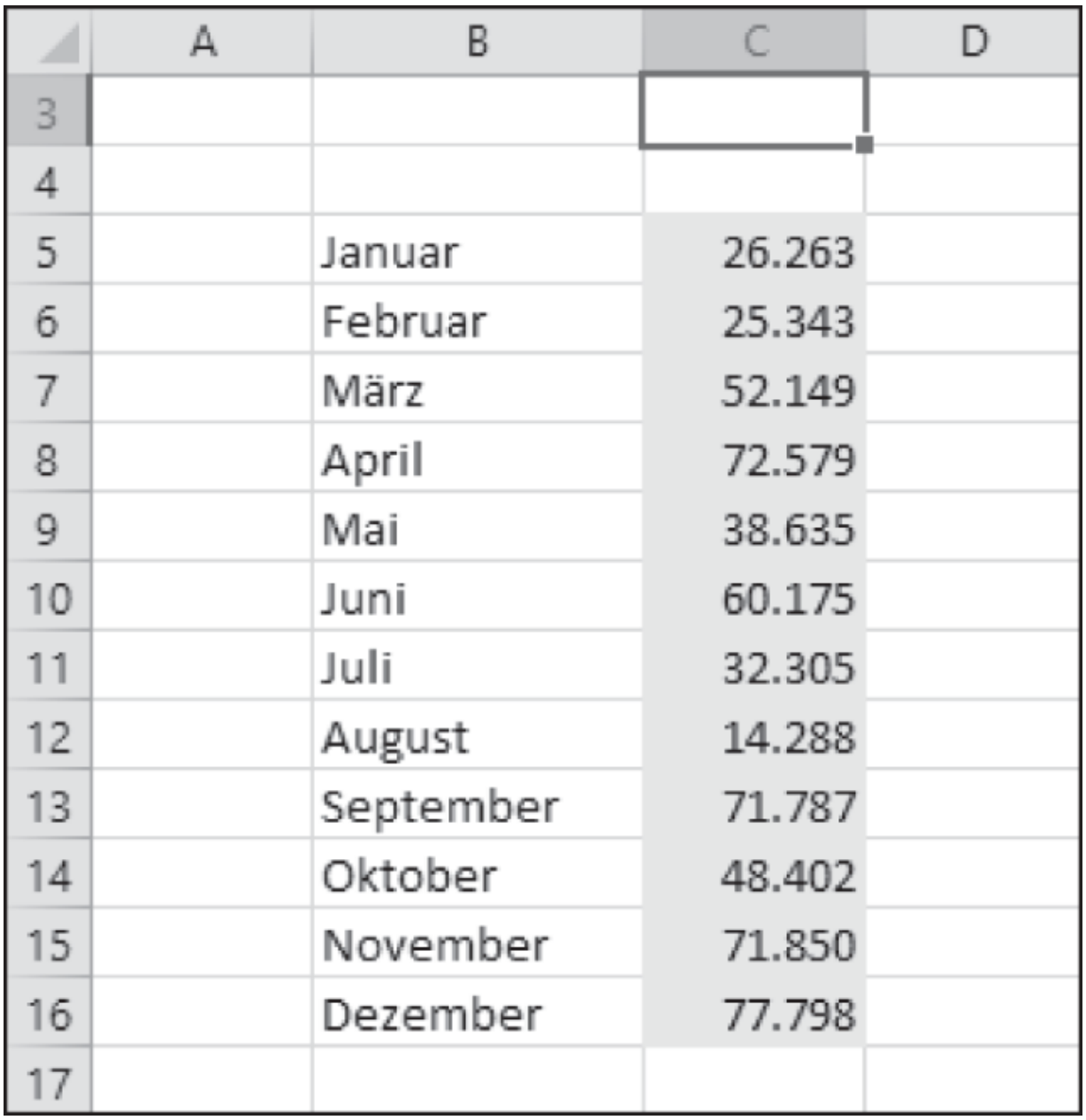
1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

   Möglicherweise müssen Sie für Ihre Arbeitsmappe den Namen des Tabellenblatts, den Zielbereich und den Speicherort anpassen.

Eine Arbeitsmappe speichern, wenn eine bestimmte Zelle geändert wird

Es kann sein, dass Sie an so wichtigen Daten arbeiten, dass Sie die Arbeitsmappe nach jeder Änderung einer Zelle oder eines Zellbereichs automatisch speichern wollen. In diesem Makro können Sie einen Zellbereich definieren. Wird in diesem Zellbereich ein Wert geändert, kümmert sich das Makro automatisch darum, dass die Arbeitsmappe gespeichert wird.

Im Beispiel in Abbildung 4.1 soll die Arbeitsmappe immer dann gespeichert werden, wenn eine beliebige Zelle des Bereichs C5:C16 geändert wurde.



	A	B	C	D
3				
4				
5		Januar	26.263	
6		Februar	25.343	
7		März	52.149	
8		April	72.579	
9		Mai	38.635	
10		Juni	60.175	
11		Juli	32.305	
12		August	14.288	
13		September	71.787	
14		Oktober	48.402	
15		November	71.850	
16		Dezember	77.798	
17				

Abbildung 4.1: Ändert sich eine beliebige Zelle im Bereich C5:C16, wird die Arbeitsmappe sofort gespeichert.

So funktioniert es

Das Geheimnis dieses Makros liegt in der Methode `Intersect`. Da Sie nicht wollen, dass die Arbeitsmappe bei der Änderung jeder beliebigen Zelle gespeichert wird, verwenden Sie die Methode INTERSECT, um zu prüfen, ob die geänderte Zelle und der Bereich, den Sie definiert haben, eine Schnittmenge bilden.

Die Methode `Intersect` gibt einen von zwei möglichen Werten zurück: entweder ein `Range`-Objekt, das die Schnittmenge von zwei Bereichen enthält, oder den Wert `Nothing`. Sie müssen also die geänderte Zelle und den vorher definierten Bereich an die Methode `Intersect` übergeben und dann prüfen, ob Sie einen neuen Bereich oder den Wert `Nothing` erhalten. An diesem Punkt können Sie entscheiden, ob die Arbeitsmappe gespeichert werden soll oder nicht.

```
Private Sub Worksheet_Change(ByVal Target As Range)

'Schritt 1: Prüfen, ob sich die geänderte Zelle und der
' angegebene Bereich überschneiden
 If Intersect(Target, Range("C5:C16")) Is Nothing Then

'Schritt 2: Falls es keine Schnittmenge gibt, wird das
' Makro beendet
 Exit Sub

'Schritt 3: Falls es eine Schnittmenge gibt,
' die Arbeitsmappe speichern
 Else
 ActiveWorkbook.Save

'Schritt 4: Die If-Anweisung abschließen
 End If

End Sub
```

1. In Schritt 1 überprüfen Sie, ob sich die Zielzelle (die Zelle, die geändert wurde) in dem Bereich befindet, den Sie als weiteren Parameter an die Methode `Intersect` übergeben. Wenn Sie den Wert `Nothing` erhalten, bedeutet dies, dass sich die Zielzelle außerhalb des angegebenen Bereichs befindet.
2. In Schritt 2 wird darum folgerichtig das Makro beendet und die Prozedur verlassen, da es keine Schnittmenge zwischen der Zielzelle und dem angegebenen Bereich gibt.
3. Wenn es eine Schnittmenge gibt, wird in Schritt 3 die Methode `Save` der aktiven Arbeitsmappe aufgerufen, was die bisher gespeicherte Version der Mappe durch die aktuelle ersetzt.

4. In Schritt 4 wird lediglich die `If`-Anweisung beendet. Immer dann, wenn Sie eine `If-Then-Else`-Überprüfung durchführen, müssen Sie diesen Block mit der Anweisung `End If` abschließen.

So verwenden Sie es

Um dieses Makro zu implementieren, müssen Sie im Codefenster eine Ereignisprozedur für das Ereignis `Worksheet_Change` einfügen. Da Sie das Makro diesem Ereignis zuordnen, wird es automatisch bei jeder Änderung des Tabellenblatts ausgeführt.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, damit Sie alle Tabellenblätter sehen.**
3. **Klicken Sie das Tabellenblatt an, das Sie mit dem Code verknüpfen wollen.**
4. **Wählen Sie in der Drop-down-Liste EREIGNISSE (siehe Abbildung 4.2) das Ereignis CHANGE aus.**
5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin und passen Sie gegebenenfalls den Zellbereich an.**

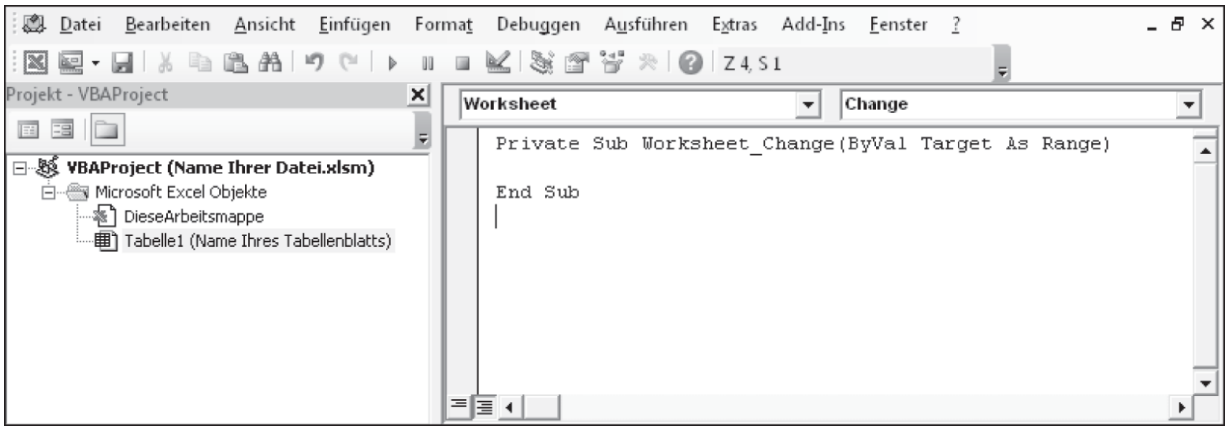


Abbildung 4.2: Tippen Sie den Code in das Tabellenblattereignis `Change` ein oder kopieren Sie ihn dorthin.

Eine Arbeitsmappe vor dem Schließen speichern

Dieses Makro ist eine hervorragendes Möglichkeit, um Anwender davor zu bewahren, eine Arbeitsmappe unbeabsichtigt zu schließen, ohne sie vorher zu speichern. Nachdem dieses Makro implementiert ist, speichert Excel die Arbeitsmappe vor dem Schließen automatisch.

Normalerweise zeigt Excel einen Warnhinweis an, wenn Sie versuchen, eine Arbeitsmappe zu schließen, die noch nicht gespeichert wurde. Viele Anwender lesen diese Meldung aber nicht allzu aufmerksam und klicken so versehentlich auf NEIN, sodass Excel die Arbeitsmappe schließt, ohne sie zu sichern. Mit diesem Makro können Sie die Anwender hiervor bewahren, indem Sie die Mappe speichern lassen und ein ähnliches Meldungsfeld anzeigen, das jedoch keine NEIN-Schaltfläche enthält.

So funktioniert es

Dieser Code wird vom Arbeitsmappenereignis `BeforeClose` ausgelöst. Wenn Sie eine Arbeitsmappe schließen, wird dieses Ereignis ausgelöst und der Code dieser Ereignisprozedur ausgeführt. Die Kernaufgabe des Codes ist recht einfach: Er fragt den Anwender, ob die Arbeitsmappe gespeichert und geschlossen werden soll (siehe Abbildung 4.3). Anschließend wertet das Makro aus, ob der Anwender auf OK oder auf ABBRECHEN geklickt hat.

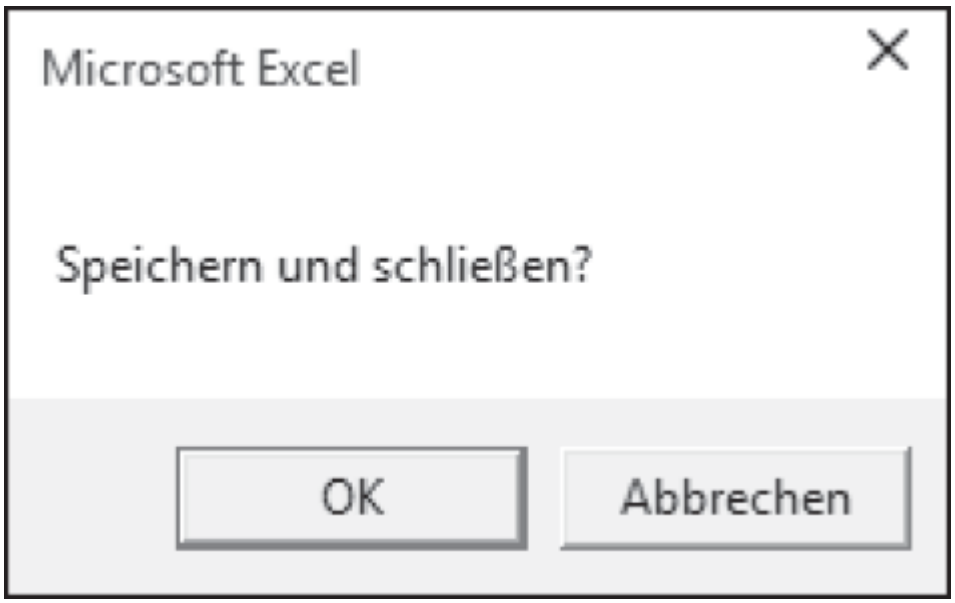


Abbildung 4.3: Wenn Sie versuchen, die Arbeitsmappe zu schließen, wird dieses Meldungsfeld angezeigt.

Für die Auswertung der Benutzereingabe wird die Anweisung `Select Case` verwendet. `Select Case` ist eine Alternative zur `If-Then-Else`-Anweisung, die es Ihnen erlaubt, in Ihren Makros Bedingungen zu überprüfen. Das allgemeine Format der `Select-Case`-Anweisung ist recht einfach:

```
Select Case &lt;Ausdruck, der geprüft wird&gt;
Case Is = &lt;ein Wert&gt;
 &lt;etwas machen&gt;
Case Is=&lt;ein anderer Wert&gt;
 &lt;etwas anderes machen&gt;
Case Is=&lt;ein dritter Wert&gt;
 &lt;etwas drittes machen&gt;
End Select
```

Mit `Select Case` können Sie mehrere Bedingungen auf einmal überprüfen. In unserem Beispiel wird lediglich geprüft, ob der Anwender auf OK oder ABBRECHEN geklickt hat. Werfen Sie einen Blick auf den Code:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

'Schritt 1: Meldungsfeld anzeigen und Überprüfung starten
 Select Case MsgBox("Speichern und schließen?", _
 vbOKCancel)
```

```
'Schritt 2: Abbrechen-Schaltfläche wurde angeklickt,
' also das Schließen abbrechen
 Case Is = vbCancel
 Cancel = True

'Schritt 3: OK-Schaltfläche wurde ausgewählt,
' also Arbeitsmappe speichern und schließen
 Case Is = vbOK
 ActiveWorkbook.Save

'Schritt 4: Die Select-Case-Anweisung beenden
 End Select
End Sub
```

1. In Schritt 1 lassen Sie das Meldungsfeld anzeigen; das Ergebnis des Meldungsfelds liefert den Ausdruck, der in der `Select-Case`-Anweisung geprüft wird. Hier geben Sie als zweiten Parameter `vbOKCancel` an, damit im Meldungsfeld lediglich die Schaltflächen OK und ABBRECHEN angezeigt werden.
2. Wenn der Anwender im Meldungsfeld auf ABBRECHEN klickt, teilt das Makro Excel in Schritt 2 mit, dass das Ereignis `Workbook_Close` abgebrochen werden soll. Hierfür wird der booleschen Eigenschaft `Cancel` der Wert `True` zugewiesen (damit wird die Aktion abgebrochen, was dazu führt, dass die Arbeitsmappe nicht geschlossen wird).
3. Wenn der Anwender auf OK klickt, wird der Code in Schritt 3 ausgeführt. Hier weisen Sie Excel an, die Arbeitsmappe zu speichern. Da Sie die Eigenschaft `Cancel` nicht auf `True` setzen, wird die Arbeitsmappe danach von Excel geschlossen.
4. In Schritt 4 wird die `Select-Case`-Anweisung beendet. Immer, wenn Sie mit `Select Case` eine `Select-Case`-Anweisung einleiten, müssen Sie sie mit einem korrespondierenden `End Select` abschließen.

So verwenden Sie es

Um dieses Makro zu implementieren, müssen Sie es kopieren und in die Ereignisprozedur für das Ereignis `Workbook_BeforeClose` einfügen. Indem Sie das Makro so erstellen, gewährleisten Sie, dass es immer dann ausgeführt wird, wenn die Arbeitsmappe geschlossen wird:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie auf DIESEARBEITSMAPPE.**
4. **Wählen Sie in der Drop-down-Liste EREIGNISSE (siehe Abbildung 4.4) das Ereignis BEFORECLOSE aus.**

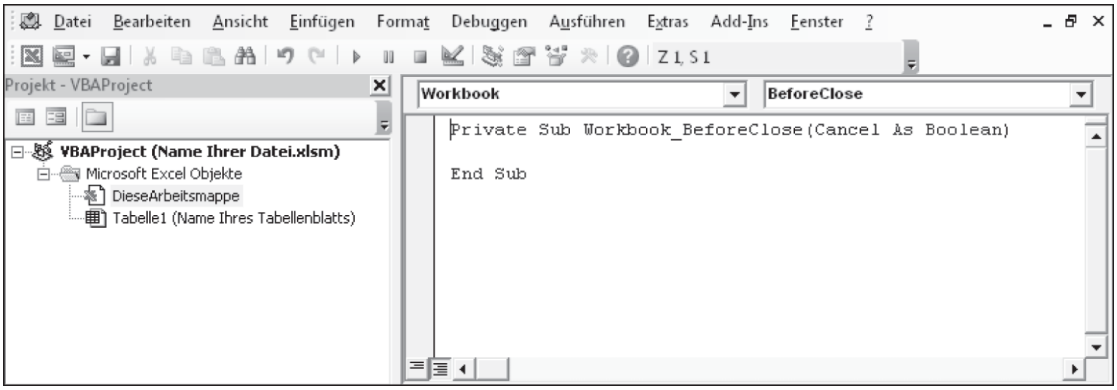


Abbildung 4.4: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis `BeforeClose` ein.

5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin und passen Sie gegebenenfalls den Zellbereich an.**

Ein Tabellenblatt vor dem Schließen schützen

Mitunter wollen Sie Ihre Arbeitsmappe mit anderen teilen, wobei jedoch bestimmte Tabellenblätter mit einem Blattschutz versehen

sein sollen. Wenn Sie diese Schritte nicht immer manuell machen wollen, zeigt Ihnen dieses Makro, wie sich so eine Aufgabe automatisieren lässt.

So funktioniert es

Auch dieser Code wird vom Arbeitsmappenereignis `BeforeClose` ausgelöst. Wenn Sie die Arbeitsmappe schließen, wird dieses Ereignis ausgelöst und der Code der betreffenden Ereignisprozedur ausgeführt. Das Makro schützt das angegebene Tabellenblatt mit dem im Code festgelegten Kennwort und speichert dann die Arbeitsmappe:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

'Schritt 1: Das Tabellenblatt mit einem Kennwort schützen
 Sheets("Tabelle1").Protect Password:="ROT"

'Schritt 2: Die Arbeitsmappe speichern
 ActiveWorkbook.Save

End Sub
```

1. In Schritt 1 legen Sie explizit fest, welches Tabellenblatt geschützt werden soll, hier das Blatt mit dem Namen `Tabelle1`. Außerdem verwenden Sie den Parameter für das Festlegen des Kennworts, `Password:="ROT"`, mit dem das für das Aufheben des Blattschutzes benötigte Kennwort definiert wird.

   Der Parameter `Password` ist optional. Wenn Sie kein Kennwort angeben, wird das Tabellenblatt zwar geschützt; Sie benötigen dann aber kein Kennwort, um den Blattschutz aufzuheben.

   Excel unterscheidet bei Kennwörtern zwischen Groß- und Kleinschreibung. Achten Sie daher bei der Eingabe des richtigen Kennworts auch auf Groß- und Kleinschreibung.
2. In Schritt 2 wird die Arbeitsmappe gespeichert. Wenn Sie die Arbeitsmappe nicht speichern, geht der soeben zugewiesene Blattschutz verloren und die Mappe kann das nächste Mal doch wieder ohne Kennwort geöffnet werden.

So verwenden Sie es

Um dieses Makro zu implementieren, müssen Sie es kopieren und in die Ereignisprozedur für das Ereignis `Workbook_BeforeClose` einfügen. Auf diese Art und Weise gewährleisten Sie, dass das Makro immer dann ausgeführt wird, wenn die Arbeitsmappe geschlossen wird:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie auf DIESEARBEITSMAPPE.**
4. **Wählen Sie in der Drop-down-Liste EREIGNISSE (siehe Abbildung 4.5) das Ereignis BEFORECLOSE aus.**

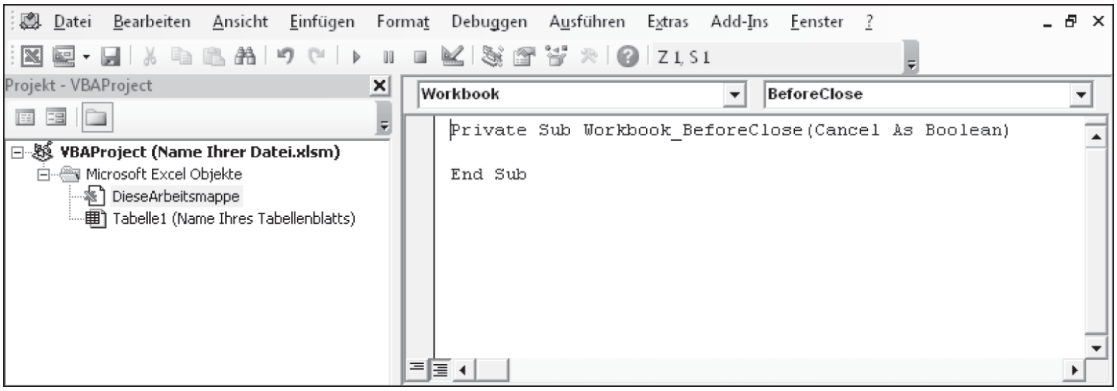


Abbildung 4.5: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis BeforeClose ein.

5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin und passen Sie den Namen des Tabellenblatts und das Kennwort an.**

Beachten Sie, dass Sie weitere Tabellenblätter schützen können, indem Sie vor der Anweisung `ActiveWorkbook.Save` weitere Codezeilen für andere Tabellenblätter einfügen.

Beim Öffnen einer Arbeitsmappe den Blattschutz aufheben

Wenn Sie Arbeitsmappen mit geschützten Tabellenblättern weitergegeben haben, ist die Wahrscheinlichkeit groß, dass Sie diese Arbeitsmappen zurückerhalten, wobei die Tabellenblätter weiterhin geschützt sind. Es kann sein, dass Sie an diesen Tabellenblättern weiterarbeiten und dazu vorher den Blattschutz wieder aufheben müssen. Damit Sie dies nicht immer wieder von Hand machen müssen, können Sie den Vorgang mit diesem Makro automatisieren.

So funktioniert es

Dieser Code wird durch das Arbeitsmappenereignis `Open` ausgelöst, das automatisch ausgelöst wird, wenn eine Arbeitsmappe geöffnet wird. Das Makro hebt beim Öffnen der Arbeitsmappe den Blattschutz für das angegebene Tabellenblatt auf und verwendet dazu das angegebene Kennwort:

```
Private Sub Workbook_Open()
'Schritt 1: Blattschutz für das angegebene
' Tabellenblatt aufheben
 Sheets("Tabelle1").Unprotect Password:="ROT"
End Sub
```

Im Makro wird der Name des Tabellenblatts, dessen Blattschutz aufgehoben werden soll, explizit angegeben. In unserem Beispiel ist es das Blatt `Tabelle1`. Anschließend wird das zum Aufheben des Blattschutzes erforderliche Kennwort angegeben. Excel unterscheidet bei Kennwörtern Groß- und Kleinschreibung. Achten Sie daher auf die Eingabe des richtigen Kennworts und insbesondere auch auf Groß- und Kleinschreibung.

So verwenden Sie es

Um dieses Makro zu implementieren, müssen Sie es kopieren und in die Ereignisprozedur für das Ereignis `Workbook_Open` einfügen. Auf diese Art und Weise gewährleisten Sie, dass das Makro immer dann ausgeführt wird, wenn die Arbeitsmappe geöffnet wird:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie auf DIESEARBEITSMAPPE.**
4. **Wählen Sie in der Drop-down-Lliste EREIGNISSE (siehe Abbildung 4.6) das Ereignis OPEN aus.**
5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin und passen Sie falls erforderlich den Namen des Tabellenblatts und das Kennwort an.**

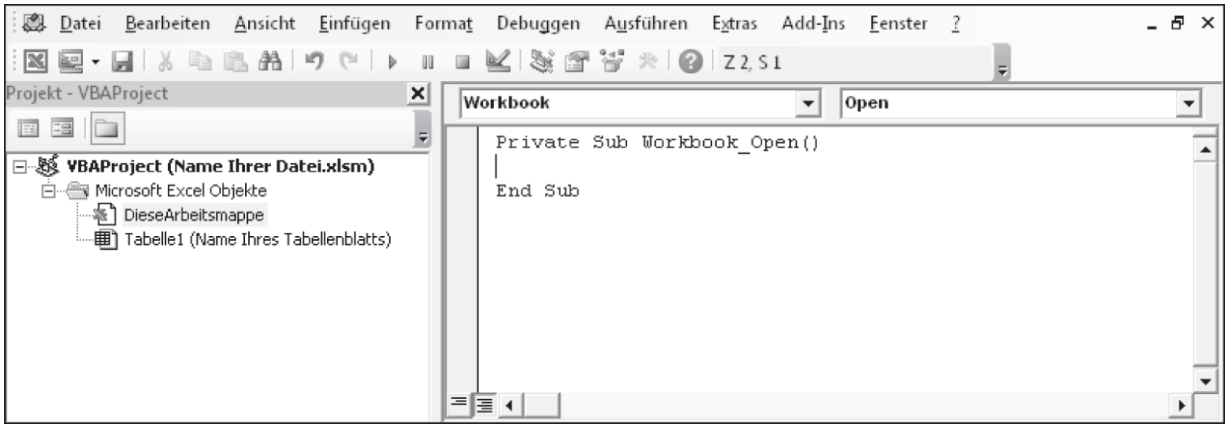


Abbildung 4.6: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis Open ein.

Beim Öffnen einer Arbeitsmappe ein bestimmtes Tabellenblatt anzeigen

Manchmal ist es zwingend erforderlich, dass das Arbeiten an einer Arbeitsmappe auf einem bestimmten Tabellenblatt beginnt. Mit dem Makro dieses Abschnitts geraten Ihre Anwender nicht auf Abwege, da beim Öffnen der Arbeitsmappe automatisch das richtige Tabellenblatt angezeigt wird.

Im Beispiel in Abbildung 4.7 soll nach dem Öffnen der Arbeitsmappe sofort das Tabellenblatt *Hier starten* angezeigt werden.

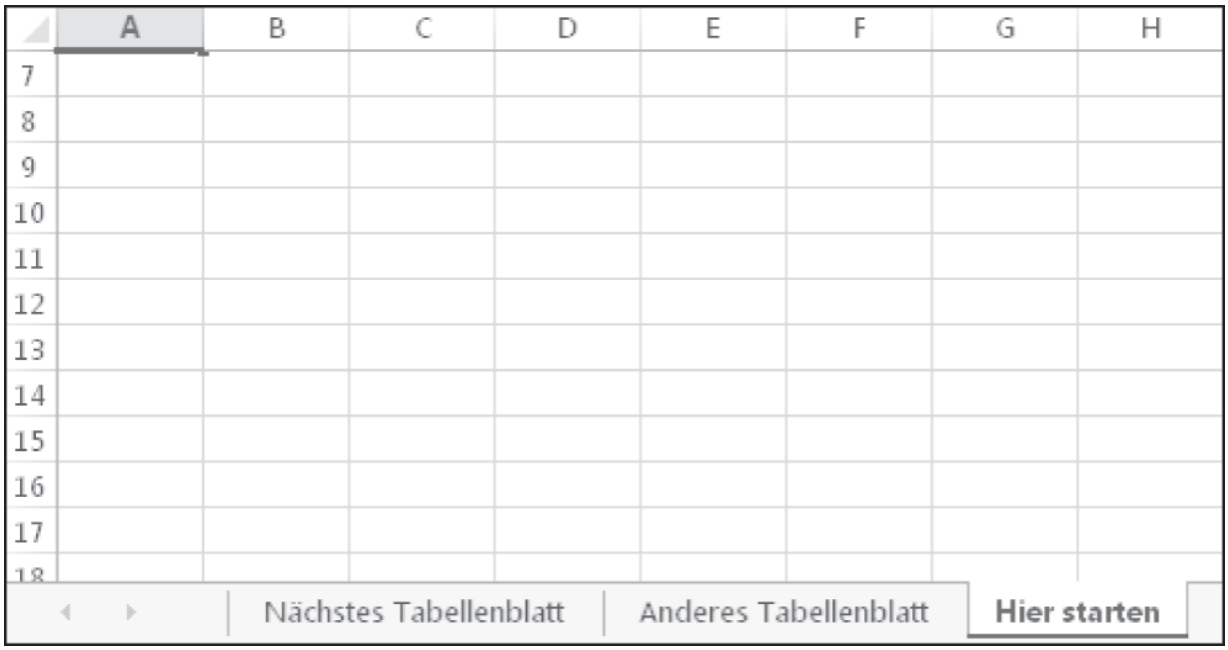


Abbildung 4.7: Sie möchten erreichen, dass beim Öffnen der Arbeitsmappe das Tabellenblatt »Hier starten« angezeigt wird.

So funktioniert es

Das Makro verwendet das Arbeitsmappenereignis `Open`, um nach dem Öffnen der Arbeitsmappe das angegebene Tabellenblatt zu aktivieren:

```
Private Sub Workbook_Open()

'Schritt 1: Das angegebene Tabellenblatt anzeigen
 Sheets("Hier starten").Select

End Sub
```

Im Makro wird außerdem der Name des Tabellenblatts angegeben, das beim Öffnen der Arbeitsmappe aktiviert werden soll.

So verwenden Sie es

Um dieses Makro zu implementieren, müssen Sie es kopieren und in die Ereignisprozedur für das Ereignis `Workbook_Open` einfügen. Auf diese Art und Weise gewährleisten Sie, dass das Makro immer dann ausgeführt wird, wenn die Arbeitsmappe geöffnet wird:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie auf DIESEARBEITSMAPPE.**
4. **Wählen Sie in der Drop-down-Liste EREIGNISSE (siehe Abbildung 4.8) das Ereignis OPEN aus.**
5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin und passen Sie den Namen des Tabellenblatts an.**

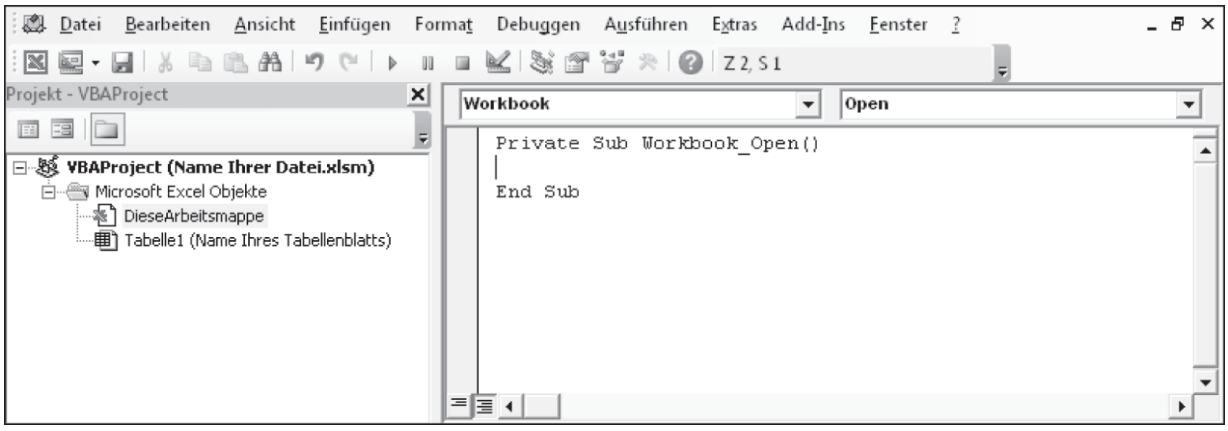


Abbildung 4.8: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis Open ein.

Eine vom Benutzer ausgewählte Arbeitsmappe öffnen

Wollen Sie Ihren Anwendern oder sich selbst eine schnelle Methode zur Verfügung stellen, eine Datei zu suchen und zu öffnen? Dieses Makro verwendet eine einfache Technik, um ein ÖFFNEN-Dialogfeld anzuzeigen, in dem nach einer Datei gesucht und die gefundene Datei geöffnet werden kann.

So funktioniert es

Dieses Makro zeigt das ÖFFNEN-Dialogfeld aus Abbildung 4.9 an; hier kann der Anwender eine Datei suchen und diese öffnen.

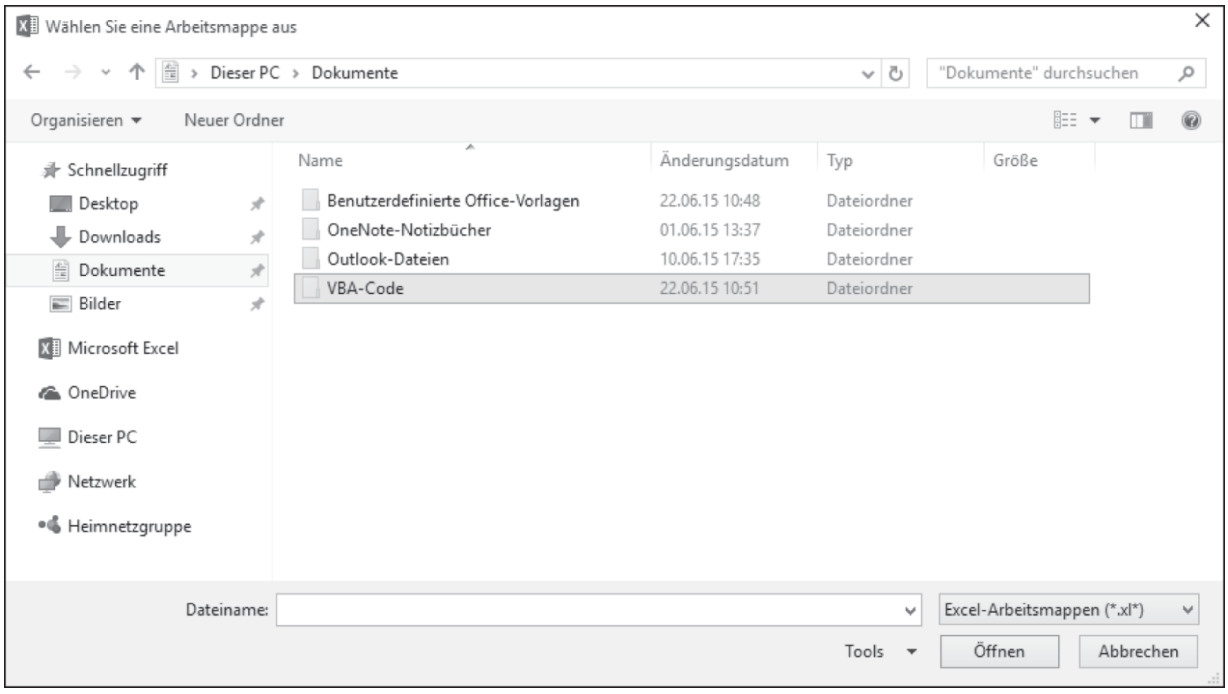


Abbildung 4.9: Das von einem Makro aktivierte Dialogfeld zum Öffnen einer Datei

Das Makro sieht so aus:

```
Sub Makro1()
```

```
'Schritt 1: Eine Variable des Typs Variant definieren
 Dim FName As Variant

'Schritt 2: Die Methode GetOpenFilename aktiviert das
' Dialogfeld

 FName = Application.GetOpenFilename( _
 FileFilter:="Excel-Arbeitsmappen,*.xl*", _
 Title:="Wählen Sie eine Arbeitsmappe aus", _
 MultiSelect:=False)

'Schritt 3: Falls eine Datei ausgewählt wurde,
' diese öffnen!

 If FName &lt;&gt; False Then
 Workbooks.Open Filename:=FName
 End If

End Sub
```

1. Zuerst wird eine Variable des Typs `Variant` deklariert, in der der Name der vom Benutzer ausgewählten Datei abgespeichert wird. Der Name der Variablen lautet `FName`.
2. In Schritt 2 verwenden Sie die Methode `GetOpenFilename`, die ein Dialogfeld anzeigt, in dem Sie nach einer Datei suchen und diese auswählen können.

   Die Methode `GetOpenFilename` unterstützt verschiedene Parameter. Mit dem Parameter `FileFilter` können Sie den Dateityp festlegen, nach dem Sie suchen. Mit dem Parameter `Title` legen Sie den Text fest, der in der Titelleiste des Dialogfelds angezeigt werden soll. Wenn der Parameter `MultiSelect` den Wert `False` hat, ist die Auswahl auf eine einzelne Datei beschränkt.
3. Wenn der Anwender im Dialogfeld eine Datei auswählt, befindet sich der Name der ausgewählten Datei in der Variablen `FName`. In Schritt 3 prüfen Sie, ob diese Variable leer ist oder nicht. Falls die Variable einen Dateinamen enthält, verwenden Sie die Methode `Open` des `Workbook`-Objekts, um die Datei zu öffnen.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es dann in ein Standardmodul ein:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen Ihres Projekts/Ihrer Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**
5. **Optional können Sie das Makro einer Schaltfläche zuweisen (siehe Kapitel 1).**

Feststellen, ob eine Arbeitsmappe bereits geöffnet ist

Das vorherige Makro öffnet eine vom Anwender ausgewählte Arbeitsmappe. Falls Sie Arbeitsmappen automatisch öffnen wollen, sollten Sie kurz überlegen, was passiert, wenn Sie versuchen, eine Arbeitsmappe zu öffnen, die bereits geöffnet ist. Ohne eine VBA-Lösung versucht Excel, die Datei erneut zu öffnen und zeigt dann einen Warnhinweis an (siehe Abbildung 4.10). Sie können diesen Warnhinweis unterdrücken, indem Sie in Ihrem Makro vor dem Öffnen einer Arbeitsmappe prüfen, ob diese bereits geöffnet ist.

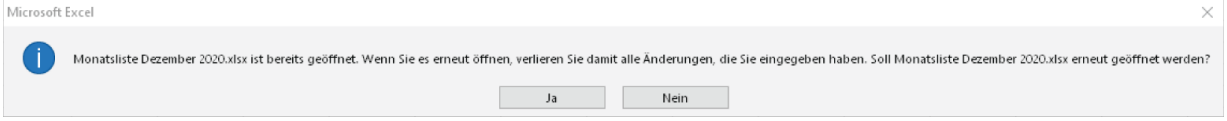


Abbildung 4.10: Sie können diese nervige Meldung vermeiden, die angezeigt wird, wenn Sie eine Arbeitsmappe öffnen wollen, die bereits geöffnet ist.

So funktioniert es

Das Erste, was Ihnen an diesem Makro auffällt, ist, dass es sich hier um eine `Function`- und nicht um eine `Sub`-Prozedur handelt. Wie Sie noch sehen werden, erlaubt Ihnen die Verwendung einer `Function`-Prozedur, dem Makro einen beliebigen Dateinamen als Parameter zu übergeben. Sie können so für beliebige Dateien testen, ob sie bereits geöffnet sind oder nicht.

Der Code der Funktion ist recht einfach. Sie überprüfen, ob eine Datei geöffnet ist, indem Sie den Dateinamen einer Objektvariablen zuweisen. Nur Arbeitsmappen, die geöffnet sind, lassen sich einer Objektvariablen zuweisen. Falls Sie versuchen, der Variablen eine nicht geöffnete Arbeitsmappe zuzuweisen, wird ein Fehler erzeugt.

Falls der Dateiname zugewiesen werden kann, ist die Arbeitsmappe geöffnet; wenn ein Fehler auftritt, dann ist sie geschlossen.

```
Function FileIsOpenTest(TargetWorkbook As String) _
 As Boolean

'Schritt 1: Variable deklarieren
 Dim TestBook As Workbook

'Schritt 2: Excel anweisen, Makroausführung bei
' Fehler fortzusetzen
 On Error Resume Next

'Schritt 3: Versuchen, TargetWorkbook an
' TestBook zuzuweisen
 Set TestBook = Workbooks(TargetWorkbook)

'Schritt 4: Falls kein Fehler auftritt, ist die ausgewählte
' Arbeitsmappe bereits geöffnet
 If Err.Number = 0 Then
 FileIsOpenTest = True
 Else
 FileIsOpenTest = False
 End If

End Function
```

1. Im Makro wird zuerst eine Variable des Typs `Workbook` definiert, die in Schritt 3 verwendet wird.
2. In Schritt 2 weisen Sie Excel an, beim Auftreten eines Fehlers während der Codeausführung mit der Abarbeitung des VBA-Codes weiterzumachen. Ohne diese Anweisung würde der Code beim Auftreten eines Fehlers einfach angehalten. Wie bereits gesagt, überprüfen Sie einen bestimmten Dateinamen, indem Sie ihn einer Objektvariablen zuweisen. Wenn die Zuweisung erfolgreich verläuft, ist die Arbeitsmappe geöffnet; wenn ein Fehler auftritt, ist sie geschlossen.
3. In Schritt 3 versuchen Sie also, der Objektvariablen `Testbook` den Namen einer Arbeitsmappe zuzuweisen. Der Name wird an die Funktion in einer Variablen des Typs `String` mit dem Namen `TargetWorkbook` übergeben (siehe erste Zeile des Codes). Hierdurch wird der Name der Arbeitsmappe nicht hart-kodiert und Sie können ihn der Funktion als Argument übergeben.
4. In Schritt 4 überprüfen Sie, ob ein Fehler aufgetreten ist oder nicht. Wenn kein Fehler auftrat, dann ist die Arbeitsmappe bereits geöffnet, und Sie weisen `FileIsOpenTest` den Wert `True` zu. Sollte ein Fehler auftreten, dann ist die Arbeitsmappe nicht geöffnet, und Sie weisen `FileIsOpenTest` den Wert `False` zu.

Wie bereits gesagt, kann die Funktion jede beliebige Arbeitsmappe überprüfen, da Sie ihr den Namen der zu untersuchenden Arbeitsmappe im Parameter `TargetWorkbook` übergeben. Dies ist der Vorteil bei der Verwendung von Funktionen in einem Makro.

Das folgende Makro zeigt, wie Sie die Funktion einsetzen können. Sie verwenden hier das Makro, das Sie bereits aus dem Abschnitt »Eine vom Benutzer ausgewählte Arbeitsmappe öffnen« kennen. Jedoch rufen Sie hier zusätzlich die neue Funktion `FileIsOpenTest` auf, um zu überprüfen, dass der Anwender keine Datei zu öffnen versucht, die bereits geöffnet ist.

```
Sub Makro1()

'Schritt 1: Variablen definieren
 Dim FName As Variant
 Dim FNFileOnly As String


'Schritt 2: GetOpenFilename aktiviert das Dialogfeld
 FName = Application.GetOpenFilename( _
 FileFilter:="Excel-Arbeitsmappen,*.xl*", _
 Title:="Wählen Sie eine Arbeitsmappe aus", _
 MultiSelect:=False)

'Schritt 3: Die ausgewählte Datei öffnen, falls sie
' nicht bereits geöffnet ist
 If FName &lt;&gt; False Then
 FNFileOnly = StrReverse(Left(StrReverse(FName), _
 InStr(StrReverse(FName), "\") - 1))

 If FileIsOpenTest(FNFileOnly) = True Then
 MsgBox "Die ausgewählte Datei ist bereits geöffnet"
 Else
 Workbooks.Open Filename:=FName
 End If
 End If

End Sub
```

Nachdem Sie dieses Makro implementiert haben, erhalten Sie die übersichtliche und klare Fehlermeldung aus Abbildung 4.11.

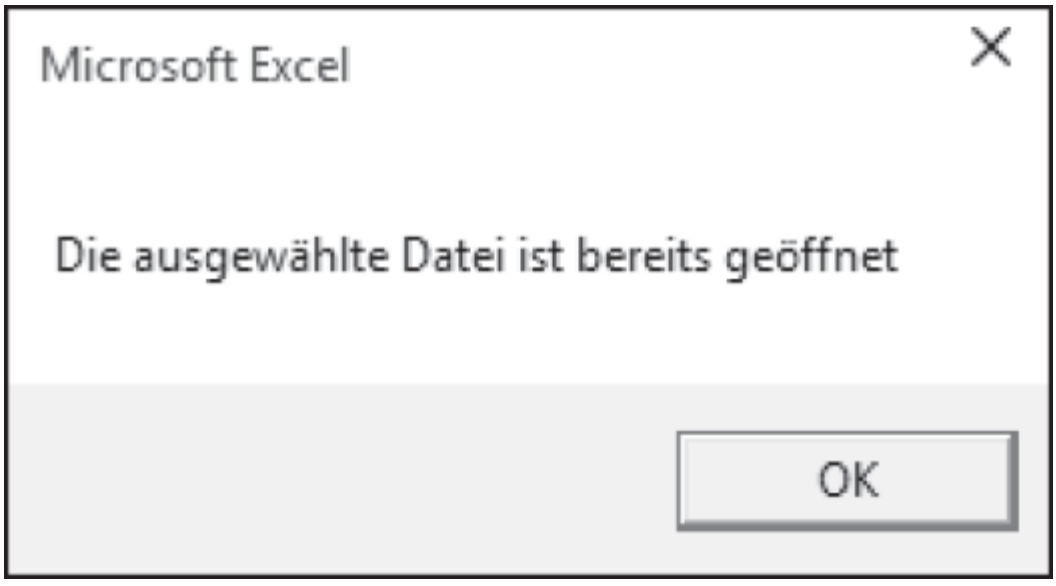


Abbildung 4.11: Eine deutliche und gut lesbare Meldung

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie beide Prozeduren und fügen sie dann in ein Standardmodul ein:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen Ihres Projekts/Ihrer Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**
5. **Optional können Sie das Makro einer Schaltfläche zuweisen (siehe Kapitel 1).**

Überprüfen, ob eine Arbeitsmappe in einem Ordner vorhanden ist

Möglicherweise folgen Sie regelmäßig einem bestimmten Prozess, in dem eine Datei irgendwo auf Ihrem Computer bearbeitet wird. Beispielsweise müssen Sie täglich eine bestimmte Arbeitsmappe öffnen, um dort neue Daten einzufügen. In diesen Fällen sollten Sie überprüfen, ob die Datei, die Sie bearbeiten wollen, (immer noch) existiert. Dieses Makro erlaubt es, den Test für beliebige Dateien durchzuführen.

So funktioniert es

Auch bei diesem Makro handelt es sich um eine `Function`- und keine `Sub`-Prozedur. Hierdurch ist es möglich, beliebige Dateien daraufhin zu überprüfen, ob sie vorhanden sind oder nicht.

Dieses Makro verwendet die Funktion `Dir`. Die Funktion `Dir` gibt eine Zeichenkette mit dem Dateinamen der Datei zurück, dessen kompletten Pfadnamen Sie an `Dir` übergeben. Diese Funktion kann für verschiedene Zwecke eingesetzt werden. In diesem Beispiel wird sie lediglich verwendet, um zu prüfen, ob der Dateipfad existiert, den Sie übergeben wollen:

```
Function FileExists(FPath As String) As Boolean

'Schritt 1: Ihre Variablen deklarieren
 Dim FName As String

'Schritt 2: Die Funktion Dir verwenden, um den Dateinamen
' zu ermitteln
 FName = Dir(FPath)

'Schritt 3: Wenn die Datei vorhanden ist, True
' zurückgeben, sonst False
 If FName &lt;&gt; "" Then FileExists = True _
 Else FileExists = False

End Function
```

1. In Schritt 1 deklarieren Sie eine Variable des Typs `String`, in welcher der von `Dir` zurückgegebene Dateiname gespeichert wird. Der Name der Variablen lautet `FName`.
2. In Schritt 2 versuchen Sie, `FName` einen Wert zuzuweisen. Hierzu übergeben Sie die Variable `FPath` an die Funktion `Dir`. Die Variable `FPath` wird an die Funktion `FileExists` übergeben (siehe erste Zeile des Codes). Diese Struktur erlaubt es Ihnen, die Funktion für beliebige Dateien zu verwenden, da der Name der Datei nicht im Code hartverdrahtet ist, sondern als Parameter übergeben wird.
3. Wenn die Variable `FName` nicht zugewiesen werden kann, dann existiert der übergebene Pfad nicht. Die Variable `FName` ist dann also leer. In Schritt 3 wird dieses Ergebnis einfach nur in den Wert `True` oder `False` übersetzt.

Wie bereits gesagt, kann die Funktion für jeden übergebenen Pfadnamen überprüfen, ob die angegebene Datei existiert oder nicht. Dies ist der Vorteil bei der Verwendung von Funktionen in einem Makro.

Das folgende Makro zeigt, wie die Funktion `FileExists` eingesetzt werden kann:

```
Sub Makro1()

 If FileExists("C:\Temp\MeineNeueArbeitsmappe.xlsx") = _
 True Then
 MsgBox "Datei existiert bereits."
 Else
 MsgBox "Datei ist nicht vorhanden."
 End If

End Sub
```

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie beide Prozeduren und fügen sie dann in ein Standardmodul ein:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt+F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen Ihres Projekts/Ihrer Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Alle Arbeitsmappen auf einmal schließen

Eine der etwas nervigeren Aufgaben in Excel ist es, mehrere Arbeitsmappen auf einmal zu schließen. Hierzu müssen Sie jede

einzelne geöffnete Arbeitsmappe aktivieren, sie schließen und dann auch noch jedes Mal bestätigen, dass die Änderungen gespeichert werden sollen. Einfacher wäre es, dies für alle geöffneten Mappen in einem Rutsch zu tun. Das folgende kleine Makro löst dieses Problem.

So funktioniert es

In diesem Makro wird die Sammlung `Workbooks` in einer Schleife durchlaufen; hierbei wird jede geöffnete Arbeitsmappe gespeichert und anschließend geschlossen:

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim wb As Workbook

'Schritt 2: Alle Arbeitsmappen durchlaufen, diese
' speichern und schließen
 For Each wb In Workbooks
 wb.Close SaveChanges:=True
 Next wb

End Sub
```

1. In Schritt 1 wird eine Objektvariable deklariert, die ein `Workbook`-Objekt repräsentiert. Dies erlaubt es Ihnen, alle geöffneten Arbeitsmappen in einer Schleife zu durchlaufen.
2. Die Schleife selbst sehen Sie in Schritt 2: Hier werden alle geöffneten Arbeitsmappen durchlaufen, diese zuerst gespeichert und dann geschlossen. Falls Sie die Arbeitsmappen nicht speichern wollen, ändern Sie den Wert des Arguments `SaveChanges` von `True` in `False` ab.

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-

Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Alle Arbeitsmappen in einem Ordner drucken

Falls Sie mehrere Arbeitsmappen drucken wollen, die sich im gleichen Ordner befinden, können Sie dieses Makro einsetzen.

So funktioniert es

In diesem Makro verwenden Sie die `Dir`-Funktion, die eine Liste aller Dateien zurückgibt, die dem an die Funktion übergebenen

Muster entsprechen, in diesem Fall alle .xlsx-Dateien in einem bestimmten Verzeichnis. Sie durchlaufen die Liste mit den Namen der ermittelten Dateien, öffnen jede Datei, drucken sie aus und schließen dann die Datei wieder.

```
'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyFiles As String

'Schritt 2: Legen Sie den Zielordner fest
 MyFiles = Dir("C:\Temp\*.xlsx")
 Do While MyFiles &lt;&gt; ""

'Schritt 3: öffnen Sie nacheinander alle Arbeitsmappen in
' diesem Ordner und drucken Sie sie
 Workbooks.Open "C:\Temp\" & MyFiles
 ActiveWorkbook.Sheets("Tabelle1").PrintOut Copies:=1
 ActiveWorkbook.Close SaveChanges:=False

'Schritt 4: Nächste Datei im Ordner auswählen
 MyFiles = Dir
 Loop

End Sub
```

1. In Schritt 1 deklarieren Sie die `String`-Variable `MyFiles`, die alle Dateinamen der Auflistung erhält.
2. In Schritt 2 verwenden Sie die Funktion `Dir` und geben den Ordner und den Dateityp an, den Sie suchen. Beachten Sie, dass im Code nach Dateien mit der Endung ».xlsx« gesucht wird; hierdurch werden nur Dateien im neuen Excel-Dateiformat gefunden. Falls Sie Dateien im älteren Format suchen, müssen Sie hier ».xls« angeben und eventuell den Pfadnamen anpassen. Das Makro übergibt die Namen aller gefundenen Dateien an die `String`-Variable `MyFiles`.
3. In Schritt 3 wird die Datei geöffnet und dann ein Exemplar ihres Tabellenblatts `Tabelle1` gedruckt. Selbstverständlich müssen Sie dies anpassen, wenn Sie andere Tabellenblätter drucken wollen. Sie können hier außerdem die Anzahl der Exemplare ändern, die gedruckt werden sollen.

4. In Schritt 4 wird die nächste Datei ermittelt, die mit dem Suchmuster übereinstimmt. Wenn keine Dateien mehr vorhanden sind, dann ist die Variable `MyFiles` leer. In diesem Fall werden zuerst die Schleife und dann das Makro beendet.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie den Code und fügen ihn dann in ein Standardmodul ein:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen Ihres Projekts/Ihrer Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein, und passen Sie die Anweisungen für das Drucken wie gewünscht an.**

Arbeitsmappe erst dann schließen, wenn eine bestimmte Zelle Inhalte besitzt

Es kann Situationen geben, in denen es sinnvoll sein ist, dass eine Arbeitsmappe erst dann geschlossen werden darf, wenn der Anwender bestimmte Informationen eingegeben hat. Also wäre es ganz nützlich, wenn Sie das Schließen der Arbeitsmappe so lange unterbinden könnten, bis eine bestimmte kritische Zelle mit Inhalten gefüllt ist (siehe Abbildung 4.12). Für diesen Zweck ist dieses Makro gedacht.

Abbildung 4.12: Sie können das Schließen einer Arbeitsmappe verhindern, solange eine bestimmte Zelle leer ist.

So funktioniert es

Der Code wird durch das Arbeitsmappenereignis `BeforeClose` ausgelöst. Beim Schließen der Arbeitsmappe wird dieses Ereignis von Excel ausgelöst und der Code der betreffenden Ereignisprozedur ausgeführt. Das Makro prüft, ob die Zielzelle (hier C7) leer ist. Falls die Zelle leer ist, wird das Schließen abgebrochen. Wenn die Zelle nicht leer ist, wird die Arbeitsmappe gespeichert und geschlossen:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

'Schritt 1: Prüfen, ob Zelle C7 leer ist
If Sheets("Tabelle 1").Range("C7").Value = "" Then

'Schritt 2: Leer: Schließen abbrechen und den
```

```
' Benutzer informieren
 Cancel = True
 MsgBox "Zelle C7 darf nicht leer sein."

'Schritt 3: Nicht leer: Speichern und Schließen
Else
 ActiveWorkbook.Close SaveChanges:=True
End If

End Sub
```

1. In Schritt 1 wird geprüft, ob Zelle C7 auf dem Blatt Tabelle 1 leer ist.
2. Wenn diese Zelle leer ist, wird der Code bei Schritt 2 ausgeführt. Das Schließen der Arbeitsmappe wird abgebrochen, indem der booleschen Eigenschaft `Cancel` der Wert `True` zugewiesen wird. Außerdem wird die Methode `MsgBox` verwendet, um den Benutzer hierüber zu informieren.
3. Falls die Zelle C7 nicht leer ist, wird der Code bei Schritt 3 aktiv: Die Arbeitsmappe wird gespeichert und geschlossen.

So verwenden Sie es

Um dieses Makro zu implementieren, müssen Sie es kopieren und in die Ereignisprozedur für das Ereignis `Workbook_BeforeClose` einfügen. Auf diese Weise gewährleisten Sie, dass das Makro immer dann ausgeführt wird, wenn die Arbeitsmappe geschlossen wird:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie auf DIESEARBEITSMAPPE.**
4. **Wählen Sie in der Drop-down-Liste EREIGNISSE (siehe Abbildung 4.13) das Ereignis BEFORECLOSE aus.**

5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin.**

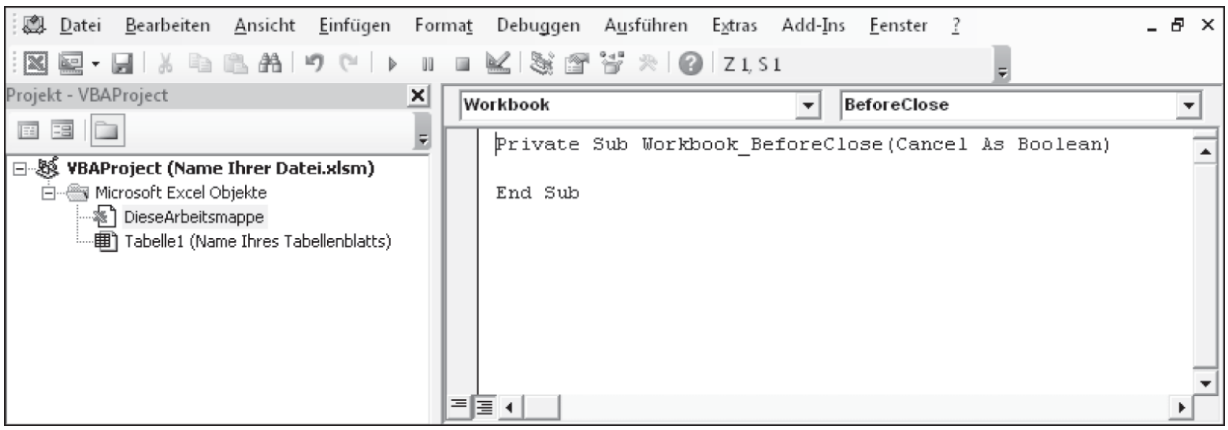


Abbildung 4.13: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis `BeforeClose` ein.

Backup der aktuellen Arbeitsmappe mit aktuellem Datum erstellen

Wir alle wissen, dass es wichtig ist, Sicherungskopien der eigenen Arbeit anzufertigen. Sie können dies auch von einem Makro erledigen lassen. Dieses einfache Makro speichert Ihre Arbeitsmappe unter einem neuen Namen, bei dem das aktuelle Datum ein Teil des Dateinamens ist.

So funktioniert es

Das Trickreiche an diesem Makro besteht darin, den neuen Dateinamen auf intelligente Weise zusammenzubauen. Der neue Dateiname besteht aus drei Teilen: dem Pfad, dem aktuellen Datum und dem ursprünglichen Dateinamen.

Der Pfad wird mit der Eigenschaft `Path` des Objekts `ThisWorkbook` bestimmt, das aktuelle Datum mit der Funktion `Date`.

Bitte achten Sie darauf, dass Sie das Datum korrekt formatieren: `Format(Date, "DD.MM.YY")`. Dies liegt daran, dass die Datumsfunktion das Datum standardmäßig im Format `mm/dd/yyyy` zurückgibt. Sie verwenden Punkte anstelle von Schrägstrichen, weil die Schrägstriche in Dateinamen nicht erlaubt sind und das Speichern der Datei fehlschlagen würde. Außerdem bekommen Sie auf diese Weise das deutsche Datumsformat.

Der letzte Teil des neuen Dateinamens ist der ursprüngliche Dateiname. Diese Info finden Sie mit der Eigenschaft `Name` des Objekts `ThisWorkbook`:

```
Sub Makro1()

'Schritt 1: Die Arbeitsmappe unter einem anderen
' Namen speichern
ThisWorkbook.SaveCopyAs _
 Filename:=ThisWorkbook.Path & "\" & _
 Format(Date, "DD.MM.YY") & " " & _
 ThisWorkbook.Name

End Sub
```

Dieses Makro erzeugt in einem einzigen Schritt den neuen Dateinamen und verwendet gleichzeitig die Methode `SaveCopyAs`, um die Datei zu speichern.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie den Code und fügen ihn dann in ein Standardmodul ein:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen Ihres Projekts/Ihrer Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein, und passen Sie die Anweisungen für das Drucken wie gewünscht an.**

Kapitel 5

Makros für Tabellenblätter

IN DIESEM KAPITEL

- Tabellenblätter einfügen, benennen, kopieren und löschen
- Tabellenblätter ausblenden, einblenden, verschieben, sortieren und gruppieren
- Alle Tabellenblätter schützen oder den Blattschutz wieder aufheben
- Ein Inhaltsverzeichnis Ihrer Tabellenblätter erstellen und drucken
- In einem Tabellenblatt rein- oder rauszoomen

Excel-Analysten müssen häufig Aufgaben automatisieren, die sich auf Tabellenblätter beziehen. Ob es darum geht, alle Tabellenblätter einer Arbeitsmappe einzublenden oder gleichzeitig alle Tabellenblätter zu drucken –viele Aufgaben lassen sich automatisieren, um Zeit zu sparen und die Arbeit effizienter zu erledigen. Dieses Kapitel stellt einige Makros vor, die bei der Arbeit mit von Tabellenblättern nützlich sind.

Ein neues Tabellenblatt einfügen und benennen

Wir beginnen dieses Kapitel mit einer der einfachsten Automatisierungen für Tabellenblätter, die sich mit einem Makro erledigen lässt: das Einfügen und Benennen eines neuen Tabellenblatts.

So funktioniert es

Schauen Sie sich die folgenden Codezeilen an und Sie werden erkennen, dass das Makro ziemlich intuitiv ist:

```
Sub Makro1()

'Schritt 1: Sagen Sie Excel, was es bei einem Fehler ' machen soll
 On Error GoTo MyError

'Schritt 2: Fügen Sie ein Tabellenblatt ein und ' benennen Sie es
 Sheets.Add
 ActiveSheet.Name = WorksheetFunction.Text(Now(), _  "d.m.yyyy hh-mm-ss")
 Exit Sub

'Schritt 3: Wenn wir hier ankommen, ist ein Fehler ' aufgetreten, über den
wir den Anwender informieren
MyError:
 MsgBox "Ein Tabellenblatt mit diesem Namen ist " & _  "bereits vorhanden."

End Sub
```

1. Wir gehen davon aus, dass ein Fehler auftritt, wenn wir dem neuen Tabellenblatt einen Namen zuweisen, der bereits existiert. Daher weisen wir Excel in Schritt 1 an, direkt bei der Sprungmarke `MyError` weiterzumachen (siehe Schritt 3), falls ein Fehler auftritt.
2. Schritt 2 verwendet die Methode `Add`, um ein neues Tabellenblatt einzufügen. Standardmäßig erhalten neue Tabellenblätter den Namen Tabelle*xx*, wobei *xx* die Nummer des Blatts angibt. Sie können den Namen des Tabellenblatts ändern, indem Sie die Eigenschaft `Name` des `ActiveSheet`-Objekts setzen. In unserem Beispiel verwenden wir als Namen das aktuelle Datum und die aktuelle Uhrzeit.

   Immer wenn Sie mit VBA ein neues Tabellenblatt einfügen, wird dieses automatisch das aktive Blatt. Dieses Verhalten kennen wir von Excel. Beachten Sie, dass in Schritt 2 die Prozedur mit `Exit Sub` verlassen wird. Dies muss geschehen, damit nicht noch der Code bei Schritt 3 ausgeführt wird, der ja nur für den Fehlerfall vorgesehen ist.

3. Schritt 3 informiert den Anwender im Fehlerfall darüber, dass ein Tabellenblatt mit dem angegebenen Namen bereits existiert. Wie bereits gesagt, sollte der Code von Schritt 3 auch wirklich nur dann ausgeführt werden, wenn ein Fehler auftritt.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen Sie es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin.**

Alle Tabellenblätter bis auf das aktive löschen

Stellen Sie sich vor, Sie müssten alle Tabellenblätter bis auf das derzeit aktive Tabellenblatt löschen. Diese Aufgabe erledigt dieses Makro.

So funktioniert es

Dieses Makro durchläuft die Auflistung `Worksheets`, die alle Tabellenblätter einer Arbeitsmappe enthält. Hierbei wird der Name der Tabellenblätter mit dem Namen des aktiven Tabellenblatts verglichen. Bei jedem Schleifendurchlauf wird ein Tabellenblatt gelöscht, es sei denn, es ist das aktive Tabellenblatt. Beachten Sie in Schritt 4 die Verwendung der Eigenschaft `DisplayAlerts`. Hierdurch werden die Warnhinweise von Excel ausgeschaltet und

Sie müssen nicht das Löschen jedes einzelnen Tabellenblatts bestätigen.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim ws As Worksheet

'Schritt 2: Alle Tabellenblätter in einer ' Schleife durchlaufen
 For Each ws In ThisWorkbook.Worksheets

'Schritt 3: Prüfen, ob es sich bei ws um das aktive ' Tabellenblatt handelt
 If ws.Name &lt;&gt; ThisWorkbook.ActiveSheet.Name Then

'Schritt 4: Warnungen abschalten, Tabelle löschen ' und Warnungen wieder
einschalten
 Application.DisplayAlerts = False
 ws.Delete
 Application.DisplayAlerts = True
 End If

'Schritt 5: Mit nächstem Tabellenblatt weitermachen
 Next ws

End Sub
```

1. Das Makro deklariert zuerst die Objektvariable `ws` des Typs `Worksheet`. Hierdurch wird ein Speicherbereich für jedes Tabellenblatt angelegt, das in der Schleife bearbeitet wird.
2. In Schritt 2 leitet das Makro die Schleife ein und weist Excel an, alle Tabellenblätter in der Arbeitsmappe zu bearbeiten. Es gibt einen Unterschied zwischen `ThisWorkbook` und `ActiveWorkbook`. Das Objekt `ThisWorkbook` verweist auf die Arbeitsmappe, in der sich auch der Code befindet. Das Objekt `ActiveWorkbook` hingegen verweist auf die derzeit aktive Arbeitsmappe. Oft geben beide das gleiche Objekt zurück; wenn die Arbeitsmappe, in der sich der Code befindet, jedoch einmal nicht die aktive Arbeitsmappe ist, verweisen sie auf unterschiedliche Objekte. In unserem Beispiel wollen Sie nicht das Risiko eingehen, versehentlich Tabellenblätter in anderen Arbeitsmappen zu löschen, und setzen daher `ThisWorkbook` ein.

3. In Schritt 3 vergleicht das Makro den Namen des aktiven Tabellenblatts mit dem des Tabellenblatts, das derzeit in der Schleife bearbeitet wird.
4. In Schritt 4 wird das Tabellenblatt gelöscht, wenn der Vergleich in Schritt 3 ergab, dass die Namen unterschiedlich sind. Wie bereits erwähnt, verwenden Sie `DisplayAlerts`, um die Löschbestätigung, die Excel anfordert, auszuschalten. Falls Sie vor dem Löschen den Warnhinweis sehen wollen (siehe Abbildung 5.1), können Sie die Anweisung `Application.DisplayAlerts = False` entfernen. Auf jeden werden die Warnungen nach dem Löschen wieder eingeschaltet.

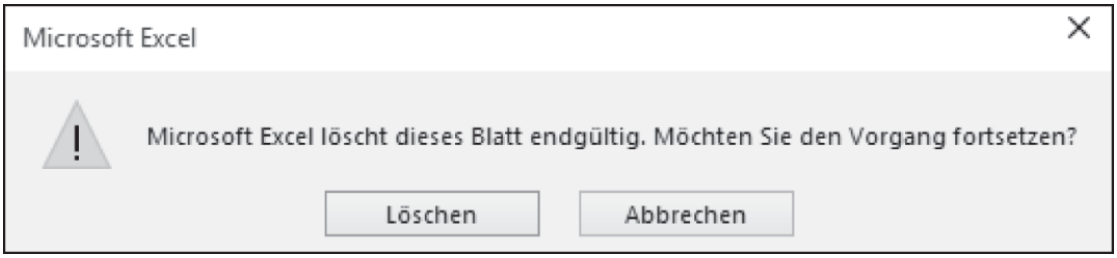


Abbildung 5.1: Entfernen Sie im Makro die Anweisung `Application.DisplayAlerts = False`, um sicherzustellen, dass Sie den Löschvorgang abbrechen können.

5. In Schritt 5 wird an den Anfang der Schleife zurückgesprungen und mit dem nächsten Tabellenblatt weitergemacht. Nachdem alle Tabellenblätter bearbeitet wurden, endet das Makro.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**

4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Wenn Sie in einem Makro `ThisWorkbook` anstatt `ActiveWorkbook` verwenden, können Sie das Makro nicht aus Ihrer persönlichen Makroarbeitsmappe ausführen lassen. Warum? Weil `ThisWorkbook` dann auf die persönliche Makroarbeitsmappe verweisen würde, und nicht auf die Arbeitsmappe, die mit dem Makro bearbeitet werden soll.

Alle Tabellenblätter bis auf das aktive ausblenden

Vielleicht wollen Sie nicht alle Tabellenblätter bis auf das aktive Blatt löschen, wie Sie es im vorigen Makro gesehen haben. Eine etwas sanftere Variante blendet diese Tabellenblätter einfach aus. Excel erlaubt Ihnen nicht, alle Tabellenblätter auszublenden; eines muss immer sichtbar sein. Sie können jedoch ohne Probleme alle Tabellenblätter bis auf das aktive ausblenden.

So funktioniert es

Dieses Makro durchläuft ebenfalls die Auflistung `Worksheets`, die alle Tabellenblätter der Arbeitsmappe enthält, und vergleicht deren Namen mit dem des aktiven Tabellenblatts. Bei jedem Schleifendurchlauf wird ein Tabellenblatt ausgeblendet, es sei denn, es ist das aktive Tabellenblatt.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim ws As Worksheet

'Schritt 2: Alle Tabellenblätter in einer ' Schleife durchlaufen
 For Each ws In ThisWorkbook.Worksheets

'Schritt 3: Prüfen, ob aktuelles Tabellenblatt aktiv ist  If ws.Name &lt;&gt;
ThisWorkbook.ActiveSheet.Name Then
```

```
'Schritt 4: Tabellenblatt ausblenden
 ws.Visible = xlSheetHidden

 End If

'Schritt 5: Mit nächstem Tabellenblatt weitermachen
 Next ws

End Sub
```

1. In diesem Schritt wird die Objektvariable `ws` des Typs `Worksheet` deklariert. Hierdurch wird ein Speicherbereich für jedes Tabellenblatt angelegt, das in der Schleife bearbeitet wird.
2. In Schritt 2 leitet das Makro die Schleife ein und weist Excel an, alle Tabellenblätter in der Arbeitsmappe zu bearbeiten. Es gibt einen Unterschied zwischen `ThisWorkbook` und `ActiveWorkbook`. Das Objekt `ThisWorkbook` verweist auf die Arbeitsmappe, in der sich auch der Code befindet. Das Objekt `ActiveWorkbook` hingegen verweist auf die derzeit aktive Arbeitsmappe. Oft geben beide das gleiche Objekt zurück; wenn die Arbeitsmappe, in der sich der Code befindet, jedoch nicht die aktive Arbeitsmappe ist, verweisen sie auf unterschiedliche Objekte. In unserem Beispiel wollen Sie nicht das Risiko eingehen, versehentlich Tabellenblätter in anderen Arbeitsmappen auszublenden, und verwenden daher `ThisWorkbook`.
3. In diesem Schritt vergleicht das Makro den Namen des aktiven Tabellenblatts mit dem Namen des Tabellenblatts, das derzeit in der Schleife bearbeitet wird.
4. Wenn sich die Namen unterscheiden, wird das Tabellenblatt in Schritt 4 ausgeblendet.
5. In Schritt 5 wird an den Anfang der Schleife zurückgesprungen und mit dem nächsten Tabellenblatt weitergemacht. Nachdem alle Tabellenblätter bearbeitet wurden, endet das Makro.

Beachten Sie im Makro die Verwendung von `xlSheetHidden`. Weisen Sie der Eigenschaft `Visible` diese Konstante zu, wird das Tabellenblatt so ausgeblendet, wie Sie es aus Excel kennen, wenn Sie eine Registerkarte mit der rechten Maustaste anklicken und dann AUSBLENDEN wählen. In diesem Zustand kann ein Anwender das Blattregister mit der rechten Maustaste anklicken und EINBLENDEN wählen. Er sieht dann eine Liste aller ausgeblendeten Tabellenblätter, in der auch unser ausgeblendetes Tabellenblatt angezeigt wird. Sie können auch die Konstante `xlSheetVeryHidden` verwenden, um ein Tabellenblatt auszublenden. Ein so ausgeblendetes Tabellenblatt kann in der Excel-Benutzeroberfläche nicht wieder eingeblendet werden; dies ist nur noch mit VBA-Code möglich.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt+F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Alle Tabellenblätter einer Arbeitsmappe einblenden

Wenn Sie jemals eine größere Zahl von Tabellenblättern einer Arbeitsmappe einblenden mussten, wissen Sie, wie umständlich das ist. Sie müssen hierfür das Dialogfeld EINBLENDEN aus

Abbildung 5.2 verwenden und jedes Tabellenblatt einzeln einblenden.

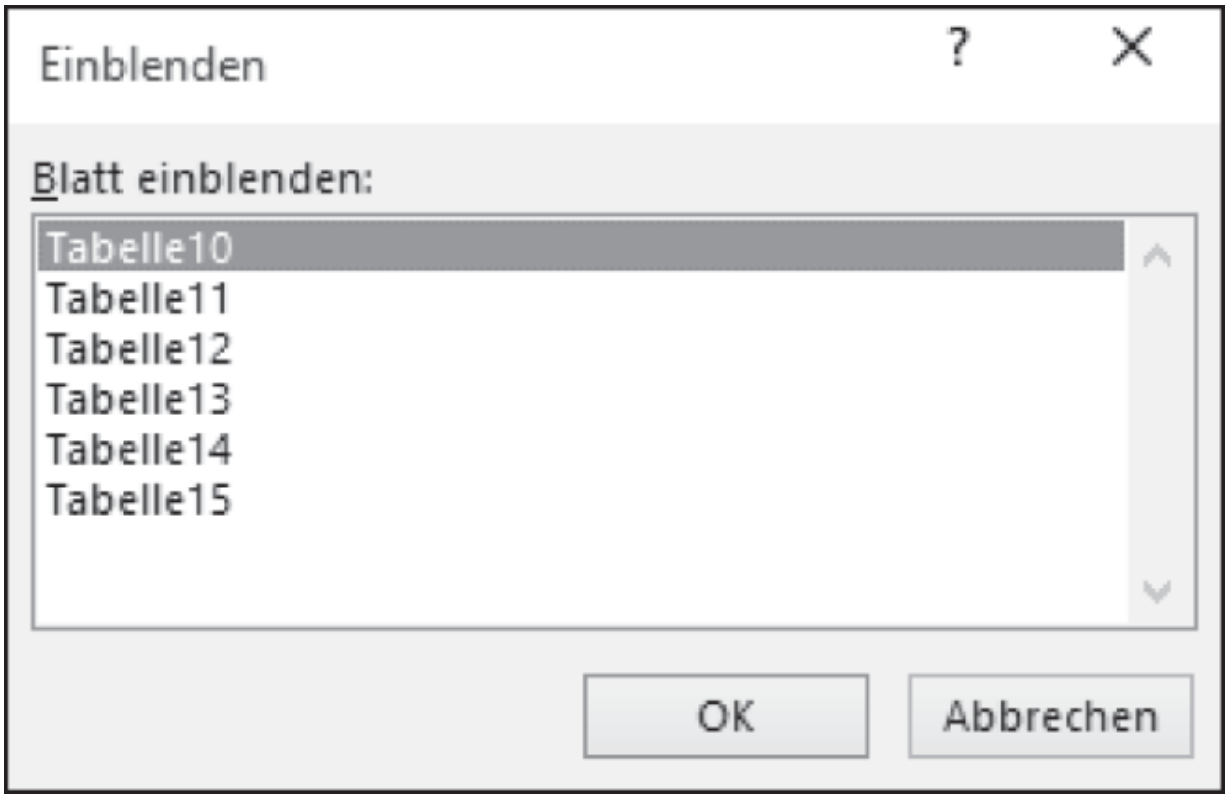


Abbildung 5.2: Ohne Makro sind Sie auf das Excel-Dialogfeld »Einblenden« angewiesen, um jedes Tabellenblatt einzeln wieder sichtbar zu machen.

Bei weniger Blättern mag dies noch gehen; was aber, wenn Sie zehn Tabellenblätter oder mehr einblenden wollen? Dieses Makro erledigt die Arbeit für Sie.

So funktioniert es

Dieses Makro durchläuft in einer Schleife alle Tabellenblätter und ändert deren Sichtbarkeit auf »eingeblendet«.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim ws As Worksheet

'Schritt 2: Alle Tabellenblätter in einer Schleife ' durchlaufen
 For Each ws In ActiveWorkbook.Worksheets
```

```
'Schritt 3: Mit dem nächsten Tabellenblatt weitermachen
 ws.Visible = xlSheetVisible
 Next ws

End Sub
```

1. In Schritt 1 wird eine Objektvariable mit Namen `ws` deklariert. Hierdurch wird ein Speicherbereich erzeugt, in dem während des Schleifendurchlaufs das jeweils aktuelle Tabellenblatt abgelegt wird.
2. In Schritt 2 startet die Schleife; Excel wird angewiesen, alle Tabellenblätter in der Sammlung Worksheets zu bearbeiten.
3. In Schritt 3 wird die Sichtbarkeit auf `xlSheetVisible` gesetzt und dann mit dem nächsten Tabellenblatt weitergemacht.

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Tabellenblätter verschieben

Wir alle müssen regelmäßig unsere Tabellenblätter verschieben, um sie in eine bestimmte, logische Reihenfolge zu bringen. Falls Sie dies regelmäßig tun müssen, kann Sie dieses Makro dabei unterstützen.

So funktioniert es

Um ein Tabellenblatt zu verschieben, verwenden Sie die Methode `Move`, die Ihnen für das `Sheets`- und das `ActiveSheet`-Objekt zur Verfügung steht. Bei der Methode `Move` verwenden Sie das Argument `After` oder `Before` oder beide zusammen, um die Zielposition anzugeben.

```
Sub Makro1()

'Aktives Tabellenblatt an das Ende verschieben
 ActiveSheet.Move After:=Worksheets(Worksheets.Count)

'Aktives Tabellenblatt an den Anfang verschieben
 ActiveSheet.Move Before:=Worksheets(1)

'Tabelle 1 vor Tabelle 12 verschieben
 Sheets("Tabelle1").Move Before:=Sheets("Tabelle12")

End Sub
```

Dieses Makro zeigt, wie Sie ein Tabellenblatt an drei verschiedene Positionen verschieben können:

- **Das aktive Tabellenblatt an das Ende verschieben:** Wenn Sie ein Tabellenblatt an das Ende des Blattregisters verschieben wollen, weisen Sie Excel an, dieses Blatt nach dem letzten Tabellenblatt einzufügen. Jedoch gibt es kein VBA-Konstrukt, um direkt auf das letzte Tabellenblatt zuzugreifen. Sie können jedoch die Anzahl der Tabellenblätter ermitteln und diese Zahl dann als Index für das Worksheets-Objekt verwenden. Mit `Worksheets(1)` beispielsweise zeigen Sie auf das erste Tabellenblatt, mit `Worksheets(3)` greifen Sie auf das dritte zu und so weiter. Um auf das letzte Tabellenblatt zuzugreifen, ersetzen Sie den Index durch die Eigenschaft `Worksheets.Count`. Diese Eigenschaft gibt die Anzahl der Tabellenblätter zurück, die dem Index des letzten Tabellenblatts entspricht. Mit `Worksheet(Worksheets.Count)` greifen Sie also auf das letzte Tabellenblatt zu.
- **Das aktive Tabellenblatt an den Anfang verschieben:** Ein Tabellenblatt an den Anfang der Arbeitsmappe zu verschieben, ist ganz einfach. Sie verwenden `Worksheets(1)`, um auf das erste Tabellenblatt zu zeigen und verschieben dann das aktuelle Tabellenblatt vor das erste.
- **Tabellenblatt x vor Tabellenblatt y verschieben:** Sie können ein Tabellenblatt einfach vor oder hinter ein anderes verschieben, indem Sie die Namen der Tabellenblätter verwenden. Im Beispiel des obigen Makros wird `Tabelle1` vor `Tabelle12` verschoben.

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur** Alt **+** F11 **drücken.**

2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Tabellenblätter nach Namen sortieren

Es kann vorkommen, dass Sie die Tabellenblätter alphabetisch nach deren Namen sortieren müssen (siehe Abbildung 5.3). Man sollte meinen, dass Excel eine Funktion enthält, die dies erledigt, doch leider ist das nicht der Fall. Wenn Sie Ihre Tabellenblätter nicht mehr weiterhin von Hand sortieren wollen, lassen Sie die Arbeit vom folgenden Makro erledigen.

Abbildung 5.3: Es ist häufig nützlich, wenn die Tabellenblätter alphabetisch sortiert sind.

So funktioniert es

Dieses Makro sieht komplizierter aus, als es ist. Tatsächlich sind die Aktivitäten dieses Makros eigentlich sogar ziemlich einfach. Das Makro durchläuft einfach alle Tabellenblätter der Arbeitsmappe und vergleicht den Namen des aktuellen Tabellenblatts mit dem des vorigen. Wenn der Name des vorigen Tabellenblatts (alphabetisch) größer ist als der des aktuellen, verschiebt das Makro das aktuelle Tabellenblatt vor das vorige. Nachdem die Schleife beendet ist, haben Sie eine sortierte Arbeitsmappe.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim CurrentSheetIndex As Integer
 Dim PrevSheetIndex As Integer

'Schritt 2: Anzahl der Schleifendurchläufe festlegen und ' Schleife starten
 For CurrentSheetIndex = 1 To Sheets.Count
 For PrevSheetIndex = 1 To CurrentSheetIndex - 1
```

```
'Schritt 3: Name des aktuellen Tabellenblatts mit dem des ' vorigen
vergleichen
 If UCase(Sheets(PrevSheetIndex).Name)&gt; _
UCase(Sheets(CurrentSheetIndex).Name) Then

'Schritt 4: Aktuelles Tabellenblatt vor das vorige ' verschieben
 Sheets(CurrentSheetIndex).Move _  Before:=Sheets(PrevSheetIndex)
 End If

'Schritt 5: Nächster Schleifendurchlauf
 Next PrevSheetIndex
 Next CurrentSheetIndex

End Sub
```

Beachten Sie, dass bei diesem Verfahren eine textbasierte Sortierung durchgeführt wird. Sie erhalten nicht das Ergebnis, dass Sie möglicherweise erwarten, wenn Sie Tabellenblätter sortieren, in deren Namen Zahlen vorkommen. Alphabetisch wird bei der textbasierten Sortierung *Blatt10* vor *Blatt2* einsortiert, da im Alphabet die 1 vor der 2 steht. Excel unterstützt für unser Beispiel keine Sortierung, bei der der Wert der Zahlen berücksichtigt wird.

1. In Schritt 1 werden zwei Variablen des Typs `Integer` deklariert. Die Variable `CurrentSheetIndex` enthält die Indexnummer des Tabellenblatts des aktuellen Schleifendurchlaufs und `PrevSheetIndex` die des vorigen.
2. In Schritt 2 wird die Anzahl der Schleifendurchläufe festgelegt und die beiden Variablen aus Schritt 1 werden initialisiert. Beachten Sie, dass die Anzahl der Durchläufe für `PrevSheetIndex` um 1 kleiner ist als `CurrentSheetIndex`. Nachdem die Anzahl festgelegt ist, wird die Schleife gestartet.
3. In Schritt 3 überprüfen Sie, ob der Name des vorigen Tabellenblatts größer ist als der des aktuellen Tabellenblatts.

   Beachten Sie die Verwendung der Funktion `UCase`, mit der beide Namen für den Vergleich in Großbuchstaben umgewandelt werden. Hierdurch werden Fehler bei der

Sortierung vermieden, die sich aus unterschiedlicher Groß-/Kleinschreibung ergeben könnten.

4. Der Code bei Schritt 4 wird nur dann ausgeführt, wenn der Name des vorigen Tabellenblatts größer ist als der des aktuellen. In diesem Schritt verwenden Sie die Methode `Move`, um das aktuelle Tabellenblatt vor das vorige zu verschieben.
5. In Schritt 5 wird der nächste Schleifendurchlauf gestartet. Jede Iteration der Schleife erhöht den Wert beider Variablen jeweils um 1, bis das letzte Tabellenblatt erreicht wurde. Nachdem alle Schleifendurchläufe gemacht sind, endet das Makro.

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE.

Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Tabellenblätter nach Farben gruppieren

Viele von uns weisen den Registerkarten unterschiedliche Farben zu. Sie können eine Registerkarte im Blattregister mit der rechten Maustaste anklicken und den Befehl REGISTERFARBE wählen (siehe Abbildung 5.4), um eine Farbe für die Registerkarte festzulegen.

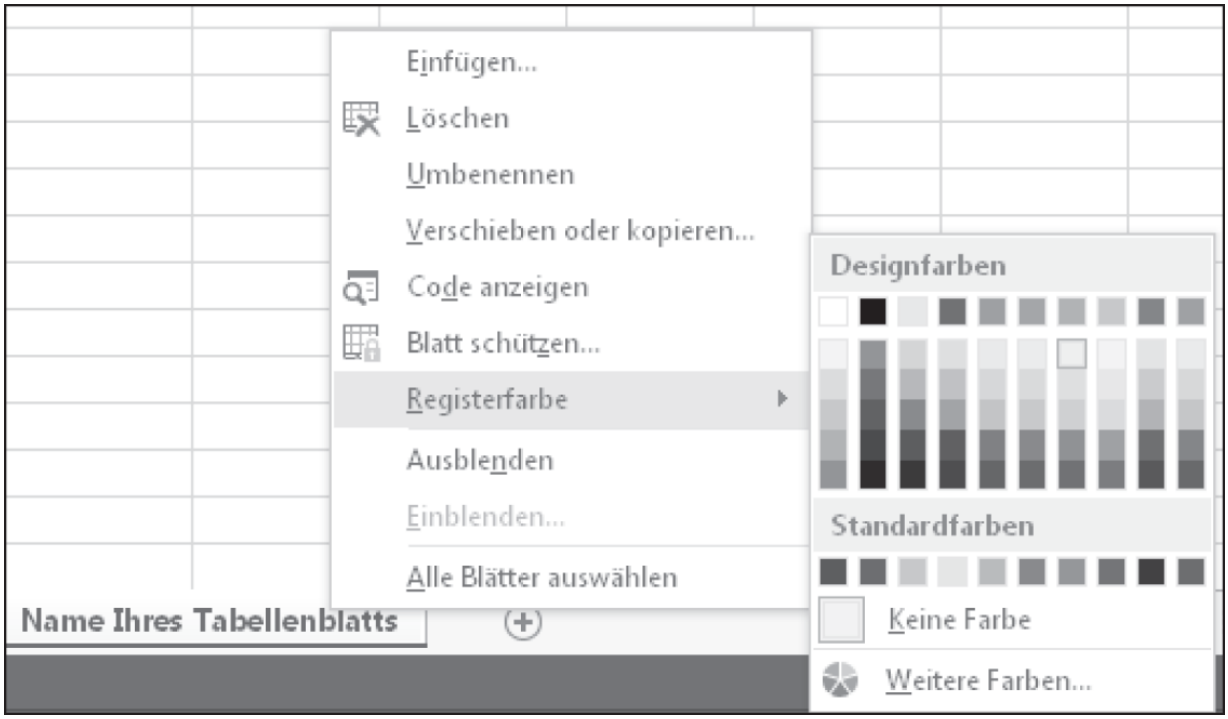


Abbildung 5.4: Sie können im Blattregister ein Tabellenblatt mit der rechten Maustaste anklicken, um die Registerfarbe festzulegen.

Sie können beispielsweise für Registerkarten die gleiche Farbe verwenden, wenn die Daten auf den betreffenden Tabellenblättern inhaltlich zusammengehören. Wenn eine Arbeitsmappe

zahlreiche farbige Registerkarten enthält, macht es Sinn, diese zu gruppieren, da Sie so die Navigation vereinfachen.

Dieses Makro gruppiert die Tabellenblätter anhand der Farbe ihrer Registerkarten.

So funktioniert es

Vielleicht vermuten Sie, dass es unmöglich ist, die Tabellenblätter nach Farben zu sortieren oder zu gruppieren. Excel stellt diese Möglichkeit jedoch zur Verfügung, da jeder Farbe ein Index zugeordnet ist. Ein helles Gelb könnte beispielsweise die Indexnummer 36 und ein Kastanienbraun den Index 42 besitzen.

Das Makro durchläuft alle Tabellenblätter der Arbeitsmappe und vergleicht den Farbindex des aktuellen Tabellenblatts mit dem des vorigen.

Wenn der Farbindex des vorigen Tabellenblatts mit dem des aktuellen identisch ist, verschiebt das Makro das aktuelle vor das vorige. Nachdem die Schleife beendet ist, sind alle Tabellenblätter nach der Farbe ihrer Registerkarte gruppiert.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim CurrentSheetIndex As Integer
 Dim PrevSheetIndex As Integer

'Schritt 2: Legen Sie die Anzahl der Durchläufe fest ' und starten Sie die
Schleife
 For CurrentSheetIndex = 1 To Sheets.Count
 For PrevSheetIndex = 1 To CurrentSheetIndex - 1

'Schritt 3: Aktuelles Tabellenblatt mit ' vorigem vergleichen
 If Sheets(PrevSheetIndex).Tab.ColorIndex = _
Sheets(CurrentSheetIndex).Tab.ColorIndex Then

'Schritt 4: Aktuelles Tabellenblatt vor das ' vorige verschieben
 Sheets(PrevSheetIndex).Move _  Before:=Sheets(CurrentSheetIndex)
 End If

'Schritt 5: Nächster Schleifendurchlauf
 Next PrevSheetIndex
 Next CurrentSheetIndex
```

```
End Sub
```

1. In Schritt 1 werden zwei Variablen des Typs `Integer` deklariert. `CurrentSheetIndex` enthält die Indexnummer des Tabellenblatts des aktuellen Schleifendurchlaufs und `PrevSheetIndex` die des vorigen.
2. In Schritt 2 wird die Anzahl der Schleifendurchläufe festgelegt und die beiden Variablen aus Schritt 1 werden initialisiert. Beachten Sie, dass die Anzahl der Durchläufe für `PrevSheetIndex` um 1 kleiner ist als `CurrentSheetIndex`. Nachdem die Anzahl festgelegt ist, wird die Schleife gestartet.
3. In Schritt 3 überprüft das Makro, ob der Farbindex des vorigen Tabellenblatts mit dem des aktuellen übereinstimmt. Hierzu wird die Eigenschaft `Tab.ColorIndex` verwendet.
4. Der Code bei Schritt 4 wird nur dann ausgeführt, wenn der Farbindex des vorigen Tabellenblatts mit dem des aktuellen identisch ist. In diesem Schritt verwenden Sie die Methode `Move`, um das aktuelle Tabellenblatt vor das vorige zu verschieben.
5. In Schritt 5 wird der nächste Schleifendurchlauf gestartet. Jede Iteration der Schleife erhöht den Wert beider Variablen jeweils um 1, bis das letzte Tabellenblatt erreicht wurde. Nachdem alle Schleifendurchläufe gemacht sind, endet das Makro.

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**

2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Ein Tabellenblatt in eine neue Arbeitsmappe kopieren

Sie können in Excel ein komplettes Tabellenblatt manuell in eine neue Arbeitsmappe kopieren, indem Sie die Registerkarte des Blatts mit der rechten Maustaste anklicken und den Befehl KOPIEREN ODER VERSCHIEBEN verwenden. Falls Sie jedoch versuchen, diese Aktion mit dem Makrorekorder aufzuzeichnen, werden Sie feststellen, dass der Code hierfür nicht richtig erstellt wird. Wenn Sie ein komplettes Tabellenblatt codegesteuert in eine brandneue Arbeitsmappe kopieren wollen, finden Sie in diesem Makro alles, was Sie brauchen.

So funktioniert es

In diesem Makro wird das aktive Tabellenblatt zuerst kopiert. Anschließend verwenden Sie den Parameter `Before`, um die Kopie an eine neue Arbeitsmappe zu senden, die gleichzeitig erstellt wird. Das kopierte Tabellenblatt wird als erstes Tabellenblatt in die neue Arbeitsmappe eingefügt.

Es ist wichtig, dass Sie hier das Objekt `ThisWorkbook` verwenden. Nur so wird das aktive Tabellenblatt aus der Arbeitsmappe kopiert, die den Code enthält, und nicht aus der neu erstellten Arbeitsmappe.

```
Sub Makro1()

'Schritt 1: Tabellenblatt kopieren und an neue ' Arbeitsmappe senden
 ThisWorkbook.ActiveSheet.Copy _  Before:=Workbooks.Add.Worksheets(1)

End Sub
```

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Eine neue Arbeitsmappe für jedes Tabellenblatt erstellen

Viele Excel-Analysten müssen eine Arbeitsmappe in mehrere Mappen aufteilen, wobei sich nach diesem Vorgang jedes Tabellenblatt in einer eigenen Arbeitsmappe befinden soll. Sie

können sich vorstellen, dass dies eine zeitraubende Aufgabe ist, wenn Sie das von Hand erledigen möchten. Das folgende Makro hilft hier und automatisiert diese Aufgabe.

So funktioniert es

In diesem Makro durchlaufen Sie in einer Schleife alle Tabellenblätter, kopieren sie und senden dann jeweils eine Kopie an eine neue Arbeitsmappe, die gleichzeitig erstellt wird. Beachten Sie in diesem Makro, dass die neu erstellten Arbeitsmappen im gleichen Ordner erstellt werden, in dem sich auch die Quellarbeitsmappe befindet. Als Dateiname wird der Name des kopierten Tabellenblatts verwendet (`wb.SaveAs ThisWorkbook.Path & "\" & ws.Name`).

```
Sub Makro1()

'Schritt 1: Alle Variablen deklarieren
 Dim ws As Worksheet
 Dim wb As Workbook

'Schritt 2: Schleife über alle Tabellenblätter
 For Each ws In ThisWorkbook.Worksheets

'Schritt 3: Neue Arbeitsmappe erstellen und speichern
 Set wb = Workbooks.Add
 wb.SaveAs ThisWorkbook.Path & "\" & ws.Name

'Schritt 4: Das aktuelle Tabellenblatt in die ' neue Arbeitsmappe kopieren
 ws.Copy Before:=wb.Worksheets(1)
 wb.Close SaveChanges:=True

'Schritt 5: In der Schleife nächstes Tabellenblatt ' bearbeiten
 Next ws

End Sub
```

Nicht alle Tabellenblattnamen können in gültige Dateinamen übersetzt werden.

In Windows gelten bestimmte Regeln für Dateinamen, was zur Folge hat, dass Sie bestimmte Zeichen nicht in Dateinamen verwenden können. In Dateinamen dürfen die

folgenden Zeichen nicht verwendet werden: Backslash (\), Schrägstrich (/), Sternchen (*), Fragezeichen (?), das Pipe-Symbol (|), doppelte Anführungszeichen ("), das Größer- (>) und das Kleinerzeichen (<).

Das Problem ist, dass Sie einige der in Dateinamen nicht erlaubten Zeichen in den Namen der Tabellenblätter verwenden können; hierzu zählen das doppelte Anführungszeichen, das Pipe-Symbol sowie das Größer- und das Kleinerzeichen.

Wenn Sie also dieses Makro ausführen, kann die Benennung der neu erstellten Dateien entsprechend dem Blattnamen zu einem Fehler führen. Beispielsweise erzeugt das Makro einen Fehler, wenn Sie beispielsweise aus dem gültigen Tabellenblattnamen *Mai|Umsatz* einen Dateinamen machen wollen (wegen des Pipe-Symbols).

Lange Rede, kurzer Sinn: Vermeiden Sie es, die oben erwähnten Zeichen in den Namen Ihrer Tabellenblätter zu verwenden!

1. In Schritt 1 werden zwei Objektvariablen deklariert. Die Variable `ws` erzeugt einen Speicherbereich, in dem ein Verweis auf das derzeit in der Schleife bearbeitete Tabellenblatt abgelegt wird. In der Variable `wb` wird ein Verweis auf die jeweils neu erstellte Arbeitsmappe abgelegt.
2. In Schritt 2 beginnt das Makro den Schleifendurchlauf durch alle Tabellenblätter. Die Verwendung von `ThisWorkbook` stellt sicher, dass das aktuelle Tabellenblatt aus der Arbeitsmappe kopiert wird, die den Code enthält, und nicht aus der neuen Arbeitsmappe, die Sie erstellen.
3. In Schritt 3 erstellen Sie die neue Arbeitsmappe und speichern sie. Die neue Arbeitsmappe wird im gleichen Ordner gespeichert wie die Quellarbeitsmappe (`ThisWorkbook`). Als Dateiname wird der Name des derzeit aktiven Tabellenblatts verwendet.

4. In Schritt 4 wird das aktive Tabellenblatt kopiert und mittels des Parameters `Before` als erste Registerkarte in die neue Arbeitsmappe eingefügt.
5. In Schritt 5 wird das nächste Tabellenblatt angefordert und die Schleife beginnt von vorne. Das Makro wird beendet, wenn das letzte Tabellenblatt bearbeitet wurde.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Bestimmte Tabellenblätter drucken

Wenn Sie in Excel mehrere Tabellenblätter auf einmal drucken wollen, müssen Sie die Strg-Taste auf der Tastatur gedrückt halten, die Tabellenblätter auswählen und abschließend den Befehl DRUCKEN wählen. Falls Sie diesen Vorgang öfter wiederholen müssen, können Sie auch dieses kleine Makro verwenden.

So funktioniert es

Dieses Makro ist ganz einfach. Alles, was Sie machen müssen, ist, ein Array mit den zu druckenden Tabellenblättern zu erstellen und dieses dann an die Methode `PrintOut` zu übergeben, die sich

um das Drucken kümmert. Alle so festgelegten Tabellenblätter werden auf einen Rutsch gedruckt.

```
Sub Makro1()
'Bestimmte Tabellenblätter drucken
 ActiveWorkbook.Sheets( _
 Array("Sheet1", "Sheet3", "Sheet5")).PrintOut _
  Copies:=1
End Sub
```

Wollen Sie alle Tabellenblätter einer Arbeitsmappe drucken? Dieses Makro ist noch einfacher:

```
Sub Makro1()
'Alle Tabellenblätter drucken
 ActiveWorkbook.Worksheets.PrintOut Copies:=1
End Sub
```

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Alle Tabellenblätter schützen

Bevor Sie eine Arbeitsmappe weitergeben, ist es sinnvoll, alle Tabellenblätter mit einem Blattschutz zu versehen. Wie Sie jedoch in Abbildung 5.5 sehen, deaktiviert Excel den Befehl BLATT SCHüTZEN, wenn Sie versuchen, mehrere Tabellenblätter auf einmal zu schützen. Sie sind daher gezwungen, die Tabellenblätter einzeln zu schützen.

Abbildung 5.5: Der Befehl »Blatt schützen« ist deaktiviert, wenn Sie mehrere Tabellenblätter ausgewählt haben.

Sie können dieses Makro verwenden, um alle Tabellenblätter in einem Schritt zu schützen.

So funktioniert es

In diesem Makro durchlaufen Sie die Sammlung der Tabellenblätter und versehen jedes Blatt einzeln mit einem Kennwortschutz. Mit dem Argument `Password` legen Sie das Kennwort fest, mit dem der Schutz wieder aufgehoben werden kann. Das Argument `Password` ist komplett optional. Wenn Sie es

weglassen, ist das Tabellenblatt weiterhin geschützt; Sie brauchen dann kein Kennwort einzugeben, um den Schutz aufzuheben. Bitte denken Sie daran, dass Excel bei Kennwörtern zwischen Groß- und Kleinschreibung unterscheidet. Achten Sie daher auf die richtige Schreibung der Kennwörter.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim ws As Worksheet

'Schritt 2: Schleife über alle Tabellenblätter starten
 For Each ws In ActiveWorkbook.Worksheets

'Schritt 3: Schützen und mit nächstem Tabellenblatt
' weitermachen

 ws.Protect Password:="ROT"
 Next ws

End Sub
```

1. In Schritt 1 wird die Objektvariable `ws` deklariert. In dieser Variablen wird ein Verweis auf das jeweils in der Schleife bearbeitete Tabellenblatt abgelegt.
2. In Schritt 2 wird die Schleife gestartet und Excel angewiesen, alle Tabellenblätter der Arbeitsmappe zu bearbeiten.
3. In Schritt 3 wird das Tabellenblatt mit dem angegebenen Kennwort geschützt, dann geht es mit dem nächsten Tabellenblatt weiter.

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Den Blattschutz aller Tabellenblätter aufheben

Möglicherweise müssen Sie regelmäßig den Blattschutz mehrerer Tabellenblätter aufheben. Wie Abbildung 5.6 zeigt, deaktiviert Excel den Befehl BLATTSCHUTZ AUFHEBEN, wenn Sie mehr als ein Tabellenblatt auswählen. Sie müssen daher den Blattschutz für jedes Tabellenblatt einzeln entfernen.

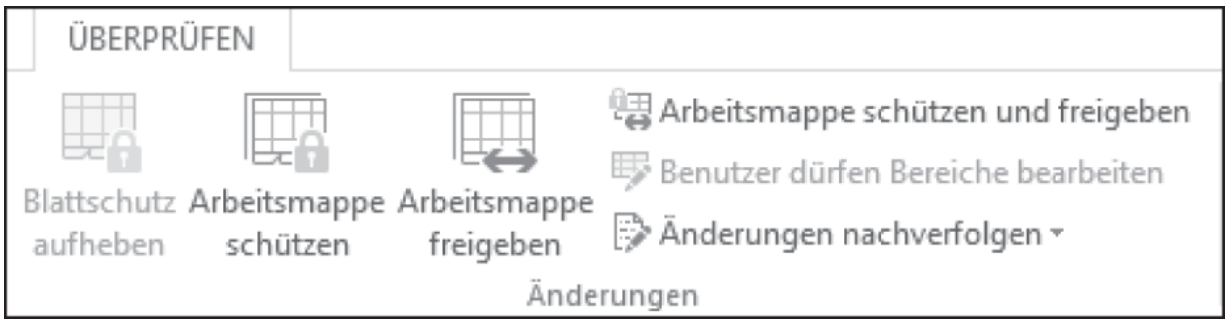


Abbildung 5.6: Der Befehl »Blattschutz aufheben« ist deaktiviert, wenn Sie mehrere Tabellenblätter ausgewählt haben.

Sie können jedoch dieses Makro verwenden, um den Blattschutz für alle Tabellenblätter in einem Rutsch aufzuheben.

So funktioniert es

Dieses Makro durchläuft die Sammlung aller Tabellenblätter und verwendet das Argument `Password` der Methode `Unprotect`, um den Blattschutz aufzuheben:

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim ws As Worksheet

'Schritt 2: In einer Schleife alle Tabellenblätter ' durchlaufen
 For Each ws In ThisWorkbook.Worksheets

'Schritt 3: Blattschutz aufheben und mit nächstem ' Tabellenblatt
weitermachen
 ws.Unprotect Password:="ROT"
 Next ws

End Sub
```

1. In Schritt 1 wird die Objektvariable `ws` deklariert. In ihr wird ein Verweis auf das jeweils in der Schleife bearbeitete Tabellenblatt abgelegt.
2. In Schritt 2 wird die Schleife gestartet und Excel angewiesen, alle Tabellenblätter der Arbeitsmappe zu bearbeiten.
3. In Schritt 3 wird der Blattschutz des aktuellen Tabellenblatts aufgehoben, hierfür wird das erforderliche Kennwort

angegeben, dann macht das Makro mit dem nächsten Tabellenblatt weiter.

Das Beispiel geht davon aus, dass alle Tabellenblätter mit dem gleichen Kennwort geschützt wurden. Sollte dies nicht der Fall sein, müssen Sie für jedes Tabellenblatt einzeln mit dem betreffenden Kennwort den Blattschutz aufheben:

```
Sub Macro1()
 Sheets("Tabelle1").UnProtect Password:="ROT"
 Sheets("Tabelle2").UnProtect Password:="BLAU"
 Sheets("Tabelle3").UnProtect Password:="GELB"
 Sheets("Tabelle4").UnProtect Password:="CYAN"
End Sub
```

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO

AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Ein Inhaltsverzeichnis Ihrer Arbeitsmappe erstellen

Neben dem Sortieren von Arbeitsblättern stellt das Erstellen eines Inhaltsverzeichnisses einer Arbeitsmappe die Aufgabe dar, für die ich am häufigsten nach einer Makrolösung gefragt werde. Der Grund hierfür ist sicherlich die Tatsache, dass man öfter mit Arbeitsmappen arbeitet, die sehr viele Registerkarten für Tabellenblätter haben. Hierdurch sieht nicht alle Registerkarten auf einen Blick, was die Navigation in der Mappe erschwert. Ein Inhaltsverzeichnis wie das in Abbildung 5.7 kann die Navigation in der Arbeitsmappe unterstützen.

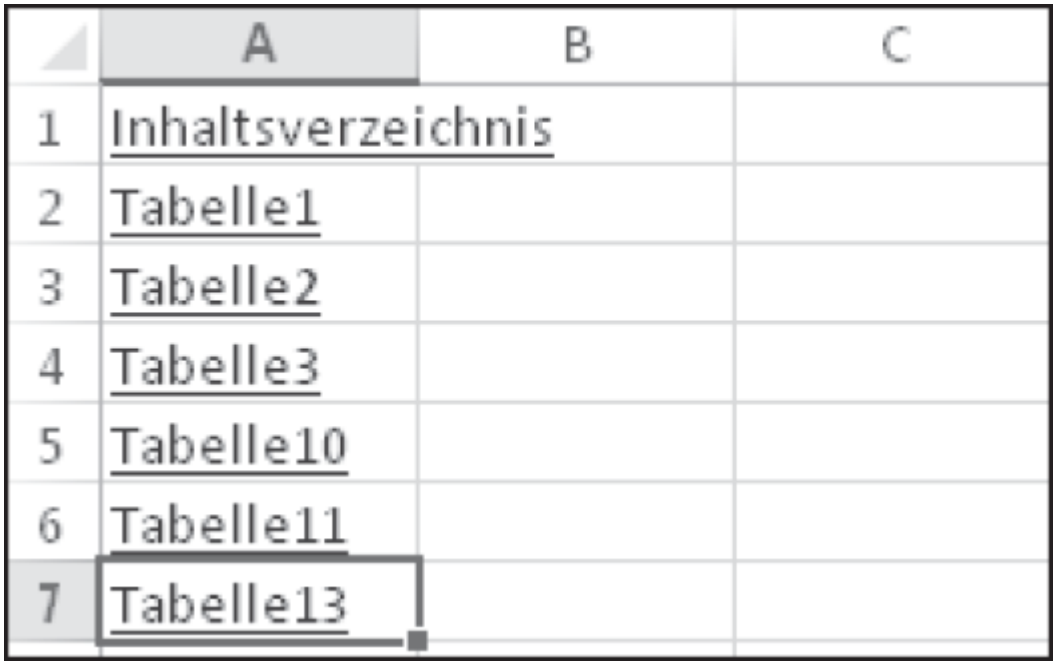


Abbildung 5.7: Mit einem Inhaltsverzeichnis lässt sich sehr schnell auch in einer umfangreichen Arbeitsmappe navigieren.

Das folgende Makro erstellt nicht nur eine Liste mit den Namen der Tabellenblätter in der Arbeitsmappe, sondern es fügt auch Hyperlinks ein, mit denen Sie durch einen einfachen Klick zu einem bestimmten Tabellenblatt springen können.

So funktioniert es

Wenn Sie sich das Makro dieses Abschnitts anschauen, könnte es Sie ganz schön einschüchtern. Wenn Sie aber einen Schritt zurücktreten und sich die einfachen Aktionen anschauen, die das Makro ausführt, wirkt es schnell schon sehr viel weniger beängstigend. Das Makro macht Folgendes:

- ✔ Es löscht ein eventuell vorhandenes Tabellenblatt mit dem alten Inhaltsverzeichnis.
- ✔ Es erstellt ein neues Tabellenblatt für das Inhaltsverzeichnis.
- ✔ Es ermittelt die Namen der einzelnen Tabellenblätter und fügt sie in das Inhaltsverzeichnis ein.
- ✔ Es ergänzt jeden Eintrag im Inhaltsverzeichnis um einen Hyperlink.

Hört sich doch recht übersichtlich an, oder? Hier ist der Code:

```
Sub Makro1()

'Schritt 1: Variablen deklarieren
 Dim i As Long

'Schritt 2: Voriges Inhaltsverzeichnis löschen, ' falls vorhanden
 On Error Resume Next
 Application.DisplayAlerts = False
 Sheets("Inhaltsverzeichnis").Delete
 Application.DisplayAlerts = True
 On Error GoTo 0

'Schritt 3: Neues Tabellenblatt für Inhaltsverzeichnis ' als erstes Blatt
einfügen
 ThisWorkbook.Sheets.Add _  Before:=ThisWorkbook.Worksheets(1)
 ActiveSheet.Name = "Inhaltsverzeichnis"

'Schritt 4: Schleife mit i als Schleifenzähler starten
 For i = 1 To Sheets.Count
```

```
'Schritt 5: Nächste verfügbare Zeile auswählen
 ActiveSheet.Cells(i, 1).Select

'Schritt 6: Name des Tabellenblatts und Hyperlink ' (Anchor) einfügen
 ActiveSheet.Hyperlinks.Add _  Anchor:=ActiveSheet.Cells(i, 1), _
Address:="", _  SubAddress:="'" & Sheets(i).Name & "'!A1", _
TextToDisplay:=Sheets(i).Name

'Schritt 7: i inkrementieren und eventuell' weiterer Schleifendurchlauf
 Next i

End Sub
```

1. In Schritt 1 wird die Variable `i` des Typs `Integer` deklariert; diese dient als Schleifenzähler, während das Makro die einzelnen Tabellenblätter bearbeitet.

   Beachten Sie, dass in diesem Makro die Schleife, in der die verschiedenen Tabellenblätter bearbeitet werden, anders aufgebaut ist als in den bisherigen Beispielen dieses Kapitels. In den vorherigen Beispielen haben Sie eine Schleife über die Sammlung Worksheets programmiert und so die einzelnen Tabellenblätter ausgewählt. In diesem Beispiel verwenden Sie einen Schleifenzähler (die Variable `i`). Der Grund dafür ist, dass Sie nicht nur die einzelnen Tabellenblätter bearbeiten, sondern zusätzlich den Namen des jeweiligen Tabellenblatts in eine neue Zeile des Inhaltsverzeichnisses eingeben müssen. Die Idee dabei ist, dass Sie während der Erstellung des Inhaltsverzeichnisses die Zellmarkierung um eine Zeile nach unten bewegen müssen, damit jeder Eintrag in einer neuen Zeile steht.
2. In Schritt 2 wird versucht, ein bereits vorhandenes Tabellenblatt mit dem Namen *Inhaltsverzeichnis* zu löschen. Da dieses Tabellenblatt eventuell nicht vorhanden sein kann, müssen Sie in Schritt 2 zuerst die Anweisung `On Error Resume Next` verwenden. Hierdurch informieren Sie Excel, dass die Makroausführung auch bei einem Fehler fortgesetzt werden soll. Anschließend schalten Sie mit der Methode `DisplayAlerts` die Excel-Warnhinweise ab, wodurch Sie das

anschließende Löschen des Tabellenblatts nicht bestätigen müssen. Abschließend setzen Sie den Fehlerhandler zurück (`On Error GoTo 0`), damit alle weiteren Fehler wieder abgefangen werden.

3. In Schritt 3 fügen Sie in die Arbeitsmappe ein neues Tabellenblatt ein. Sie verwenden das Argument `Before`, damit es als erstes Tabellenblatt eingefügt wird. Das neue Tabellenblatt erhält den Namen *Inhaltsverzeichnis*. Wie bereits weiter vorn in diesem Kapitel erwähnt, wird ein neu eingefügtes Tabellenblatt automatisch aktiviert. Da das neue Tabellenblatt während der gesamten Prozedur den Fokus hat, beziehen sich im Code ab jetzt alle Verweise mit `ActiveSheet` auf das Tabellenblatt *Inhaltsverzeichnis*.
4. In Schritt 4 wird die Schleife gestartet. Der Schleifenzähler wird mit dem Wert 1 initialisiert und die Schleife soll enden, nachdem alle Tabellenblätter bearbeitet wurden. Anders als in den bisherigen Makros, in denen Sie die Sammlung `Worksheets` verwendet haben, um alle Tabellenblätter zu bearbeiten, verwenden Sie den Schleifenzähler `i` als Index für das `Sheets`-Objekt. Wenn die maximale Anzahl erreicht ist, wird das Makro beendet.
5. In Schritt 5 wird die richtige Zeile im Tabellenblatt *Inhaltsverzeichnis* ausgewählt: Wenn der Schleifenzähler den Wert 1 hat, wird die erste Zeile ausgewählt. Hat der Schleifenzähler den Wert 2, wird die zweite Zeile verwendet und so weiter.

   Um dies zu tun, verwenden Sie das Element `Cells`. Das Element `Cells` ist eine extrem nützliche Möglichkeit, mittels Code Bereiche auszuwählen. Sie brauchen hierbei nur die Nummer der Zeile und der Spalte als Argumente anzugeben. `Cells(1,1)` entspricht Zeile 1/Spalte 1, also Zelle A1. `Cells(5,3)` entspricht Zeile 5/Spalte 3, also Zelle C5. Die numerischen Parameter bei `Cells` sind besonders dann nützlich, wenn Sie eine Reihe von Spalten oder Zeilen

bearbeiten und hierfür einen inkrementierenden Index verwenden wollen (wie wir es hier gerade tun).

6. In Schritt 6 wird die Methode `Hyperlinks.Add` verwendet, um in die ausgewählte Zelle den Namen des Tabellenblatts als Hyperlink einzufügen. Die Methode `Hyperlinks.Add` erhält in diesem Schritt alle Parameter, die für die Erstellung dieses Hyperlinks nötig sind.
7. Im letzten Schritt wird der Schleifenzähler um den Wert 1 erhöht und dann am Anfang der Schleife weitergemacht. Wenn der Wert des Schleifenzählers `i` mit der Anzahl der in der Arbeitsmappe enthaltenen Tabellenblätter übereinstimmt, wird das Makro beendet.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Mit einem Doppelklick in einem Tabellenblatt rein- und rauszoomen

Manche Kalkulationstabellen sind riesig. Daher sind Sie manchmal gezwungen, Zellen und Schriftgröße zu verkleinern, um so einen angemessenen Bereich der Tabelle auf Ihrem

Bildschirm sehen zu können. Falls Sie immer wieder die Zoomfunktion von Excel verwenden, um abwechselnd den Inhalt der Zellen und eine Übersicht der Tabelle zu sehen, können Sie auch das folgende Makro verwenden, in dem eine Art Autozoom mit einem Doppelklick aufgerufen wird.

So funktioniert es

Wenn das Makro aktiv ist, können Sie eine beliebige Zelle des Tabellenblatts anklicken, um den Zoomfaktor auf 200 % zu setzen. Der nächste Doppelklick stellt den Zoomfaktor auf 100 % ein. Sie können die Werte und die Komplexität des Makros natürlich so anpassen, dass sie genau zu Ihren Anforderungen passen.

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target _  As Range, Cancel As
Boolean)

'Aktuellen Zoomfaktor überprüfen 'Zoomfaktor auf 100% setzen, wenn er nicht
auf 100% steht 'Zoomfaktor auf 200% setzen, wenn er derzeit 100% beträgt
 If ActiveWindow.Zoom &lt;&gt; 100 Then
 ActiveWindow.Zoom = 100
 Else
 ActiveWindow.Zoom = 200
 End If

End Sub
```

Beachten Sie, dass der Nebeneffekt eines Doppelklicks auf eine Zelle darin besteht, dass so der Bearbeitungsmodus aktiviert wird. Sie können den Bearbeitungsmodus durch Drücken von Esc auf Ihrer Tastatur beenden. Falls es Sie stört, dass Sie nach der Verwendung des Makros Esc drücken müssen, können Sie am Ende der Prozedur noch diese Anweisung ergänzen:

```
Application.SendKeys ("{ESC}")
```

Diese Anweisung simuliert das Drücken der Esc-Taste.

So verwenden Sie es

Um dieses Makro zu implementieren, müssen Sie im Codefenster eine Ereignisprozedur für das Ereignis `Worksheet_BeforeDoubleClick` einfügen. Da Sie das Makro diesem Ereignis zuordnen, wird es automatisch ausgeführt, wenn auf dem Tabellenblatt ein Doppelklick ausgeführt wird.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie das Tabellenblatt an, das Sie mit dem Code verknüpfen wollen.**
4. **Wählen Sie in der Drop-down-Liste EREIGNISSE (siehe Abbildung 5.8) das Ereignis BEFOREDOUBLECLICK aus.**
5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin und passen Sie gegebenenfalls den Zellbereich an.**

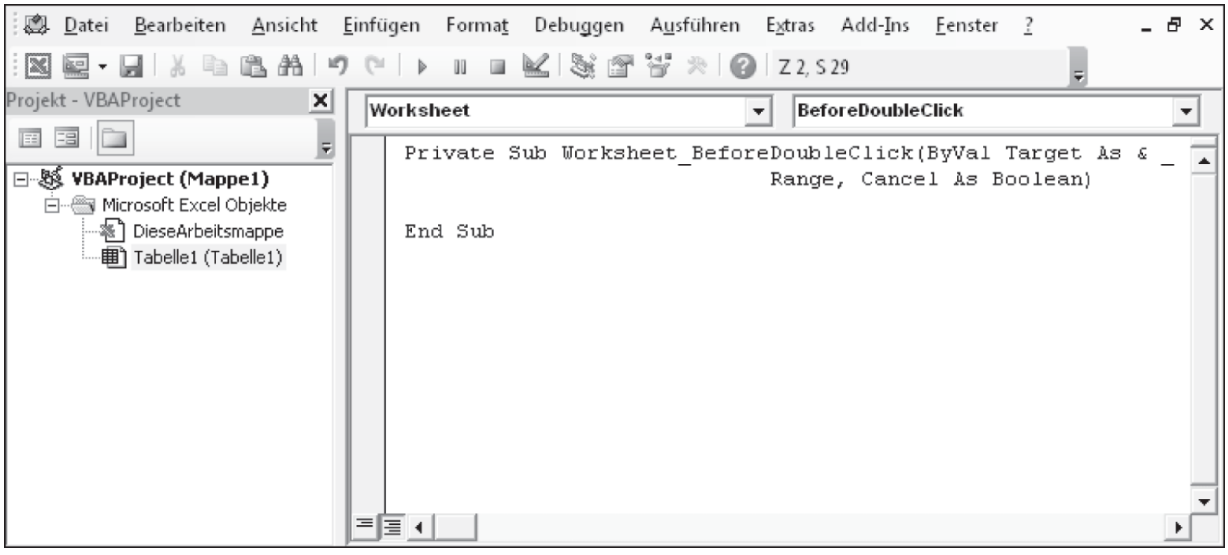


Abbildung 5.8: Tippen Sie den Code in das Tabellenblattereignis `BeforeDoubleClick` ein oder kopieren Sie den Code dorthin.

Die aktive Zeile und Spalte hervorheben

Wenn Sie sich eine größere Tabelle ansehen, wäre es hilfreich, wenn Excel automatisch die Zeile und Spalte hervorheben würde, in denen sich die Zellmarkierung befindet (siehe Abbildung 5.9). Ihre Augen erhalten so eine Führungslinie zu den anderen Werten in der gleichen Zeile und Spalte.

Das folgende Makro stellt Ihnen den Effekt aus Abbildung 5.9 mit einem einfachen Doppelklick zur Verfügung. Wenn das Makro aktiv ist, hebt Excel die Zeile und Spalte der Zelle hervor, in der sich die Zellmarkierung befindet. Das Bearbeiten und die Analyse umfangreicher Tabellen werden so einfacher.

	A	B	C	D	E	F
7						
8		Jan	Feb	Mrz	Apr	Mai
9	Produkt 1	74.083,84 €	41.353,24 €	37.032,13 €	77.941,36 €	35.220,88 €
10	Produkt 2	70.049,07 €	42.425,26 €	51.965,73 €	25.159,41 €	35.928,65 €
11	Produkt 3	13.513,06 €	98.468,34 €	18.818,30 €	27.000,91 €	11.372,94 €
12	Produkt 4	72.705,11 €	25.553,00 €	68.708,83 €	86.277,59 €	58.277,59 €
13	Produkt 5	35.636,76 €	81.466,83 €	83.445,02 €	51.796,66 €	58.971,24 €
14	Produkt 6	61.118,33 €	71.932,14 €	42.152,73 €	20.369,92 €	44.917,29 €
15	Produkt 7	42.302,98 €	19.756,75 €	78.250,07 €	32.396,48 €	1.862,81 €
16	Produkt 8	74.734,52 €	53.599,25 €	52.356,69 €	55.777,79 €	89.745,38 €
17	Produkt 9	29.764,26 €	31.476,35 €	92.660,67 €	76.510,44 €	93.957,18 €
18	Produkt 10	37.576,68 €	68.726,32 €	42.900,03 €	60.591,57 €	7.627,07 €
19	Produkt 11	98.304,20 €	19.808,88 €	56.833,70 €	62.310,98 €	54.039,42 €
20	Produkt 12	64.827,38 €	85.194,97 €	16.953,12 €	47.823,94 €	26.565,25 €
21						
22						

Abbildung 5.9: Dank der hervorgehobenen Zeile und Spalte ist es einfacher, die anderen Daten in der Zeile und Spalte zu sehen, in der sich die Zellmarkierung befindet.

So funktioniert es

Schauen Sie sich an, wie das Makro funktioniert:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target _  As Range, Cancel As
Boolean)

'Schritt 1: Variablen deklarieren
 Dim strRange As String

'Schritt 2: Zeichenkette mit dem Bereich zusammenbauen
 strRange = Target.Cells.Address & "," & _
Target.Cells.EntireColumn.Address & "," & _
Target.Cells.EntireRow.Address

'Schritt 3: Die Zeichenfolge mit dem Bereich an ein
' Range-Objekt übergeben

 Range(strRange).Select

End Sub
```

1. Zuerst deklarieren Sie die Variable `strRange` des Typs `String`. Sie erhalten so einen Speicherbereich, in dem Sie die Bereichszeichenfolge zusammenbauen können.
2. Eine Bereichszeichenfolge ist nichts anderes als die Adresse eines Bereichs. »A1« ist eine Bereichszeichenfolge, die auf die Zelle A1 verweist. »A1:G5« ist ebenfalls eine Bereichszeichenfolge: Sie umfasst die Zellen A1 bis G5. In Schritt 2 bauen Sie diese Zeichenfolge zusammen. Sie umfasst die Zelle, auf die doppelt geklickt wurde (im Code `Target` genannt), sowie die komplette aktive Zeile und die komplette aktive Spalte. Die Eigenschaft `Address` dieser drei Bereiche wird jeweils ermittelt und dann an die Variable `strRange` angefügt.
3. In Schritt 3 übergeben Sie die Variable `strRange` als Adresse an die `Range.Select`-Anweisung. Das ist die Codezeile, mit der letztendlich die Hervorhebung angezeigt wird.

So verwenden Sie es

Um dieses Makro zu implementieren, müssen Sie im Codefenster eine Ereignisprozedur für das Ereignis `Worksheet_BeforeDoubleClick` einfügen. Da Sie das Makro diesem

Ereignis zuordnen, wird es automatisch ausgeführt, wenn auf dem Tabellenblatt ein Doppelklick ausgeführt wird.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie das Tabellenblatt an, das Sie mit dem Code verknüpfen wollen.**
4. **Wählen Sie in der Drop-down-Lliste EREIGNISSE (siehe Abbildung 5.10) das Ereignis BEFOREDOUBLECLICK aus.**

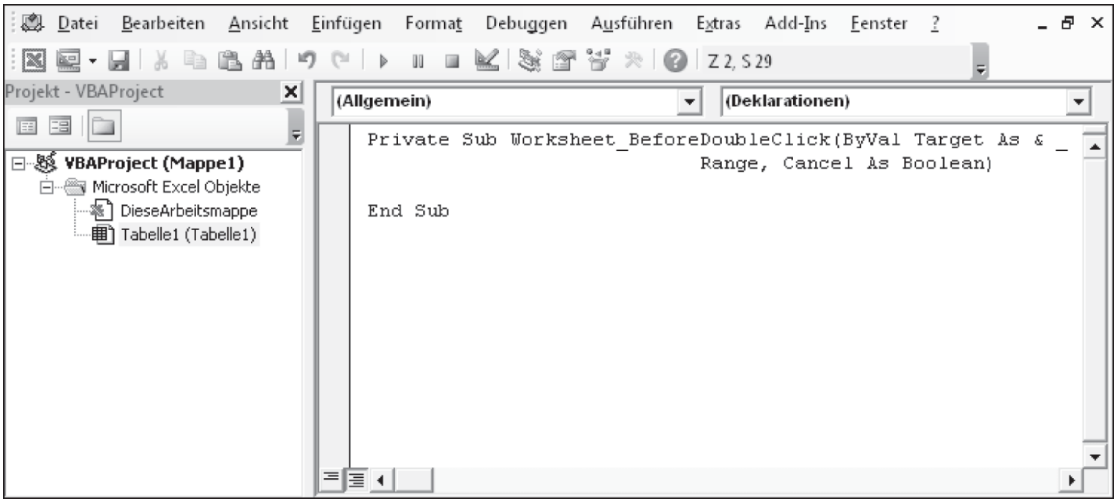


Abbildung 5.10: Geben Sie den Code in das Tabellenblattereignis BeforeDoubleClick ein oder kopieren Sie ihn dorthin.

5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin und passen Sie gegebenenfalls den Zellbereich an.**

Teil III

Datenbearbeitung mit einem Klick

IN DIESEM TEIL …

- ✔ Gehen Sie über die Erstellung einfacher Makros hinaus und schauen sich fortgeschrittene Techniken an, um mit VBA in Zellbereichen zu navigieren
- ✔ Lernen Sie, wie sich die Auswahl und die Bearbeitung von Zellen automatisieren lassen
- ✔ Untersuchen Sie, wie Sie mithilfe von Makros die Daten Ihrer Arbeitsmappen aufräumen und konvertieren können
- ✔ Entdecken Sie Techniken, um den Export von Daten zu automatisieren

Kapitel 6

Makros für Zellen und Zellbereiche

IN DIESEM KAPITEL

- Zellbereiche auswählen und formatieren
- Benannte Bereiche erstellen und auswählen
- Alle Zeilen und Spalten einblenden
- Leere Zeilen und Spalten einfügen oder löschen
- Den Scrollbereich einschränken
- Alle Formeln einer Arbeitsmappe auswählen und formatieren
- Die erste leere Zeile oder Spalte finden und auswählen

Eine der wichtigsten Aktionen, die Sie in Excel vornehmen, ist die Navigation in den Tabellenblättern. Während Sie Excel manuell nutzen, navigieren Sie zu den Bereichen, die Sie bearbeiten wollen, suchen nach der letzten Zeile, verschieben die Zellmarkierung in die letzte Spalte, blenden Bereiche aus und wieder ein und so weiter. All dies geschieht instinktiv als Teil der Arbeit in Excel.

Auch während Sie Ihre Arbeit mit VBA automatisieren, werden Sie feststellen, dass die Navigation im Tabellenblatt einen wichtigen Bestandteil des Automatisierungsprozesses darstellt. In vielen Fällen müssen Sie Excel-Bereiche dynamisch ansteuern und bearbeiten, wie Sie es von der manuellen Arbeit her kennen – nur dass Sie es diesmal mithilfe von VBA-Code tun. Dieses Kapitel stellt einige der am häufigsten benötigten Makros vor, um in einem Tabellenblatt zu navigieren und mit Bereichen zu arbeiten.

Einen Bereich auswählen und formatieren

Eines der grundlegendsten Dinge, die Sie in VBA machen müssen, ist das Auswählen einer Zelle oder eines Zellbereichs, um danach etwas mit der Auswahl tun zu können. Dieses einfache Makro markiert den Zellbereich D5:D16.

So funktioniert es

In diesem Makro legen Sie den Zellbereich, der ausgewählt werden soll, mit dem `Range`-Objekt fest:

```
Sub Makro1()

 Range("D5:D16").Select

End Sub
```

Nachdem Sie den Zellbereich ausgewählt haben, können Sie die anderen Eigenschaften des `Range`-Objekts verwenden, um die markierten Zellen zu bearbeiten. Erweitern Sie dieses Makro nun folgendermaßen: Der Zellbereich soll ein bestimmtes Zahlenformat bekommen, die Schrift sollfett dargestellt werden und der Bereich soll eine gelbe Hintergrundfarbe erhalten:

```
Sub Makro1()

 Range("D5:D16").Select
 Selection.NumberFormat = "#,##0"
 Selection.Font.Bold = True
 Selection.Interior.ColorIndex = 36

End Sub
```

Sie brauchen sich nicht alle Eigenschaften des `Range`-Objekts zu merken, um sie zu verwenden. Sie können einfach ein Makro aufzeichnen, die Formatierungen vornehmen und sich dann den von Excel erzeugten Code ansehen. Da Sie nun die korrekte Syntax vor Augen haben,

können Sie sie in Ihrem Code verwenden. Viele Excel-Programmierer nutzen diesen Weg, um VBA zu erlernen.

Beachten Sie, dass in dem obigen Beispiel `Selection` verwendet wird, um auf den markierten Bereich zuzugreifen. Um effizienteren Code zu schreiben, können Sie auch einfach direkt auf den Bereich zugreifen und dabei die `With-End-With`-Anweisung verwenden. Diese Anweisung informiert Excel darüber, dass alle innerhalb der `With`-Anweisung stehenden Befehle auf das Objekt angewendet werden sollen, auf das Sie verweisen. Beachten Sie außerdem, dass im folgenden Makro nicht zuerst der Zellbereich ausgewählt wird. Dies ist der entscheidende Punkt: Sie können in einem Makro einen Zellbereich bearbeiten, ohne ihn vorher auszuwählen.

```
Sub Makro1()

 With Range("D5:D16")
 .NumberFormat = "#,##0"
 .Font.Bold = True
 .Interior.ColorIndex = 36
 End With

End Sub
```

Alternativ können Sie auch einen Zellbereich markieren, indem Sie das Element `Cells` des `Range`-Objekts verwenden.

Das `Cells`-Element bietet ein extrem einfaches Verfahren, um Bereiche mittels Code zu markieren. Sie brauchen hierbei lediglich die Nummern von Spalte und Zeile anzugeben. `Cells(5,4)` entspricht Zeile 5, Spalte 4 (oder Zelle D5). `Cells(16,4)` entspricht Zeile 16, Spalte 4 (oder Zelle D16).

Um einen Zellbereich auszuwählen, übergeben Sie einfach zwei `Cells`-Elemente an das `Range`-Objekt. Das folgende Makro wählt ebenfalls den Zellbereich D5:D16 aus:

```
Sub Makro1()
 Range(Cells(5, 4), Cells(16, 4)).Select
End Sub
```

Hier folgt jetzt der überarbeitete Code zur Formatierung des Bereichs, diesmal aber mit dem Element `Cells`. Beachten Sie auch hier, dass im Makro der Zellbereich, der formatiert wird, nicht zuerst ausgewählt wird. Sie können wie gesagt einen Bereich bearbeiten, ohne ihn vorher auszuwählen.

```
Sub Makro1()

 With Range(Cells(5, 4), Cells(16, 4))
 .NumberFormat = "#,##0"
 .Font.Bold = True
 .Interior.ColorIndex = 36
 End With

End Sub
```

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen Sie es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Tippen Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Benannte Bereiche erstellen und auswählen

Eines der nützlichsten Features von Excel ist die Möglichkeit, Bereiche mit einem Namen zu versehen. Auf diese Zellbereiche können Sie dann im VBA-Code mithilfe dieses Namens zugreifen.

Hier sind die Schritte, mit denen Sie in der Excel-Benutzeroberfläche einem Zellbereich einen Namen zuweisen:

1. **Markieren Sie den Zellbereich, dem Sie einen Namen zuweisen wollen.**
2. **Wechseln Sie zur Registerkarte FORMELN und klicken Sie in der Gruppe DEFINIERTE NAMEN auf NAMEN DEFINIEREN (siehe Abbildung 6.1).**

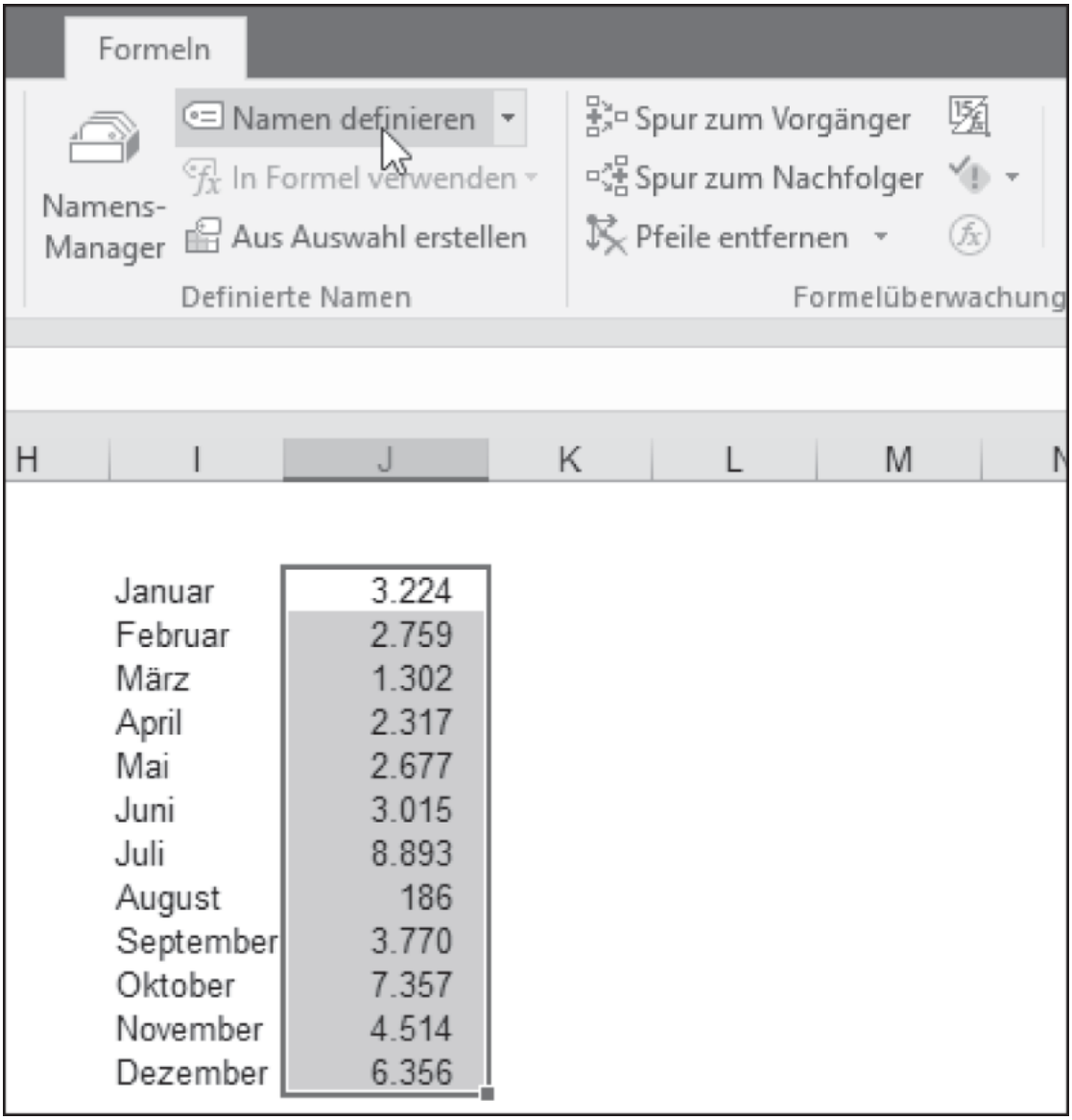


Abbildung 6.1: Klicken Sie auf den Befehl »Namen definieren«, um den ausgewählten Bereich mit einem Namen zu versehen.

3. **Geben Sie im Dialogfeld NEUER NAME einen aussagkräftigen Namen für den Bereich ein (siehe Abbildung 6.2).**

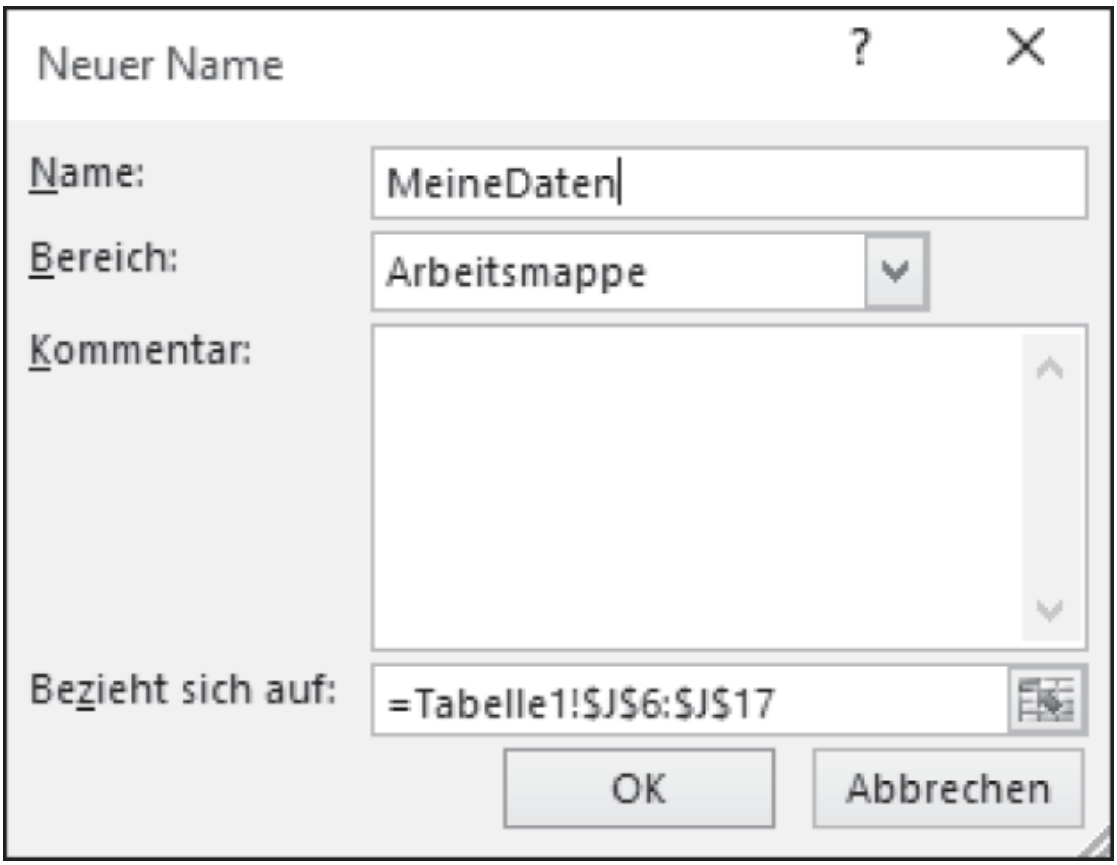


Abbildung 6.2: Hier legen Sie den Namen für den Bereich fest.

Nachdem Sie auf OK geklickt haben, besitzt der ausgewählte Bereich einen Namen. Um zu überprüfen, ob der Name richtig zugewiesen wurde, können Sie in der Registerkarte FORMELN auf NAMENS-MANAGER klicken. Das gleichnamige Dialogfeld wird angezeigt (siehe Abbildung 6.3). Sie sehen dort alle aktuell benannten Bereiche.

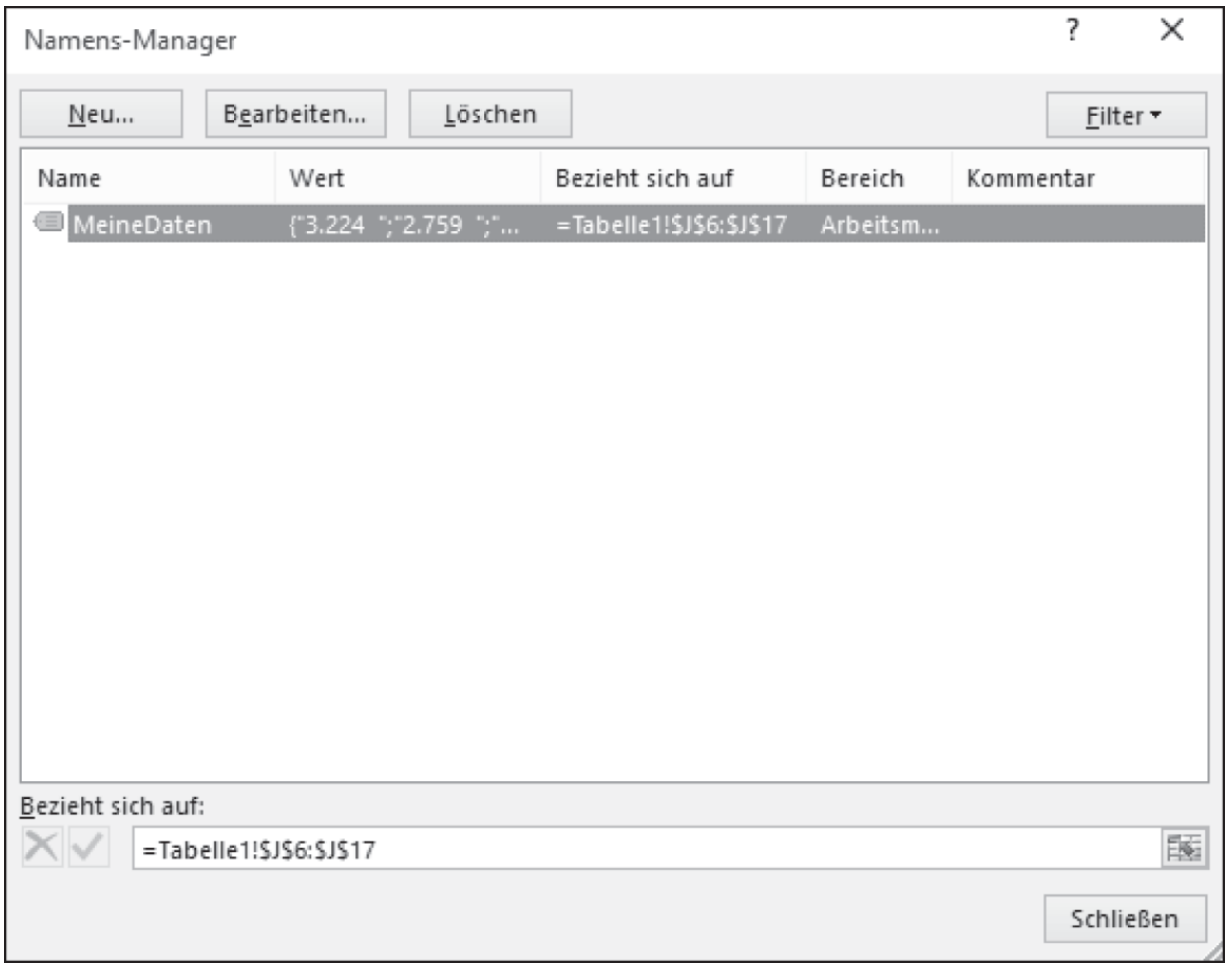


Abbildung 6.3: Das Dialogfeld »Namens-Manager« zeigt alle benannten Bereiche an.

Um mit VBA einen Namen für einen Bereich zu erstellen, ist nur eine Aktion erforderlich. Sie können direkt die Eigenschaft `Name` des `Range`-Objekts verwenden:

```
Sub Makro1()

 Range("D6:D17").Name = "MeineDaten"

End Sub
```

Zugegeben, vermutlich wird es selten vorkommen, dass Sie die Erstellung von benannten Bereichen automatisieren müssen. Die echten Vorteile ergeben sich erst, wenn Sie in VBA-Code benannte Bereiche bearbeiten.

So funktioniert es

Sie übergeben einfach den Namen des Zellbereichs an das `Range`-Objekt. Dies ermöglicht es, Zellbereiche auszuwählen:

```
Sub Makro1()

 Range("MeineDaten").Select"

End Sub
```

Wie bei normalen Bereichen können Sie auch auf benannte Bereiche mit der `With-End-With`-Anweisung zugreifen. Mit dieser Anweisung wird Excel darüber informieret, dass alle Aktionen auf das Objekt angewendet werden sollen, auf das Sie verweisen. Diese spart nicht nur Tipparbeit, da Sie die gleiche Syntax nicht wiederholt eingeben müssen, sondern Sie können den Code auch sehr einfach erweitern, indem Sie zwischen `With` und `End With` weitere Befehle einfügen.

```
Sub Makro1()

 With Range("MeineDaten")
 .NumberFormat = "#,##0"
 .Font.Bold = True
 .Interior.ColorIndex = 36
 End With

End Sub
```

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**

4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Alle Zellen eines Bereichs in einer Schleife durchlaufen

Eine Technik, die Sie als VBA-Programmierer unbedingt kennen müssen, besteht darin, alle Zellen eines Bereichs in einer Schleife zu durchlaufen. Wenn Sie in Excel ernsthaft mit Makros arbeiten wollen, kommen Sie eher früher als später an den Punkt, an dem Sie einen Bereich Zelle für Zelle durchlaufen und an diesen Zellen irgendwelche Aktionen vornehmen müssen.

Dieses Makro zeigt einen einfachen Weg, um einen Bereich zu durchlaufen.

So funktioniert es

In diesem Makro verwenden Sie zwei `Range`-Objektvariablen. Eine Variable enthält den gesamten Zellbereich, den Sie bearbeiten wollen, und in der zweiten befindet sich die Zelle, die Sie gerade aktuell bearbeiten. Anschließend verwenden Sie die `For-Each`-Anweisung, um alle Zellen des Bereichs nacheinander zu bearbeiten.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim MyCell As Range

'Schritt 2: Den Zielbereich definieren
 Set MyRange = Range("D6:D17")

'Schritt 3: Die einzelnen Zellen des Bereichs in einer
' Schleife durchlaufen
 For Each MyCell In MyRange

'Schritt 4: Mit jeder Zelle etwas machen
 If MyCell.Value&gt; 3000 Then
 MyCell.Font.Bold = True
```

```
    End If

'Schritt 5: Die nächste Zelle des Bereichs holen
    Next MyCell

End Sub
```

1. Das Makro deklariert zuerst zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zellbereich, der bearbeitet werden soll, und in `MyCell` befindet sich die jeweils aktuelle Zelle, die in der Schleife bearbeitet wird.
2. In Schritt 2 weisen Sie der Variablen `MyRange` den Zielbereich zu. In diesem Beispiel verwenden Sie `Range("D6:D17")`. Falls Sie dem Zielbereich einen Namen zugewiesen haben, können Sie hier auch einfach diesen Namen verwenden: `Range("Udo")`.
3. In Schritt 3 wird die Schleife gestartet und es werden alle Zellen des Zielbereichs durchlaufen. In der Schleife werden nach und nach die einzelnen Zellen aktiviert und der Variablen `MyCell` zugewiesen.
4. Nach der Aktivierung der Zelle können Sie etwas mit ihr machen. Dieses *Etwas* hängt natürlich davon ab, welche Aufgabe das Makro erledigen soll. Sie können beispielsweise Zeilen löschen, wenn die aktive Zelle einen bestimmten Wert enthält, oder nach jeder aktiven Zelle eine neue Zeile einfügen. In diesem Beispiel wird die Zelle fett formatiert, wenn der Zellwert größer ist als 3000.
5. In Schritt 5 macht das Makro mit der nächsten Zelle weiter. Das Makro wird beendet, nachdem alle Zellen im Zielbereich bearbeitet wurden.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

In einen Bereich leere Zeilen einfügen

Bisweilen kann es nötig sein, in Ihre Daten dynamisch neue Zeilen einzufügen. Auch wenn leere Zeilen in den meisten Fällen eher stören, muss die fertige Version eines Berichts oft leere Zeilen enthalten, um so die Daten zu trennen und besser lesbar zu machen. Das Makro dieses Abschnitts fügt in einen Bereich leere Zeilen ein.

So funktioniert es

Das Makro führt über den angegebenen Bereich von unten nach oben einen Schleifendurchlauf durch und verwendet hierfür einen Schleifenzähler. Es beginnt mit der letzten Zeile des Bereichs, fügt zwei leere Zeilen ein und macht dann mit der vorigen Zeile des Bereichs weiter. Bei jedem Schleifendurchlauf werden die gleichen Einfügungen vorgenommen, außerdem wird der Schleifenzähler angepasst.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim iCounter As Long

'Schritt 2: Den Zielbereich definieren
 Set MyRange = Range("C6:D17")

'Schritt 3: Den Bereich rückwärts in einer Schleife
```

```
' durchlaufen
 For iCounter = MyRange.Rows.Count To 2 Step -1

'Schritt 4: Zwei leere Zeilen einfügen
 MyRange.Rows(iCounter).EntireRow.Insert
 MyRange.Rows(iCounter).EntireRow.Insert

'Schritt 5: Den Wert des Zählers um 1 vermindern
 Next iCounter

End Sub
```

1. Sie deklarieren zuerst zwei Variablen. Die erste Variable ist eine Objektvariable mit dem Namen `MyRange`, die den Zielbereich definiert. Die zweite Variable ist eine Ganzzahlvariable des Typs `Long` mit dem Namen `iCounter`; diese dient als Schleifenzähler.
2. In Schritt 2 wird der Variablen `MyRange` der Zielbereich zugewiesen, in unserem Beispiel ist dies der Bereich `Range("C6:D17")`. Falls Sie dem Zielbereich einen Namen zugewiesen haben, können Sie hier auch einfach diesen Namen verwenden: `Range("Eva")`.
3. In diesem Schritt wird die Schleife gestartet, der Schleifenzähler initialisiert und die Schleifenende-Bedingung festgelegt. Der Startwert des Schleifenzählers wird auf die Anzahl der Zeilen im Zielbereich festgelegt (`MyRange.Rows.Count`); die Schleife soll beendet werden, wenn der Schleifenzähler den Wert 2 besitzt (die zweite Zeile des Zielbereichs). Beachten Sie, dass Sie Excel mit der Anweisung `Step -1` darüber informieren, dass der Zähler jeweils um den Wert 1 vermindert werden soll. So erreichen Sie, dass Sie den Bereich von unten (größter Wert des Zählers) nach oben (kleinster Wert des Zählers) durchlaufen. Kurz gesagt, Sie weisen Excel in Schritt 3 an, bei der letzten Zeile des Zielbereichs zu beginnen und den Schleifenzähler so lange zu dekrementieren (um den Wert 1 zu vermindern), bis die zweite Zeile des Zielbereichs erreicht wird.

Wenn Sie mit Bereichen arbeiten, können Sie eine Zeile des Bereichs direkt ansprechen, indem Sie die Zeilennummer als Index an die Sammlung `Rows` dieses Bereichs übergeben. So zeigt beispielsweise `Range("D6:D17").Rows(5)` auf die fünfte Zeile des Bereichs D6:D17.

4. In Schritt 4 verwendet das Makro die Variable `iCounter` als Index für die Sammlung `Rows` von `MyRange`. Diese Variable bestimmt die Nummer der Zeile, die derzeit in der Schleife vom Makro bearbeitet wird. Anschließend wird die Methode `EntireRow.Insert` verwendet, um eine leere Zeile einzufügen. Da Sie zwei leere Zeilen erhalten wollen, rufen Sie `EntireRow.Insert` zwei Mal auf.
5. In Schritt 5 macht das Makro mit dem nächsten Wert des Schleifenzählers weiter.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Alle Zeilen und Spalten einblenden

Wenn Sie eine Kalkulationstabelle prüfen, die Sie nicht selbst erstellt haben, wollen Sie oft sicherstellen, dass Sie auch wirklich alles sehen können, was sich auf dem Tabellenblatt befindet. Sie

müssen dann dafür sorgen, dass keine Zeilen oder Spalten ausgeblendet sein. Dieses einfache Makro automatisiert den Vorgang und blendet zuverlässig alle Zeilen und Spalten ein.

So funktioniert es

In diesem Makro verwenden Sie die Sammlungen `Columns` und `Rows` des Tabellenblatts. Jede Sammlung stellt Eigenschaften zur Verfügung, die festlegen, ob deren Objekte sichtbar sind oder nicht. Wenn Sie das Makro starten, werden alle Spalten der Sammlung `Columns` und alle Zeilen der Sammlung `Rows` eingeblendet.

```
Sub Makro1()

 Columns.EntireColumn.Hidden = False
 Rows.EntireRow.Hidden = False

End Sub
```

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein

Makro aufzeichnen und dabei die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Leere Zeilen löschen

Wenn Sie lange genug mit Excel gearbeitet haben, wissen Sie, dass leere Zeilen oft auf mehreren Ebenen Chaos verursachen können. Sie können zu Problemen bei Formeln führen, bergen Risiken beim Kopieren und Einfügen und können manchmal auch dazu führen, dass sich Pivot-Tabellen seltsam verhalten. Falls Sie in Ihren Daten regelmäßig nach leeren Zeilen suchen und diese manuell löschen, kann Ihnen dieses Makro die Arbeit abnehmen.

So funktioniert es

In diesem Makro verwenden Sie die Eigenschaft `UsedRange` des `ActiveSheet`-Objekts, um den Bereich zu definieren, den Sie bearbeiten wollen. Die `UsedRange`-Eigenschaft gibt einen Bereich zurück, der alle Zellen des Tabellenblatts umfasst, in die Daten eingegeben wurden. Sie richten dann einen Schleifenzähler ein, der bei der letzten Zeile des Bereichs beginnt, und überprüfen dann, ob die komplette Zeile leer ist. Falls diese Zeile tatsächlich leer ist, löschen Sie diese Zeile. In der Schleife führen Sie die gleiche Aktion für jede Zeile aus und setzen den Schleifenzähler jeweils auf die vorige Zeile.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
```

```
 Dim iCounter As Long

'Schritt 2: Den Zielbereich definieren
 Set MyRange = ActiveSheet.UsedRange

'Schritt 3: Den Zellbereich rückwärts durchlaufen
 For iCounter = MyRange.Rows.Count To 1 Step -1

'Schritt 4: Wenn die Zeile komplett leer ist, diese Zeile löschen
 If Application.CountA(Rows(iCounter).EntireRow) = 0 _
 Then
 Rows(iCounter).Delete
 End If

'Schritt 5: Den Schleifenzähler inkrementieren
 Next iCounter

End Sub
```

1. Das Makro deklariert zuerst zwei Variablen. Die erste Variable ist eine Objektvariable mit dem Namen `MyRange`; sie erhält später den Bereich, der bearbeitet werden soll. Die andere Variable ist eine Ganzzahlvariable des Typs `Long` mit dem Namen `iCounter`; diese dient als Schleifenzähler.
2. In Schritt 2 wird der Variablen `MyRange` der Zielbereich zugewiesen, in unserem Beispiel ist dies die Eigenschaft `UsedRange` des `ActiveSheet`-Objekts. `UsedRange` gibt einen Bereich mit allen Zellen zurück, in denen Daten eingegeben wurden. Sie können hier auch einen festgelegten oder benannten Bereich verwenden.
3. In Schritt 3 wird die Schleife gestartet, der Schleifenzähler initialisiert und die Schleifenende-Bedingung festgelegt. Der Startwert des Schleifenzählers wird auf die Anzahl der Zeilen im Zielbereich festgelegt (`MyRange.Rows.Count`); die Schleife soll beendet werden, wenn der Schleifenzähler den Wert 1 besitzt (die erste Zeile des Zielbereichs). Beachten Sie, dass Sie Excel mit der Anweisung `Step -1` darüber informieren, dass der Zähler jeweils um den Wert 1 vermindert werden soll. So erreichen Sie, dass Sie den Bereich von unten (größter Wert des Zählers) nach oben (kleinster Wert des Zählers)

durchlaufen. Kurz gesagt, Sie weisen Excel in Schritt 3 an, bei der letzten Zeile des Zielbereichs zu beginnen und den Schleifenzähler so lange zu dekrementieren (um den Wert 1 zu vermindern), bis die erste Zeile des Zielbereichs erreicht wird.

Wenn Sie mit Bereichen arbeiten, können Sie direkt eine Zeile des Bereichs ansprechen, indem Sie die Zeilennummer als Index an die Sammlung `Rows` des Bereichs übergeben. So zeigt beispielsweise `Range("D6:D17").Rows(5)` die fünfte Zeile des Bereichs D6:D17.

4. In Schritt 4 verwendet das Makro die Variable `iCounter` als Index für die Sammlung `Rows` von `MyRange`. Diese Variable bestimmt die Nummer der Zeile, die derzeit in der Schleife vom Makro bearbeitet wird. Das Makro überprüft, ob alle Zellen in dieser Zeile leer sind. Ist dies der Fall, wird die Zeile gelöscht.
5. In Schritt 5 macht das Makro mit dem nächsten Wert des Schleifenzählers weiter.

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Leere Spalten löschen

Wie leere Zeilen besitzen auch leere Spalten das Potenzial, zu unvorhergesehenen Fehlern zu führen. Falls Sie in Ihren Daten regelmäßig nach leeren Spalten suchen und diese manuell löschen, kann Ihnen das folgende Makro auch diese Arbeit abnehmen.

So funktioniert es

In diesem Makro verwenden Sie die Eigenschaft `UsedRange` des `ActiveSheet`-Objekts, um den Bereich zu definieren, den Sie bearbeiten wollen. Die `UsedRange`-Eigenschaft gibt einen Bereich zurück, der alle Zellen des Tabellenblatts umfasst, in die Daten eingegeben wurden. Sie richten dann einen Schleifenzähler ein, der bei der letzten Spalte des Bereichs beginnt, und überprüfen, ob die komplette Spalte leer ist. Falls diese Spalte tatsächlich leer ist, löschen Sie diese Spalte. In der Schleife führen Sie die gleiche Aktion für jede Spalte aus und setzen den Schleifenzähler jeweils auf die vorige Spalte.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
```

```
 Dim iCounter As Long

'Schritt 2: Definieren Sie den Zielbereich
 Set MyRange = ActiveSheet.UsedRange

'Schritt 3: Den Bereich in einer Schleife rückwärts
' durchlaufen
 For iCounter = MyRange.Columns.Count To 1 Step -1

'Schritt 4: Wenn die aktuelle Spalte komplett leer ist,
' diese Spalte löschen
 If Application.CountA(Columns(iCounter). _
 EntireColumn) = 0 Then
 Columns(iCounter).Delete
 End If

'Schritt 5: Den Schleifenzähler dekrementieren
 Next iCounter

End Sub
```

1. Das Makro deklariert zuerst zwei Variablen. Die erste Variable ist eine Objektvariable mit dem Namen `MyRange`; sie erhält später den Bereich, der bearbeitet werden soll. Die andere Variable ist eine Ganzzahlvariable des Typs `Long` mit dem Namen `iCounter`; diese dient als Schleifenzähler.
2. In Schritt 2 wird der Variablen `MyRange` der Zielbereich zugewiesen, in unserem Beispiel ist dies die Eigenschaft `UsedRange` des `ActiveSheet`-Objekts. `UsedRange` gibt einen Bereich mit allen Zellen zurück, in denen Daten eingegeben wurden. Sie können hier auch einen festgelegten oder benannten Bereich verwenden.
3. In Schritt 3 wird die Schleife gestartet, der Schleifenzähler initialisiert und die Schleifenende-Bedingung festgelegt. Der Startwert des Schleifenzählers wird auf die Anzahl der Spalten im Zielbereich festgelegt (`MyRange.Columns.Count`); die Schleife soll beendet werden, wenn der Schleifenzähler den Wert 1 besitzt. Beachten Sie, dass Sie die Anweisung `Step -1` verwenden. Mit `Step -1` weisen Sie Excel an, dass der Zähler jeweils um den Wert 1 vermindert werden soll. So erreichen

Sie, dass Sie den Bereich von rechts (größter Wert des Zählers) nach links (kleinster Wert des Zählers) durchlaufen. Kurz gesagt, Sie weisen Excel in Schritt 3 an, bei der letzten Spalte des Zielbereichs zu beginnen und den Schleifenzähler so lange zu dekrementieren (um den Wert 1 zu vermindern), bis die erste Spalte des Zielbereichs erreicht wird.

Wenn Sie mit Bereichen arbeiten, können Sie direkt eine Spalte des Bereichs ansprechen, indem Sie die Spaltennummer als Index an die Sammlung `Columns` des Bereichs übergeben. So zeigt beispielsweise `Range("A1:D17").Columns(2)` auf die zweite Spalte des Bereichs A1:D17, nämlich Spalte B.

4. In Schritt 4 verwendet das Makro die Variable `iCounter` als Index für die Sammlung `Columns` von `MyRange`. Diese Variable gibt die Nummer der Spalte an, die derzeit in der Schleife vom Makro bearbeitet wird. Das Makro überprüft, ob alle Zellen in dieser Spalte leer sind. Ist dies der Fall, wird die komplette Spalte gelöscht.
5. In Schritt 5 macht das Makro mit dem nächsten Wert des Schleifenzählers weiter.

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**

4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Den Scrollbereich einschränken

Excel gibt Ihnen die Möglichkeit, den Bereich einzuschränken, in dem der Anwender scrollen und Zellen auswählen kann. Das hier beschriebene Makro lässt sich ohne großen Aufwand sofort umsetzen.

So funktioniert es

Mit der Eigenschaft `ScrollArea` können Sie den Bildlauf beeinflussen, das heißt den Bereich eines Tabellenblatts festlegen, in dem gescrollt werden kann. Die folgende Anweisung beispielsweise legt den Scrollbereich für `Tabelle1` fest, sodass der Anwender keine Zellen mehr aktivieren kann, die außerhalb des Bereichs A1:M17 liegen:

```
Sheets("Tabelle1").ScrollArea = "A1:M17"
```

Da diese Einstellung nicht in der Arbeitsmappe gespeichert wird, müssen Sie sie bei jedem Öffnen der Mappe erneut festlegen. Sie erreichen dies, indem Sie diese Anweisung in die Arbeitsmappen-Ereignisprozedur `Workbook_Open` einfügen:

```
Private Sub Worksheet_Open()
 Sheets("Tabelle1").ScrollArea = "A1:M17"
End Sub
```

Falls Sie die Einschränkungen des Scrollbereichs aus irgendeinem Grunde wieder aufheben wollen, entfernen Sie sie mit dieser Anweisung:

```
ActiveSheet.ScrollArea = ""
```

So verwenden Sie es

Um dieses Makro zu implementieren, müssen Sie es kopieren und in die Ereignisprozedur für das Ereignis `Workbook_Open` einfügen. Indem Sie das Makro so erstellen, gewährleisten Sie, dass es immer dann ausgeführt wird, wenn die Arbeitsmappe geöffnet wird:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie auf DIESEARBEITSMAPPE.**
4. **Wählen Sie in der Dropdownliste EREIGNISSE (siehe Abbildung 6.4) das Ereignis OPEN aus.**
5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin.**

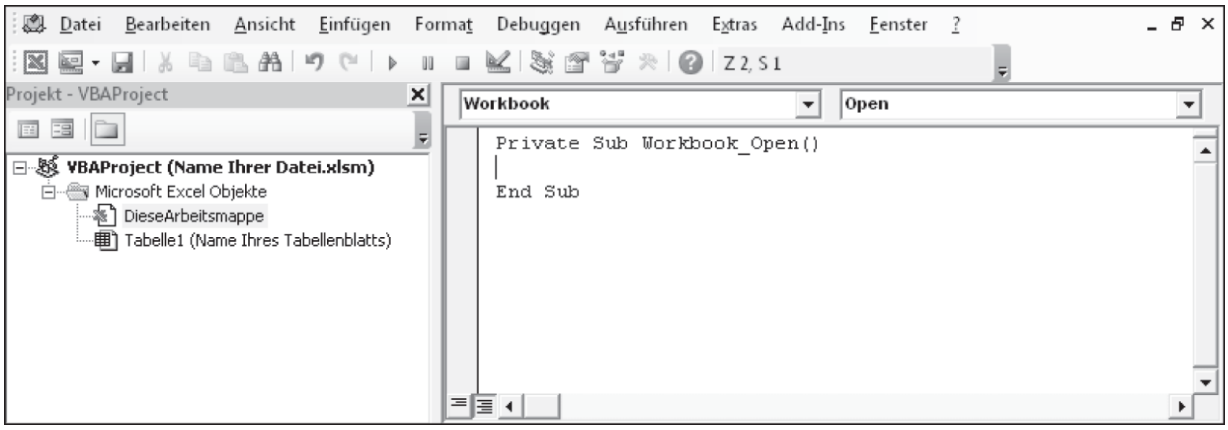


Abbildung 6.4: Tippen Sie den Code in die Prozedur für das Arbeitsmappenereignis Open ein oder kopieren Sie den Code dorthin.

Alle Formeln in einer Arbeitsmappe auswählen und formatieren

Wenn Sie eine Excel-Arbeitsmappe überprüfen, ist es extrem wichtig, gut auf alle Formeln in jedem Tabellenblatt zugreifen zu können. Hierzu müssen Sie die Formeln natürlich zuerst einmal alle finden, was mühsam sein kann, wenn Sie diese Aufgabe von Hand erledigen.

Excel bietet jedoch ein cleveres Feature, um alle Formeln in einem Tabellenblatt zu finden und zu markieren. Das Makro in diesem Abschnitt macht sich diese Funktionalität zunutze, um dynamisch alle Zellen aufzustöbern, die Formeln enthalten.

So funktioniert es

Excel kennt eine Reihe spezieller Zelltypen, die Sie mit dem Dialogfeld INHALTE AUSWäHLEN markieren können. Um die Zellen von Hand zu markieren, öffnen Sie die Registerkarte START und klicken in der Gruppe BEARBEITEN zuerst auf SUCHEN UND AUSWäHLEN und dann auf INHALTE AUSWäHLEN. Excel zeigt dann das gleichnamige Dialogfeld an (siehe Abbildung 6.5). Alternativ können Sie auch die Taste F5 verwenden und dann im Dialogfeld GEHE ZU auf INHALTE klicken.

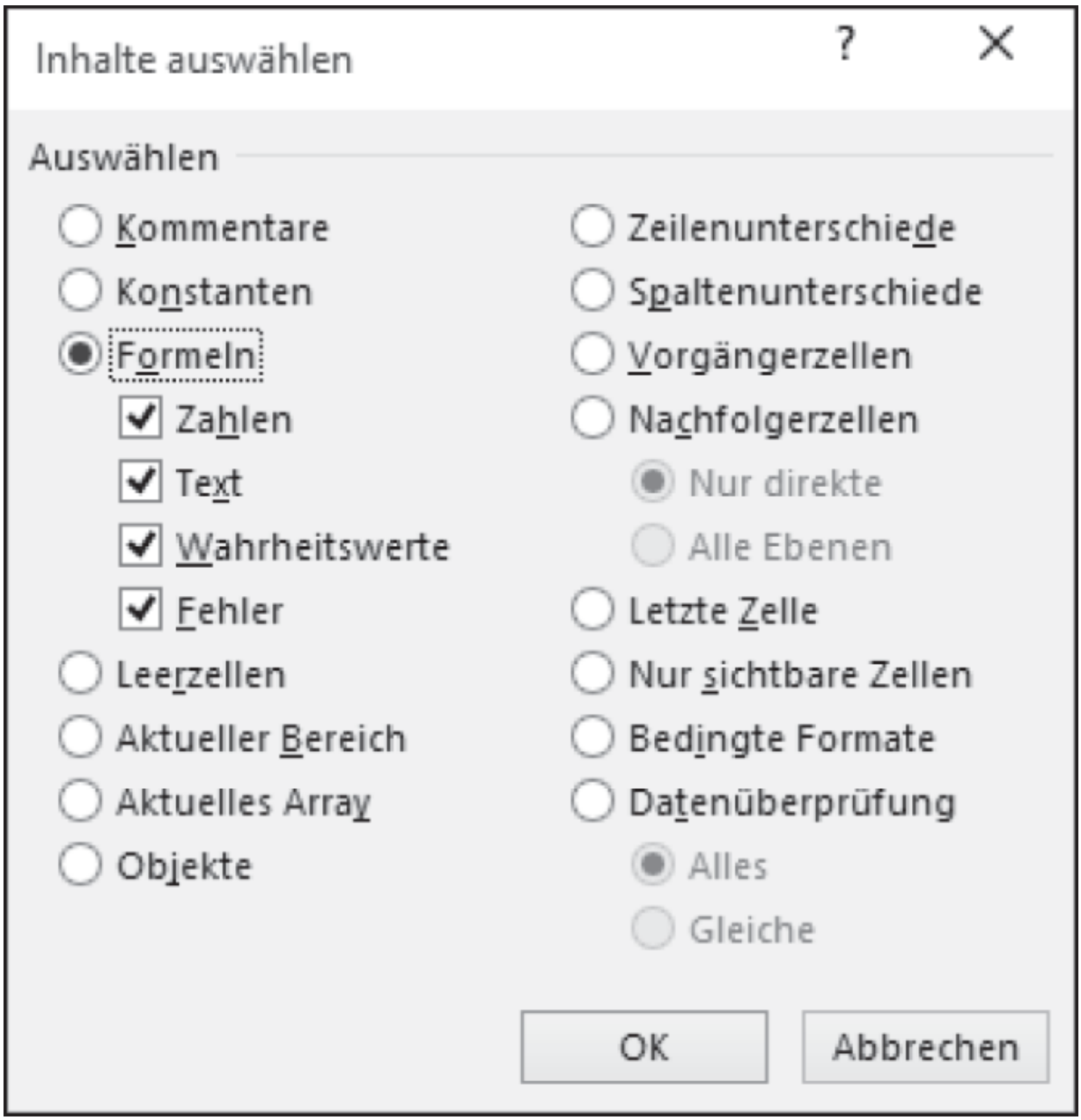
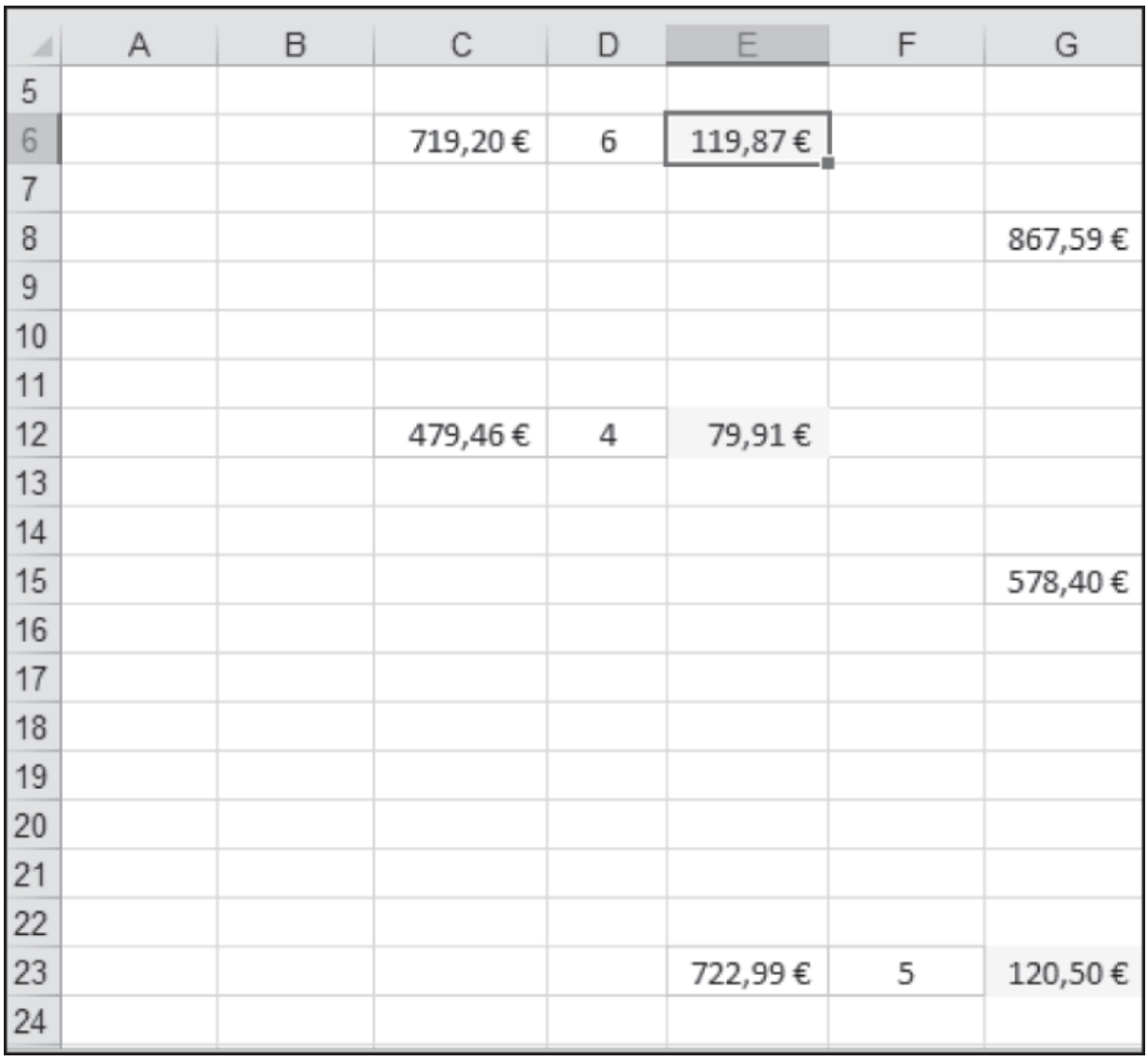


Abbildung 6.5: Das Dialogfeld »Inhalte auswählen«

In diesem Dialogfeld legen Sie fest, nach welchen Inhalten Excel im aktiven Tabellenblatt suchen soll; eine Option sind die Formeln. Wenn Sie hier also FORMELN einschalten, werden alle Zellen ausgewählt, in denen sich eine Formel befindet (siehe Abbildung 6.6). An diesem Punkt können Sie die Zellen einfärben, um anzuzeigen, dass sie Formeln enthalten.



	A	B	C	D	E	F	G
5							
6			719,20 €	6	119,87 €		
7							
8							867,59 €
9							
10							
11							
12			479,46 €	4	79,91 €		
13							
14							
15							578,40 €
16							
17							
18							
19							
20							
21							
22							
23					722,99 €	5	120,50 €
24							

Abbildung 6.6: Aktivieren Sie im Dialogfeld »Inhalte auswählen« das Kontrollkästchen »Formeln«, um Excel anzuweisen, nur die Zellen auszuwählen, in denen sich eine Formel befindet.

Dieses Makro führt für alle Tabellenblätter der gesamten Arbeitsmappe die gleiche Aktion durch. Sie verwenden hier die Methode `SpecialCells` der Sammlung `Cells`. Bei der Methode `SpecialCells` geben Sie mit einem Parameter den Zelltyp an, den Sie auswählen wollen. In unserem Beispiel verwenden Sie `xlCellTypeFormulas`.

Kurz gesagt, Sie verweisen auf einen speziellen Zellbereich, der nur die Zellen enthält, in denen Formeln stehen. Anschließend

verwenden Sie für diesen Zellbereich die `With-End-With`-Anweisung, um die Zellen in diesem Bereich einzufärben.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim ws As Worksheet

'Schritt 2: Fehlermeldung unterdrücken, falls keine
' Formeln gefunden wurden
 On Error Resume Next

'Schritt 3: Die einzelnen Tabellenblätter in einer
' Schleife durchlaufen
 For Each ws In ActiveWorkbook.Worksheets

'Schritt 4: Zellen mit Formeln markieren und hervorheben
 With ws.Cells.SpecialCells(xlCellTypeFormulas)
 .Interior.ColorIndex = 36
 End With

'Schritt 5: Nächstes Tabellenblatt holen
 Next ws

End Sub
```

1. In Schritt 1 wird eine Objektvariable mit dem Namen `ws` deklariert. In dieser Variablen wird ein Verweis auf das derzeit in der Schleife bearbeitete Tabellenblatt abgelegt.
2. Falls ein Tabellenblatt keine Formeln enthält, erzeugt Excel beim Versuch, Zellen mit Formeln auszuwählen, eine Fehlermeldung. Um diesen Fehler zu ignorieren, weisen Sie Excel daher an, auch beim Auftreten eines Fehlers mit der Makroausführung weiterzumachen.
3. Die Schleife beginnt in Schritt 3; hier wird Excel angewiesen, alle Tabellenblätter der Arbeitsmappe zu bearbeiten.
4. In diesem Schritt wählt das Makro alle Zellen aus, die Formeln enthalten, und formatiert sie.
5. In Schritt 5 springen Sie zum Anfang der Schleife und machen mit dem nächsten Tabellenblatt weiter. Nachdem alle Tabellenblätter bearbeitet sind, endet das Makro.

So verwenden Sie es

Der beste Speicherort für dieses Makro ist Ihre persönliche Makroarbeitsmappe. Auf diese Weise steht Ihnen das Makro immer zur Verfügung. Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf PERSONAL.XLSB.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Kopieren Sie den Code in das neu erstellte Modul oder tippen Sie ihn ein.**

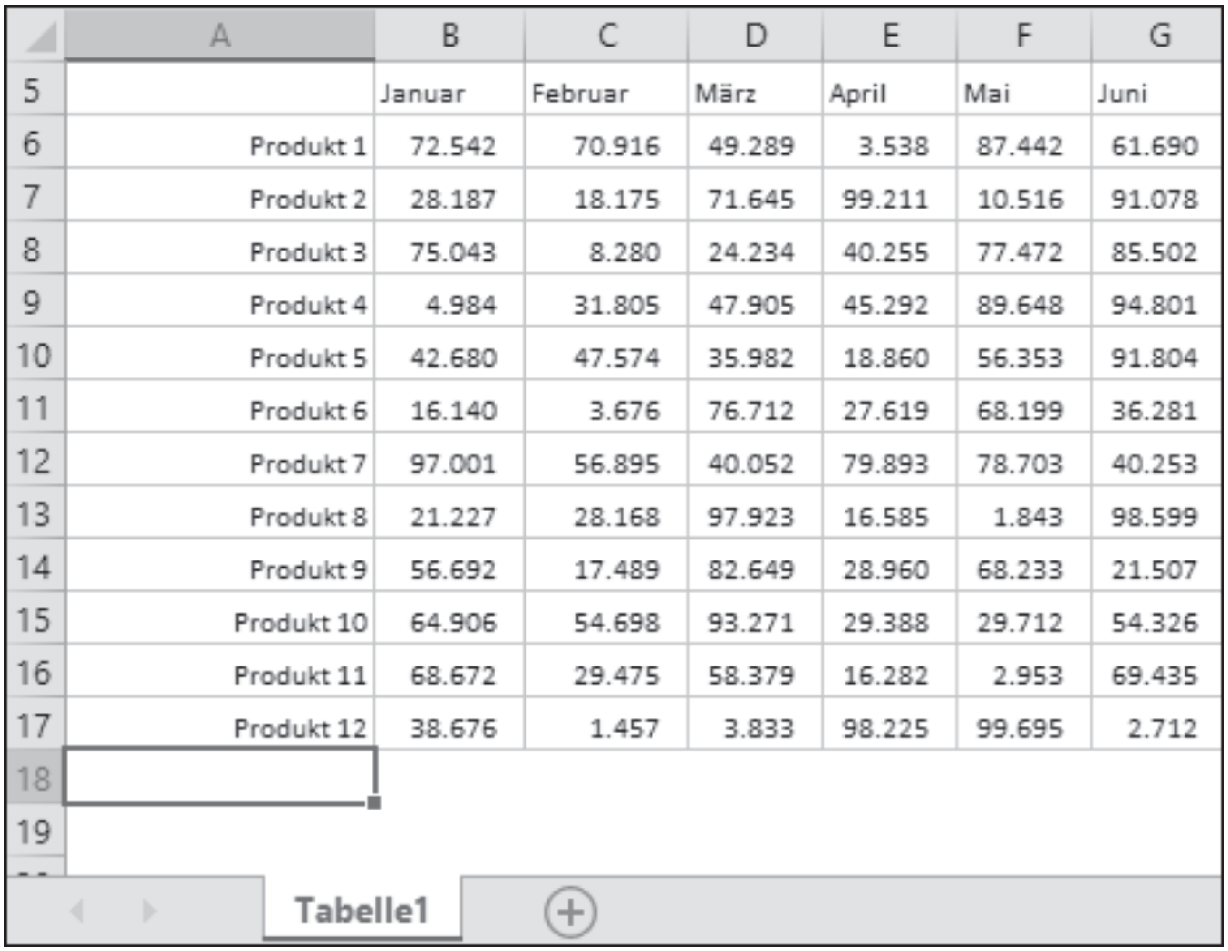
Falls Sie die Datei *personal.xlsb* im Projekt-Explorer nicht sehen, wurde sie noch nicht erstellt. Um sie zu erstellen, müssen Sie ein Makro aufzeichnen und die persönliche Makroarbeitsmappe als Speicherort angeben.

Um ein Makro in Ihrer persönlichen Makroarbeitsmappe aufzuzeichnen, öffnen Sie das Dialogfeld MAKRO AUFZEICHNEN. Öffnen Sie die Liste MAKRO SPEICHERN IN und wählen Sie PERSöNLICHE MAKROARBEITSMAPPE. Zeichnen Sie dann einfach ein paar Klicks auf und beenden Sie die Aufzeichnung. Sie können dann die aufgezeichneten Anweisungen entfernen und diese durch den Code des obigen Makros ersetzen.

Die erste leere Zeile oder Spalte finden und auswählen

Geht es Ihnen auch häufiger so, dass Sie Zeilen oder Spalten in eine vorhandene Datentabelle einfügen müssen? Wenn Sie

unterhalb der Tabelle Zeilen einfügen wollen, müssen Sie zuerst die letzte Zeile ermitteln, dann die Markierung eine Zeile nach unten bewegen und abschließend die Zeile einfügen (wie in Abbildung 6.7 zu sehen). Das Gleiche gilt auch für das Einfügen einer Spalte: Sie müssen die letzte benutzte Spalte ermitteln und dann die Markierung eine Zelle nach rechts bewegen.



	A	B	C	D	E	F	G
5		Januar	Februar	März	April	Mai	Juni
6	Produkt 1	72.542	70.916	49.289	3.538	87.442	61.690
7	Produkt 2	28.187	18.175	71.645	99.211	10.516	91.078
8	Produkt 3	75.043	8.280	24.234	40.255	77.472	85.502
9	Produkt 4	4.984	31.805	47.905	45.292	89.648	94.801
10	Produkt 5	42.680	47.574	35.982	18.860	56.353	91.804
11	Produkt 6	16.140	3.676	76.712	27.619	68.199	36.281
12	Produkt 7	97.001	56.895	40.052	79.893	78.703	40.253
13	Produkt 8	21.227	28.168	97.923	16.585	1.843	98.599
14	Produkt 9	56.692	17.489	82.649	28.960	68.233	21.507
15	Produkt 10	64.906	54.698	93.271	29.388	29.712	54.326
16	Produkt 11	68.672	29.475	58.379	16.282	2.953	69.435
17	Produkt 12	38.676	1.457	3.833	98.225	99.695	2.712
18							
19							

Tabelle1

Abbildung 6.7: Sie können ein Makro verwenden, um dynamisch die erste freie Zelle in einer Zeile beziehungsweise Spalte zu finden.

Die beiden Makros in diesem Abschnitt zeigen Ihnen, wie Sie dynamisch die erste freie Zeile oder Spalte finden und auswählen.

So funktioniert es

Diese Makros verwenden das Element `Cells` und die Eigenschaft `Offset` für die Navigation im Tabellenblatt.

Das Element `Cells` gehört zum `Range`-Objekt und stellt einen extrem einfachen Weg bereit, um mit Code Bereiche auszuwählen. Sie brauchen als Parameter nur die Nummern von Spalte und Zeile anzugeben. `Cells(5,4)` übersetzt sich also in Zeile 5, Spalte 4 (das heißt Zelle D5). Mit `Cells(16,4)` wiederum verweisen Sie auf Zeile 16, Spalte 4 (also Zelle D16). Und das Schönste dabei: Sie können an `Cells` nicht nur Zahlen übergeben, sondern auch Ausdrücke.

Mit `Cells(Rows.Count,1)` wählen Sie die letzte Zeile und die erste Spalte des Tabellenblatts aus. In Excel 2007 und neuer entspricht dies faktisch der Zelle A1048576.

Mit `Cells(1,Columns.Count)` wählen Sie die erste Zeile und die letzte Spalte des Tabellenblatts aus. In Excel 2007 und neuer entspricht dies faktisch der Zelle XFD1.

Indem Sie die `Cells`-Anweisung mit der Eigenschaft `End` kombinieren, können Sie Excel zur letzten verwendeten Zeile oder Spalte springen lassen. Diese Anweisung liefert das gleiche Ergebnis, wie wenn Sie manuell zur Zelle A1048576 gehen und dann auf der Tastatur Strg+⇧+↑ eingeben. Wenn Sie die folgende Codezeile ausführen, springt Excel zur letzten benutzten Zeile in Spalte A:

```
Cells(Rows.Count, 1).End(xlUp).Select
```

Die folgende Anweisung bringt Sie zur letzten verwendeten Spalte in Zeile 1. Sie entspricht der Auswahl der Zelle XFD1 und dem anschließenden Drücken von Strg+⇧+← auf der Tastatur:

```
Cells(1, Columns.Count).End(xlToLeft).Select
```

Nachdem Sie sich in der letzten Spalte beziehungsweise Zeile eingefunden haben, können Sie die Eigenschaft `Offset` verwenden, um die Zellmarkierung nach unten beziehungsweise nach rechts zu verschieben.

Bei der Eigenschaft `Offset` geben Sie die gewünschte Bewegung als Anzahl an Zeilen und Spalten an.

Diese Anweisung wählt beispielsweise die Zelle A2 aus: Der Ausgangspunkt ist Zelle A1 und die Zeile soll um 1 erhöht werden, während die Spalte wegen der 0 unverändert bleibt):

```
Range("A1").Offset(1, 0).Select
```

Mit der folgenden Anweisung wählen Sie die Zelle C4 aus: Der Ausgangspunkt ist Zelle A1 und die Zeile soll um 3 und die Spalte um 2 erhöht werden:

```
Range("A1").Offset(3, 2).Select
```

Indem Sie die verschiedenen Konzepte miteinander kombinieren, können Sie zwei Makros erstellen, welche die erste leere Zeile beziehungsweise die erste leere Spalte markieren.

Dieses Makro wählt die erste leere Zeile aus:

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim LastRow As Long

'Schritt 2: Die Nummer der letzten benutzten
' Zeile ermitteln
 LastRow = Cells(Rows.Count, 1).End(xlUp).Row

'Schritt 3: Die Zeile nach der letzten benutzten markieren
 Cells(LastRow, 1).Offset(1, 0).Select

End Sub
```

1. Zuerst deklarieren Sie eine Ganzzahlvariable des Typs `Long` mit dem Namen `LastRow`; hier wird die Nummer der letzten Zeile abgespeichert.
2. In Schritt 2 berechnen Sie die letzte benutzte Zeile, indem Sie in der letzten Zeile des Tabellenblatts beginnen und sich dann mithilfe der Eigenschaft `End` zur letzten nicht leeren Zelle bewegen (dies entspricht dem Auswählen von Zelle A1048576 und anschließendem Drücken von Strg+⇧+↑ auf der Tastatur).

3. In diesem Schritt verwenden Sie die Eigenschaft `Offset`, um sich um eine Zeile nach unten zu bewegen, und wählen dann die erste leere Zelle in Spalte A aus.

Dieses Makro wählt die erste leere Spalte aus:

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim LastColumn As Long

'Schritt 2: Die Nummer der letzten benutzten
' Spalte ermitteln
 LastColumn = Cells(5, Columns.Count).End(xlToLeft).Column

'Schritt 3: Die Spalte rechts von der letzten
' benutzten markieren
 Cells(5, LastColumn).Offset(0, 1).Select

End Sub
```

In Schritt 1 deklarieren Sie eine Ganzzahlvariable des Typs `Long` mit dem Namen `LastColumn`; hier wird die Nummer der letzten Spalte abgespeichert.

In Schritt 2 berechnen Sie die letzte benutzte Spalte, indem Sie in der letzten Spalte des Tabellenblatts beginnen und sich dann mithilfe der Eigenschaft `End` zur letzten, nicht leeren Zelle bewegen (dies entspricht dem Auswählen von Zelle XFD5 und anschließendem Drücken von Strg+⇧+←).

In Schritt 3 verwenden Sie die Eigenschaft `Offset`, um sich um eine Spalte nach rechts zu bewegen und wählen dann die erste leere Zelle in Zeile 5 aus.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt+F11 drücken.**

2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Kapitel 7

Makros für das Bearbeiten von Zelldaten

IN DIESEM KAPITEL

- Einen Bereich kopieren und einfügen und dort Formeln in Werte konvertieren
- »Text in Spalten« für alle Spalten anwenden
- Nachstehende Minuszeichen konvertieren und Zellen mit führenden Nullen versehen
- Vor oder nach dem Zellinhalt Text einfügen
- Daten bereinigen: Duplikate, zusätzliche Zellinhalte sowie leere Zellen
- AutoFilter-Drop-down-Listen ausblenden
- Gefilterte Zeilen kopieren und in der Statusleiste Spalten anzeigen, nach denen gefiltert wird

Beim Arbeiten mit Informationen in Excel müssen Sie häufig die vorhandenen Daten irgendwie transformieren. Daten transformieren bedeutet, sie zu konvertieren, zu bereinigen, in ein Standardformat zu bringen oder anderweitig umzuformen, dass sie zu der Aufgabe passen, die Sie erledigen müssen. Das Umwandeln von Daten kann vielerlei bedeuten: zum Beispiel das Entfernen überflüssiger Leerzeichen, das Auffüllen mit führenden Nullen oder das Filtern der Daten nach bestimmten Kriterien.

Dieses Kapitel stellt einige nützliche Makros vor, mit denen Sie die Daten Ihrer Arbeitsmappen dynamisch transformieren können. Sie können mehrere dieser Makros miteinander kombinieren und

erhalten so Schritt für Schritt einen Werkzeugkasten, mit dem Sie Ihre Daten so umformen, wie Sie es gerade brauchen.

Einen Zellbereich kopieren und einfügen

Eine der Aufgaben, vor denen Sie sehr häufig stehen, ist das Kopieren und Einfügen eines Zellbereichs. Es ist sehr einfach, dies von Hand zu tun. Glücklicherweise ist das Kopieren und Einfügen mit VBA auch nicht schwer.

So funktioniert es

In diesem Makro verwenden Sie die Methode `Copy` des `Range`-Objekts, um Daten von D6:D17 nach L6:L17 zu kopieren. Bitte beachten Sie die Verwendung des Parameters `Destination`. Dieses Argument informiert Excel darüber, wo die Daten eingefügt werden sollen:

```
Sub Makro1()

 Sheets("Tabelle1").Range("D6:D17").Copy _
 Destination:=Sheets("Tabelle1").Range("L6:L17")

End Sub
```

Beim Arbeiten mit Ihrer Kalkulationstabelle müssen Sie ab und zu Formeln kopieren und diese als Werte einfügen. Um dies mithilfe eines Makros zu erledigen, verwenden Sie die Methode `PasteSpecial`. Im folgenden Beispiel kopieren Sie die Formeln in F6:F17 nach M6:M17. Beachten Sie, dass Sie mit `xlPasteValues` die Formeln als Werte einfügen und dass Sie anschließend außerdem `xlPasteFormats` verwenden, um die Formatierung des Quellbereichs dann auch noch auf den Zielbereich zu übertragen.

```
Sub Makro1()

 Sheets("Tabelle1").Range("F6:F17").Copy
 Sheets("Tabelle1").Range("M6:M17").PasteSpecial _
 xlPasteValues
```

```
    Sheets("Tabelle1").Range("M6:M17").PasteSpecial _
    xlPasteFormats

End Sub
```

Bitte beachten Sie, dass diese Bereiche lediglich als Beispiel dienen. Sie müssen die Bereiche so anpassen, dass sie zu den Daten in Ihrem Tabellenblatt passen.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Alle Formeln eines Zellbereichs in Werte konvertieren

In bestimmten Situationen möchten Sie die mühsam ausgetüftelten Formeln in Ihrem Tabellenblatt entfernen, bevor Sie die Arbeitsmappe weitergeben. Sie können dazu alle Formeln in einem bestimmten Bereich durch die aktuellen Werte ersetzen.

So funktioniert es

In diesem Makro verwenden Sie zwei `Range`-Objekte. Eine der Variablen enthält den kompletten Bereich, den Sie bearbeiten wollen, die zweite nimmt die Zelle auf, die derzeit in der Schleife bearbeitet wird. Danach verwenden Sie eine `For-Each`-Schleife,

um der Reihe nach jede Zelle des Zielbereichs zu aktivieren und zu bearbeiten. Sie prüfen für jede dieser Zellen, ob sie eine Formel enthält. Ist dies der Fall, ersetzen Sie die Formel durch den in der Zelle angezeigten Wert.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim MyCell As Range

'Schritt 2: Arbeitsmappe speichern, bevor Zellen geändert
' werden?
 Select Case MsgBox("Diese Aktion kann nicht " & _
 "rückgängig gemacht werden. Soll die " & _
 "Arbeitsmappe zuerst gespeichert werden?", _
 vbYesNoCancel)

 Case Is = vbYes
 ThisWorkbook.Save

 Case Is = vbCancel
 Exit Sub
 End Select

'Schritt 3: Definieren Sie den Zielbereich
 Set MyRange = Selection

'Schritt 4: Den gesamten Bereich in einer Schleife
' durchlaufen
 For Each MyCell In MyRange

'Schritt 5: Wenn die Zelle eine Formel enthält, die Zelle
' auf den angezeigten Wert setzen
 If MyCell.HasFormula Then
 MyCell.Formula = MyCell.Value
 End If

'Step 6: Die nächste Zelle im Bereich holen
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell`

nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.

2. Alle Aktionen, die Sie in einem Makro ausführen, können nicht rückgängig gemacht werden. Da Sie im Makro Daten verändern, sollten Sie dem Anwender die Möglichkeit bieten, die Arbeitsmappe vor der Ausführung des Makros zu speichern. Diese Aufgabe wird in Schritt 2 erledigt.

   Dazu lassen Sie ein Meldungsfeld anzeigen, das fragt, ob die Arbeitsmappe gespeichert werden soll. Das Meldungsfeld bietet die Schaltflächen JA, NEIN und ABBRECHEN an. Wenn auf JA geklickt wird, wird die Arbeitsmappe gespeichert und die Makroausführung fortgesetzt. Ein Klick auf ABBRECHEN beendet die Prozedur, ohne das Makro weiter auszuführen. Die Makroausführung wird dann ohne voriges Speichern fortgesetzt, wenn auf NEIN geklickt wird.

3. In Schritt 3 wird der Zielbereich der Variablen `MyRange` zugewiesen. In unserem Beispiel wird die aktuelle Auswahl auf dem Tabellenblatt verwendet. Sie können `MyRange` auch auf einen bestimmten Bereich setzen, wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich")`.

4. Dieser Schritt startet die Schleife; bei jedem Schleifendurchlauf wird eine Zelle des Zielbereichs aktiviert.

5. Nach der Aktivierung der Zelle verwendet das Makro die Eigenschaft `HasFormula`, um zu prüfen, ob die Zelle eine Formel enthält. Enthält die Zelle eine Formel, wird ihr Inhalt auf den in der Zelle angezeigten Wert gesetzt. Hierdurch wird die Formel dauerhaft durch einen hart-kodierten Wert ersetzt.

6. In Schritt 6 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet, nachdem alle Zellen des Zielbereichs bearbeitet sind.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Den Befehl »Text in Spalten« auf alle Spalten anwenden

Wenn Sie Daten aus anderen Quellen importieren, kann es sein, dass Sie Zellen erhalten, in denen die Werte als Text formatiert sind. Excel zeigt neben diesen Zellen eine Smarttag-Schaltfläche an, die Sie darüber informiert, dass die Zelle als Text formatiert ist (siehe Abbildung 7.1).

C	D	E	F	G	H
	Einnahmen	Einheiten	Preis je Einheit		
Januar	3224,1791553785	43	74,98		
Februa	2758,83834214959	30	91,96		
März	1301,70489760339	25	52,07		
April					
Mai	2677,077357431O6	25	107,08		
Juni	3015,29997101164	42	71,79		
Juli	8892,72320795156	41	216,9		
August	3185,53161972604	41	77,7		
September	3770,3130347558	32	117,82		
Oktober	7357,41604042586	40	183,94		
November	4514,43505181198	39	115,75		
Dezember	6355,67839756981	43	147,81		

Die Zahl in dieser Zelle ist als Text formatiert oder es ist ein Apostroph vorangestellt.

Abbildung 7.1: Importierte Zahlen sind manchmal als Text formatiert.

Sie können dieses Problem manuell lösen, indem Sie auf der Registerkarte DATEN in der Gruppe DATENTOOLS auf TEXT IN SPALTEN klicken (siehe Abbildung 7.2). Excel zeigt daraufhin den Textkonvertierungs-Assistenten an (Abbildung 7.3). Sie brauchen die verschiedenen Schritte des Assistenten nicht einzeln zu durchlaufen. Klicken Sie einfach auf FERTIG STELLEN, damit das Problem gelöst wird.

Auch dies ist eine ziemlich einfache Aktion. Das Problem besteht jedoch darin, dass Sie in Excel TEXT IN SPALTEN nicht für mehrere Spalten durchführen können. Sie müssen diese Korrektur spaltenweise anwenden. Dies ist etwas lästig, sollte das Tabellenblatt zahlreiche Spalten mit dem gleichen Problem aufweisen.

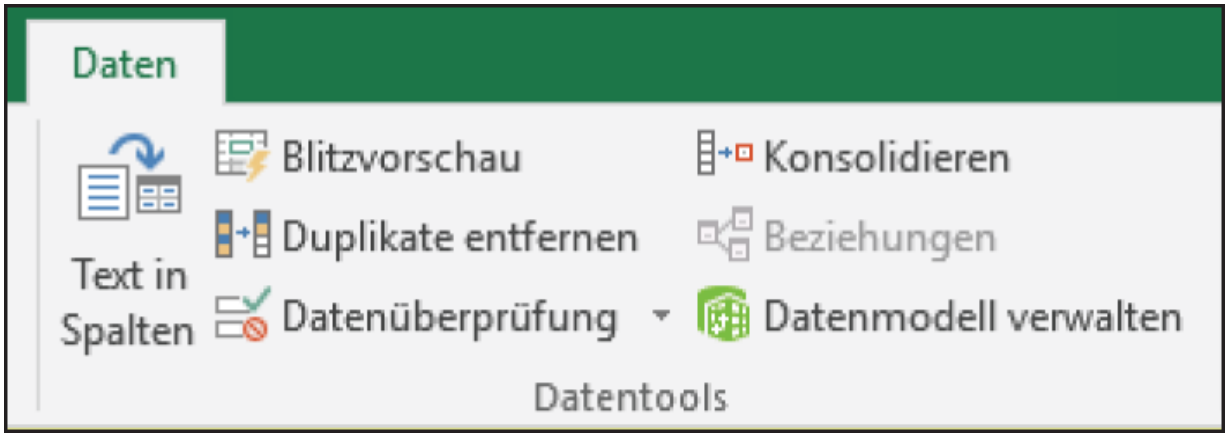


Abbildung 7.2: Wählen Sie den Befehl »Text in Spalten«.

Dieser Abschnitt enthält ein einfaches Makro, mit dem Sie mehrere Spalten wie oben beschrieben bearbeiten können.

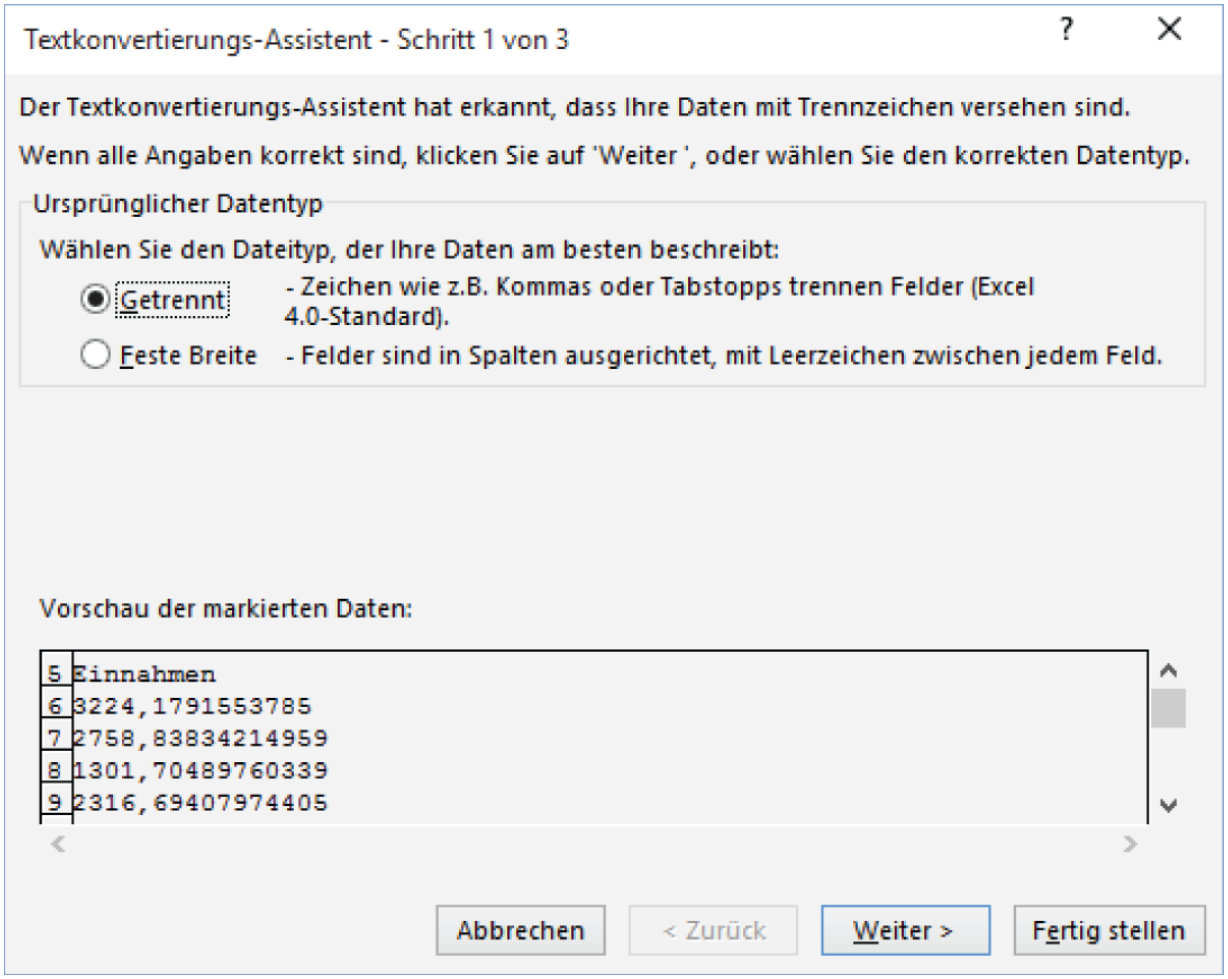


Abbildung 7.3: Klicken Sie auf »Fertig stellen«, um falsch formatierte Zahlen zu korrigieren.

So funktioniert es

In diesem Makro verwenden Sie zwei `Range`-Objekte. Eine der Variablen enthält den kompletten Bereich, den Sie bearbeiten wollen, die zweite nimmt die Zelle auf, die derzeit in der Schleife bearbeitet wird. Danach verwenden Sie eine `For-Each`-Schleife, um der Reihe nach jede Zelle des Zielbereichs zu aktivieren und zu bearbeiten. Hierbei setzen Sie einfach den Wert in der Zelle zurück. Das Makro macht das Gleiche wie der Befehl TEXT IN SPALTEN.

```
Sub Macro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
```

```
 Dim MyCell As Range


'Schritt 2: Arbeitsmappe speichern, bevor Zellen geändert
' werden?
 Select Case MsgBox("Diese Aktion kann nicht " & _
 "rückgängig gemacht werden. Soll die " & _
 "Arbeitsmappe zuerst gespeichert werden?", _
 vbYesNoCancel)

 Case Is = vbYes
 ThisWorkbook.Save

 Case Is = vbCancel
 Exit Sub
 End Select


'Schritt 3: Den Zielbereich festlegen
 Set MyRange = Selection


'Schritt 4: Jede Zelle des Bereichs in einer
' Schleife bearbeiten
 For Each MyCell In MyRange


'Schritt 5: Den Wert der Zelle zurücksetzen, um
' fehlerhafte Formate zu löschen
 If Not IsEmpty (MyCell) Then
 MyCell.Value = MyCell.Value
 End If


'Schritt 6: Die nächste Zelle des Bereichs holen
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell` nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.
2. Alle Aktionen, die Sie in einem Makro ausführen, können nicht rückgängig gemacht werden. Da Sie im Makro Daten

verändern, sollten Sie dem Anwender die Möglichkeit bieten, vor der Ausführung des Makros die Arbeitsmappe zu speichern. Diese Aufgabe wird in Schritt 2 erledigt. Sie lassen hierfür ein Meldungsfeld anzeigen, das fragt, ob die Arbeitsmappe gespeichert werden soll. Das Meldungsfeld bietet die Schaltflächen JA, NEIN und ABBRECHEN an. Wenn auf JA geklickt wird, wird die Arbeitsmappe gespeichert und die Makroausführung fortgesetzt. Ein Klick auf ABBRECHEN beendet die Prozedur, ohne das Makro weiter auszuführen. Die Makroausführung wird ohne voriges Speichern fortgesetzt, wenn auf NEIN geklickt wird.

3. In Schritt 3 wird der Zielbereich der Variablen `MyRange` zugewiesen. In diesem Beispiel verwenden Sie die aktuelle Auswahl auf dem Tabellenblatt. Sie können `MyRange` auch auf einen bestimmten Bereich setzen, wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich")`.
4. In Schritt 4 wird die Schleife gestartet und bei jedem Schleifendurchlauf eine Zelle des Zielbereichs aktiviert.
5. Nach der Aktivierung der Zelle verwendet das Makro die Funktion `IsEmpty`, um zu prüfen, ob die Zelle leer ist oder nicht. Sie verbessern so ein wenig die Ausführungsgeschwindigkeit des Makros, da leere Zellen übersprungen werden. Wenn die Zelle nicht leer ist, setzen Sie die Zelle auf den eigenen Wert zurück. Hierdurch werden fehlerhafte Formatierungen entfernt.
6. In Schritt 6 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet, nachdem alle Zellen des Zielbereichs bearbeitet sind.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Nachstehende Minuszeichen konvertieren

Ältere und Mainframe-Systeme sind dafür bekannt, dass sie oft nachstehende Minuszeichen ausgeben. Anstatt »–142« exportieren diese Systeme »142–«. Dies kann zu einem Durcheinander in Ihren Kalkulationstabellen führen, und zwar besonders dann, wenn Sie an den Daten mathematische Operationen durchführen wollen. Das nützliche Makro in diesem Abschnitt bearbeitet einen Zielbereich und korrigiert alle nachstehenden Minuszeichen: Nach der Korrektur stehen diese vor der Zahl, wie es sich gehört.

So funktioniert es

In diesem Makro verwenden Sie zwei `Range`-Objekte. Eine der Variablen enthält den kompletten Bereich, den Sie bearbeiten wollen, die zweite nimmt die Zelle auf, die derzeit in der Schleife bearbeitet wird. Danach verwenden Sie eine `ForEach`-Schleife, um nacheinander jede Zelle des Zielbereichs zu aktivieren und zu bearbeiten. Sie nutzen hier die Funktion `CDbl`, die den Wert der Zelle in den numerischen Datentyp `Double` konvertiert. Beim Datentyp `Double` muss das Minuszeichen vor der Zahl stehen.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim MyCell As Range
```

```
'Schritt 2: Arbeitsmappe speichern, bevor Zellen geändert
' werden?
 Select Case MsgBox("Diese Aktion kann nicht " & _
 "rückgängig gemacht werden. Soll die " & _
 "Arbeitsmappe zuerst gespeichert werden?", _
 vbYesNoCancel)

 Case Is = vbYes
 ThisWorkbook.Save

 Case Is = vbCancel
 Exit Sub
 End Select


'Schritt 3: Legen Sie den Zielbereich fest
 Set MyRange = Selection


'Schritt 4: Den Bereich zellenweise in einer Schleife
' durchlaufen
 For Each MyCell In MyRange


'Schritt 5: Den Wert in den Datentyp Double konvertieren
 If IsNumeric(MyCell) Then
 MyCell = CDbl(MyCell)
 End If


'Schritt 6: Die nächste Zelle im Bereich holen
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell` nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.
2. Alle Aktionen, die Sie in einem Makro ausführen, können nicht rückgängig gemacht werden. Da Sie im Makro Daten verändern, sollten Sie dem Anwender die Möglichkeit bieten, die Arbeitsmappe vor der Ausführung des Makros zu

speichern. Diese Aufgabe wird in Schritt 2 erledigt. Sie lassen hierfür ein Meldungsfeld anzeigen, das fragt, ob die Arbeitsmappe gespeichert werden soll. Das Meldungsfeld bietet die Schaltflächen JA, NEIN und ABBRECHEN an. Wenn auf JA geklickt wird, wird die Arbeitsmappe gespeichert und die Makroausführung fortgesetzt. Ein Klick auf ABBRECHEN beendet die Prozedur, ohne das Makro weiter auszuführen. Die Makroausführung wird ohne voriges Speichern fortgesetzt, wenn auf NEIN geklickt wird.

3. In Schritt 3 wird der Zielbereich der Variablen `MyRange` zugewiesen. In unserem Beispiel wird die aktuelle Auswahl auf dem Tabellenblatt verwendet. Sie können `MyRange` auch auf einen bestimmten Bereich setzen, wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich")`.
4. In Schritt 4 wird die Schleife gestartet und bei jedem Schleifendurchlauf eine Zelle des Zielbereichs aktiviert.
5. Nach der Aktivierung der Zelle verwendet das Makro die Funktion `IsNumeric`, um zu prüfen, ob es sich beim Wert in der Zelle um eine Zahl handelt. Dieser Schritt stellt sicher, dass Sie nicht versehentlich Zellen umwandeln, die Text enthalten. Anschließend übergeben Sie den Inhalt der Zelle an die Funktion `CDbl`, die den Wert in den numerischen Datentyp `Double` konvertiert. Hierdurch wird das Minuszeichen vor die Zahl gesetzt.
6. In Schritt 6 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet, nachdem alle Zellen des Zielbereichs bearbeitet sind.

Da Sie den Zielbereich auf die aktuelle Auswahl festlegen, müssen Sie den Bereich mit den Daten markieren, bevor Sie das Makro starten. Wählen Sie nicht das gesamte Tabellenblatt aus, da ansonsten jede leere Zelle mit einer Null

gefüllt wird. Um dieses Problem zu vermeiden, sollten Sie den Zielbereich exakt angeben, beispielsweise mit `Set MyRange = Range("A1:Z100")`.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Führende und nachstehende Leerzeichen aller Zellen in einem Bereich entfernen

Ein häufiges Problem beim Import von Daten aus anderen Quellen sind führende oder nachstehende Leerzeichen. Die importierten Werte besitzen dann am Anfang oder Ende der Zelle überflüssige Leerzeichen. Es liegt auf der Hand, dass diese zusätzlichen Leerzeichen die Sortierung und die Verwendung von Funktionen wie beispielsweise SVERWEIS erschweren. Hier ist ein Makro, das überflüssige Leerzeichen in Zellen findet und entfernt.

So funktioniert es

In diesem Makro durchlaufen Sie den Zielbereich in einer Schleife und schicken den Inhalt jeder Zelle durch die Funktion `Trim`.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim MyCell As Range


'Schritt 2: Arbeitsmappe speichern, bevor Zellen geändert
' werden?
 Select Case MsgBox("Diese Aktion kann nicht " & _
 "rückgängig gemacht werden. Soll die " & _
 "Arbeitsmappe zuerst gespeichert werden?", _
 vbYesNoCancel)

 Case Is = vbYes
 ThisWorkbook.Save

 Case Is = vbCancel
 Exit Sub
 End Select


'Schritt 3: Legen Sie den Zielbereich fest
 Set MyRange = Selection


'Schritt 4: Den Zielbereich in einer Schleife durchlaufen
 For Each MyCell In MyRange


'Schritt 5: Die Leerzeichen entfernen
 If Not IsEmpty (MyCell) Then
 MyCell = Trim(MyCell)
 End If


'Schritt 6: Mit der nächsten Zelle des Bereichs
' weitermachen
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell` nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.

2. Alle Aktionen, die Sie in einem Makro ausführen, können nicht rückgängig gemacht werden. Da Sie im Makro Daten verändern, sollten Sie dem Anwender oder sich selbst vor der Ausführung des Makros die Möglichkeit bieten, die Arbeitsmappe zu speichern. Diese Aufgabe wird in Schritt 2 erledigt. Sie lassen hierfür ein Meldungsfeld anzeigen, das fragt, ob die Arbeitsmappe gespeichert werden soll. Das Meldungsfeld enthält die Schaltflächen JA, NEIN und ABBRECHEN. Wenn auf JA geklickt wird, wird die Arbeitsmappe gespeichert und die Makroausführung fortgesetzt. Ein Klick auf ABBRECHEN beendet die Prozedur, ohne das Makro weiter auszuführen. Die Makroausführung wird ohne voriges Speichern fortgesetzt, wenn auf NEIN geklickt wird.
3. In Schritt 3 wird der Zielbereich der Variablen `MyRange` zugewiesen. In unserem Beispiel wird die aktuelle Auswahl auf dem Tabellenblatt verwendet. Sie können die Variable `MyRange` auch auf einen bestimmten Bereich setzen, wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich")`.
4. In Schritt 4 wird die Schleife gestartet und bei jedem Schleifendurchlauf eine Zelle des Zielbereichs aktiviert.
5. Nach der Aktivierung der Zelle verwendet das Makro die Funktion `IsEmpty`, um zu prüfen, ob die Zelle leer ist oder nicht. Hierdurch werden leere Zellen übersprungen und die Ausführungsgeschwindigkeit des Makros wird ein bisschen besser. Anschließend übergeben Sie den Wert der Zelle an die Excel-Funktion `Trim`. Die Funktion `Trim` entfernt führende und nachstehende Leerzeichen.
6. In Schritt 6 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet, nachdem alle Zellen des Zielbereichs bearbeitet sind.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

US-Postleitzahlen auf die ersten fünf Stellen kürzen

Die US-amerikanischen Postleitzahlen bestehen entweder aus fünf oder zehn Stellen. Einige Systeme geben die Postleitzahl mit zehn Stellen aus, was in vielen Fällen für eine Analyse in Excel nicht nötig ist. Daher ist es üblich, die Postleitzahl auf die ersten fünf Stellen zu kürzen. Man kann diese Aufgabe mithilfe von Formeln lösen. Falls Sie jedoch ständig US-Postleitzahlen auf diese Art bereinigen müssen, können Sie die Aufgabe besser mit dem Makro dieses Abschnitts erledigen.

Auch wenn das Makro in diesem Beispiel ein ganz spezielles Problem löst, so können Sie das Konzept des Kürzens von Daten auch in vielen anderen Bereichen einsetzen.

So funktioniert es

Das Makro verwendet die Funktion `Left`, um die ersten fünf Zeichen jeder Postleitzahl im angegebenen Zielbereich zu extrahieren:

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
```

```
 Dim MyCell As Range


'Schritt 2: Arbeitsmappe speichern, bevor Zellen geändert
' werden?
 Select Case MsgBox("Diese Aktion kann nicht " & _
 "rückgängig gemacht werden. Soll die " & _
 "Arbeitsmappe zuerst gespeichert werden?", _
 vbYesNoCancel)

 Case Is = vbYes
 ThisWorkbook.Save

 Case Is = vbCancel
 Exit Sub
 End Select


'Schritt 3: Legen Sie den Zielbereich fest
 Set MyRange = Selection


'Schritt 4: Den Zielbereich in einer Schleife durchlaufen
 For Each MyCell In MyRange


'Schritt 5: Extrahieren Sie die ersten fünf Zeichen
 If Not IsEmpty(MyCell) Then
 MyCell = Left(MyCell, 5)
 End If


'Schritt 6: Machen Sie mit der nächsten Zelle im Zielbereich weiter
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell` nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.
2. Alle Aktionen, die Sie in einem Makro ausführen, können nicht rückgängig gemacht werden. Da Sie im Makro Daten verändern, sollten Sie dem Anwender die Möglichkeit bieten, die Arbeitsmappe vor der Ausführung des Makros zu

speichern. Diese Aufgabe wird in Schritt 2 erledigt. Sie lassen hierfür ein Meldungsfeld anzeigen, das fragt, ob die Arbeitsmappe gespeichert werden soll. Das Meldungsfeld zeigt die Schaltflächen JA, NEIN und ABBRECHEN an. Wenn auf JA geklickt wird, wird die Arbeitsmappe gespeichert und die Makroausführung fortgesetzt. Ein Klick auf ABBRECHEN beendet die Prozedur, ohne das Makro weiter auszuführen. Die Makroausführung wird ohne voriges Speichern fortgesetzt, wenn auf NEIN geklickt wird.

3. In Schritt 3 wird der Zielbereich der Variablen `MyRange` zugewiesen. In unserem Beispiel wird die aktuelle Auswahl auf dem Tabellenblatt verwendet. Sie können `MyRange` auch auf einen bestimmten Bereich setzen, wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich")`.
4. In Schritt 4 wird die Schleife gestartet und bei jedem Schleifendurchlauf eine Zelle des Zielbereichs aktiviert.
5. Nach der Aktivierung der Zelle verwendet das Makro die Funktion `IsEmpty`, um zu prüfen, ob die Zelle leer ist oder nicht. Hierdurch werden leere Zellen übersprungen und die Ausführungsgeschwindigkeit des Makros steigt ein wenig. Anschließend übergeben Sie den Wert der Zelle an die Excel-Funktion `Left`, mit der Sie die ersten *x* Zeichen einer Zeichenkette abschneiden können. In diesem Szenario benötigen Sie die ersten fünf Zeichen, um die Postleitzahlen auf fünf Stellen zu kürzen, es ist hier also $x = 5$.
6. In Schritt 6 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet, nachdem alle Zellen des Zielbereichs bearbeitet sind.

Wie Sie vielleicht vermuten, können Sie auch die Excel-Funktion `Right` verwenden, um die letzten *x* Zeichen einer Zeichenfolge zu extrahieren. Es ist beispielsweise nicht

unüblich, dass die ersten Zeichen einer Produktnummer eine bestimmte übergeordnete Bedeutung haben und dass nur die letzten Zeichen Informationen über das eigentliche Produkt enthalten, wie beispielsweise in 100-4567. Sie können aus dieser Angabe die eigentliche vierstellige Produktnummer mit `Right(Produktnummer,4)` extrahieren.

Da Sie den Zielbereich auf die aktuelle Auswahl festlegen, müssen Sie den Bereich mit den Daten markieren, bevor Sie das Makro starten. Anders gesagt: Wählen Sie keine Zellen aus, deren Daten nicht zu der Logik passen, die im Makro verwendet wird. Andernfalls werden alle Zellen gekürzt, ob dies nun beabsichtigt ist oder nicht. Sie können natürlich zur Sicherheit den Zielbereich auch direkt angeben, beispielsweise mit `Set MyRange = Range("A1:Z100").`

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Zellen mit führenden Nullen versehen

In vielen Systemen müssen Kennungen, wie Kundennummer, Bestellnummer oder Produktnummer, eine bestimmte Anzahl von Zeichen aufweisen. Ein typisches Beispiel ist eine

Kundennummer im Format 0000045478. Um eine Zelle mit einer vorgegebenen festen Länge zu erhalten, werden den bereits vorhandenen Ziffern führende Nullen vorangestellt.

Dies in Excel von Hand zu machen, ist ziemlich aufwändig. Mit einem Makro hingegen ist das Auffüllen mit führenden Nullen ein Kinderspiel.

Einige Excel-Experten könnten jetzt erwidern, dass man hierfür auch ein benutzerdefiniertes Zahlenformat verwenden könnte. Hierzu rufen Sie das Dialogfeld ZELLEN FORMATIEREN auf, wählen auf der Registerkarte ZAHLEN die Kategorie BENUTZERDEFINIERT aus und geben als Format `0000000000` ein.

Das Problem bei dieser Lösung ist, dass hierdurch nur die *Darstellung* des Zellwerts geändert wird. Ein schneller Blick auf die Bearbeitungsleiste von Excel zeigt, dass die Zelle weiterhin den ursprünglichen Wert enthält und er nicht in einen Text mit der vorgeschriebenen Länge umgewandelt wurde. Falls Sie die Daten in eine Nicht-Excel-Tabelle kopieren, geht so ein rein kosmetisches Auffüllen mit Nullen verloren.

So funktioniert es

Angenommen, alle Kundennummern sollen zehn Stellen lang sein. Sie müssen dann den einzelnen Kundennummern so viele Nullen voranstellen, dass sie nach dem Bearbeiten zehn Zeichen enthalten. Das ist genau das, was dieses Makro macht.

Beachten Sie, dass Sie die Logik in Schritt 5 so anpassen müssen, dass die Anzahl der Stellen dem entspricht, was Sie benötigen.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim MyCell As Range
```

```
'Schritt 2: Arbeitsmappe speichern, bevor Zellen geändert
' werden?
 Select Case MsgBox("Diese Aktion kann nicht " & _
 "rückgängig gemacht werden. Soll die " & _
 "Arbeitsmappe zuerst gespeichert werden?", _
 vbYesNoCancel)

 Case Is = vbYes
 ThisWorkbook.Save

 Case Is = vbCancel
 Exit Sub
 End Select


'Schritt 3: Definieren Sie den Zielbereich
 Set MyRange = Selection


'Schritt 4: Den ausgewählten Bereich zellenweise
' durchlaufen
 For Each MyCell In MyRange


'Schritt 5: Dem aktuellen Inhalt der Zelle zehn Nullen
' voranstellen und dann die 10 Zeichen rechts
' verwenden
 If Not IsEmpty(MyCell) Then
 MyCell.NumberFormat = "@"
 MyCell = "0000000000" & MyCell
 MyCell = Right(MyCell, 10)
 End If


'Schritt 6: Mit der nächsten Zelle weitermachen
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell` nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.

2. Alle Aktionen, die Sie in einem Makro ausführen, können nicht rückgängig gemacht werden. Da Sie im Makro Daten verändern, sollten Sie dem Anwender die Möglichkeit bieten, die Arbeitsmappe vor der Ausführung des Makros zu speichern. Diese Aufgabe wird in Schritt 2 erledigt. Sie lassen hierfür ein Meldungsfeld anzeigen, das fragt, ob die Arbeitsmappe gespeichert werden soll. Das Meldungsfeld bietet die Schaltflächen JA, NEIN und ABBRECHEN an. Wenn auf JA geklickt wird, wird die Arbeitsmappe gespeichert und die Makroausführung fortgesetzt. Ein Klick auf ABBRECHEN beendet die Prozedur, ohne das Makro weiter auszuführen. Die Makroausführung wird ohne voriges Speichern fortgesetzt, wenn auf NEIN geklickt wird.
3. In Schritt 3 wird der Zielbereich der Variablen `MyRange` zugewiesen. In unserem Beispiel wird die aktuelle Auswahl auf dem Tabellenblatt verwendet. Sie können `MyRange` auch auf einen bestimmten Bereich setzen, wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich")`.
4. In Schritt 4 wird die Schleife gestartet und bei jedem Schleifendurchlauf eine Zelle des Zielbereichs aktiviert.
5. Nach der Aktivierung der Zelle verwendet das Makro die Funktion `IsEmpty`, um zu prüfen, ob die Zelle leer ist oder nicht. Hierdurch werden leere Zellen übersprungen und die Ausführungsgeschwindigkeit des Makros wird besser.

   Anschließend stellt das Makro sicher, dass die Zelle als Text formatiert ist. Der Grund hierfür ist, dass eine als Zahl formatierte Zelle keine führenden Nullen enthalten kann – Excel würde diese automatisch entfernen. In der nächsten Zeile verwenden Sie die Eigenschaft `NumberFormat` und geben an, dass als Format `@` verwendet werden soll. Der »Klammeraffe« bewirkt, dass die Zelle als Text formatiert wird.

   Anschließend werden dem aktuellen Inhalt der Zelle zehn Nullen vorangestellt. Sie geben hierfür im Code einfach zehn

Nullen ein und verwenden das Kaufmanns-Und (`&`), um diese mit dem Zellinhalt zu kombinieren.

Abschließend wird in Schritt 5 die Funktion `Right` verwendet, um die zehn am weitesten rechts stehenden Zeichen zu extrahieren. Wir erhalten so den gewünschten Zellwert, der inklusive der führenden Nullen exakt eine Länge von zehn Zeichen hat.

6. In Schritt 6 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet, nachdem alle Zellen des Zielbereichs bearbeitet sind.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Leere Zellen durch einen Wert ersetzen

Bei der Analyse von Daten in Excel können leere Zellen zu unterschiedlichsten Problemen führen. Das Sortieren kann hierdurch nicht mehr funktionieren, das automatische Ausfüllen wird beeinträchtigt und bei Pivot-Tabellen können leere Zellen bewirken, dass anstelle der Funktion ANZAHL die Funktion SUMME verwendet wird, und so weiter.

Leere Zellen sind nicht an sich schlecht, aber wenn sie Probleme verursachen, können Sie mit dem folgenden Makro leere Zellen in einem bestimmten Bereich durch einen Dummy-Wert ersetzen lassen, der als Kennzeichen für Leerzellen dient.

So funktioniert es

Das Makro durchläuft alle Zellen in einem festgelegten Bereich und verwendet die Funktion `Len`, um die Länge (Anzahl der Zeichen) des Werts in der aktiven Zelle zu ermitteln. Leerzellen besitzen die Länge 0. Wenn die Länge der Zelle 0 ist, fügt das Makro in diese Zelle die Zahl 0 ein und ersetzt so die leeren Zellen durch solche mit einer Null als Wert.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim MyCell As Range


'Schritt 2: Arbeitsmappe speichern, bevor Zellen geändert
' werden?
 Select Case MsgBox("Diese Aktion kann nicht " & _
 "rückgängig gemacht werden. Soll die " & _
 "Arbeitsmappe zuerst gespeichert werden?", _
 vbYesNoCancel)

 Case Is = vbYes
 ThisWorkbook.Save

 Case Is = vbCancel
 Exit Sub
 End Select


'Schritt 3: Den Zielbereich definieren
 Set MyRange = Selection


'Schritt 4: Den Zielbereich zellenweise in einer Schleife
' durchlaufen
 For Each MyCell In MyRange
```

```
'Schritt 5: Wenn die Länge null beträgt, eine 0 einfügen
 If Len(MyCell.Value) = 0 Then
 MyCell = 0
 End If


'Schritt 6: Mit der nächsten Zelle des Bereichs
' weitermachen
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell` nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.
2. Alle Aktionen, die Sie in einem Makro ausführen, können nicht rückgängig gemacht werden. Da Sie im Makro Daten verändern, sollten Sie dem Anwender die Möglichkeit bieten, die Arbeitsmappe vor der Ausführung des Makros zu speichern. Diese Aufgabe wird in Schritt 2 erledigt. Sie lassen hierfür ein Meldungsfeld anzeigen, das fragt, ob die Arbeitsmappe gespeichert werden soll. Das Meldungsfeld bietet die Schaltflächen JA, NEIN und ABBRECHEN an. Wenn auf JA geklickt wird, wird die Arbeitsmappe gespeichert und die Makroausführung fortgesetzt. Ein Klick auf ABBRECHEN beendet die Prozedur, ohne das Makro weiter auszuführen. Die Makroausführung wird ohne voriges Speichern fortgesetzt, wenn auf NEIN geklickt wird.
3. In Schritt 3 wird der Zielbereich der Variablen `MyRange` zugewiesen. In unserem Beispiel wird die aktuelle Auswahl auf dem Tabellenblatt verwendet. Sie können `MyRange` auch auf einen bestimmten Bereich setzen, wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich")`.
4. In Schritt 4 wird die Schleife gestartet und bei jedem Schleifendurchlauf eine Zelle des Zielbereichs aktiviert.

5. Nach der Aktivierung der Zelle verwendet das Makro die Funktion `IsEmpty`, um zu prüfen, ob die Zelle leer ist oder nicht. Hierdurch werden leere Zellen übersprungen und die Ausführungsgeschwindigkeit des Makros steigt. Anschließend verwenden Sie die Funktion `Len`; dies ist eine der Standardfunktionen von Excel (in der Benutzeroberfläche heißt sie auf Deutsch LÄNGE). Sie gibt die Anzahl der Zeichen einer Zeichenfolge zurück. Wenn die Zelle leer ist, gibt `Len` den Wert 0 zurück. In diesem Fall ersetzt das Makro den (fehlenden) Zellinhalt durch die Zahl 0. Sie können die Leerzellen natürlich auch durch einen anderen Wert ersetzen (»NV«, »keine Daten« oder Ähnliches).
6. In Schritt 6 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet, nachdem, alle Zellen des Zielbereichs bearbeitet sind.

Da Sie den Zielbereich auf die aktuelle Auswahl festlegen, müssen Sie den Bereich mit den Daten markieren, bevor Sie das Makro starten. Sie sollten auf keinen Fall das gesamte Tabellenblatt auswählen, da dann alle leeren Zellen des Tabellenblatts mit einer 0 gefüllt würden. Um sicherzustellen, dass dies nie ein Problem wird, können Sie den Zielbereich auch direkt angeben, beispielsweise mit `Set MyRange = Range("A1:Z100").`

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**

4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Vor oder nach dem Zellinhalt Text einfügen

Gelegentlich kann es vorkommen, dass Sie vor oder nach dem aktuellen Inhalt der Zelle weitere Daten einfügen müssen, beispielsweise um Telefonnummern durch eine Vorwahl zu ergänzen. Das Makro in diesem Abschnitt verdeutlicht, wie Sie Aufgaben bei der Standardisierung von Daten automatisieren, bei denen Werte ergänzt werden müssen.

So funktioniert es

Das Makro verwendet zwei Objektvariablen des Typs `Range`, um alle Zellen in einem Zielbereich zu bearbeiten. Um alle Zellen zu bearbeiten, wird die `For-Each`-Anweisung verwendet. Nachdem eine einzelne Zellen aktiviert wurde, fügt das Makro am Anfang des Zellwerts eine Vorwahl ein.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim MyCell As Range


'Schritt 2: Arbeitsmappe speichern, bevor Zellen geändert
' werden?
 Select Case MsgBox("Diese Aktion kann nicht " & _
 "rückgängig gemacht werden. Soll die " & _
 "Arbeitsmappe zuerst gespeichert werden?", _
 vbYesNoCancel)

 Case Is = vbYes
 ThisWorkbook.Save

 Case Is = vbCancel
 Exit Sub
 End Select
```

```
'Schritt 3: Definieren Sie den Zielbereich
 Set MyRange = Selection


'Schritt 4: Den Zielbereich zellenweise in einer Schleife
' durchlaufen
 For Each MyCell In MyRange


'Schritt 5: Sicherstellen, dass Zelle nicht leer ist,
' und nur dann Vorwahl »972« ergänzen
 If Not IsEmpty(MyCell) Then
 MyCell = "(972) " & MyCell
 End If


'Schritt 6: Nächste Zelle des Zielbereichs holen
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell` nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.
2. Alle Aktionen, die Sie in einem Makro ausführen, können nicht rückgängig gemacht werden. Da Sie im Makro Daten verändern, sollten Sie dem Anwender die Möglichkeit bieten, die Arbeitsmappe vor der Ausführung des Makros zu speichern. Diese Aufgabe wird in Schritt 2 erledigt. Sie lassen hierfür ein Meldungsfeld anzeigen, das fragt, ob die Arbeitsmappe gespeichert werden soll. Das Meldungsfeld bietet die Schaltflächen JA, NEIN und ABBRECHEN an. Wenn auf JA geklickt wird, wird die Arbeitsmappe gespeichert und die Makroausführung fortgesetzt. Ein Klick auf ABBRECHEN beendet die Prozedur, ohne das Makro weiter auszuführen. Die Makroausführung wird ohne voriges Speichern fortgesetzt, wenn auf NEIN geklickt wird.

3. In Schritt 3 wird der Zielbereich der Variablen `MyRange` zugewiesen. In unserem Beispiel wird die aktuelle Auswahl auf dem Tabellenblatt verwendet. Sie können `MyRange` auch auf einen bestimmten Bereich setzen wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich")`.
4. In Schritt 4 wird die Schleife gestartet und bei jedem Schleifendurchlauf eine Zelle des Zielbereichs aktiviert.
5. Nach der Aktivierung der Zelle verwenden Sie das Kaufmanns-Und (`&`), um die Vorwahl mit dem bisherigen Wert der Zelle zu kombinieren. Falls Sie Text am Ende der Zelle einfügen wollen, ändern Sie die Anweisung so ab, dass Sie zuerst den aktuellen Zellwert nehmen, dann ein Kaufmanns-Und (`&`) und abschließend den zu ergänzenden Text eingeben, beispielsweise `MyCell = MyCell & "(972)"`.
6. In Schritt 6 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet, nachdem alle Zellen des Zielbereichs bearbeitet sind.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Nicht druckbare Zeichen entfernen

In Ihren Daten können sich gelegentlich nicht druckbare Zeichen befinden, wie Zeilenwechsel, Wagenrücklaufzeichen und geschützte Leerzeichen. Oft müssen diese Zeichen entfernt werden, bevor Sie die Daten analysieren können.

Jeder, der mehr als einen Monat mit Excel gearbeitet hat, kennt Suchen und Ersetzen. Vielleicht haben Sie auch bereits ein Makro aufgezeichnet, um den Vorgang des Suchens und Ersetzens zu automatisieren. Dabei könnten Sie auch versucht haben, die nicht druckbaren Zeichen einfach zu suchen und zu ersetzen. Das Problem ist, dass diese nicht druckbaren Zeichen meistens unsichtbar sind, weswegen sie sich mit normalen Suchen- und Ersetzen-Aktionen nur schwer aufräumen lassen. Der einfachste Weg, sie zu entfernen, ist die Verwendung von VBA-Code.

Falls Sie auch mit diesen nervigen Zeichen kämpfen, verwenden Sie das allgemeine Makro in diesem Abschnitt, um alle nicht druckbaren Zeichen zu suchen und zu entfernen.

So funktioniert es

Bei diesem Makro handelt es sich um eine einfache Suchen-und-Ersetzen-Routine. Sie verwenden die Methode `Replace` und informieren Excel, wonach gesucht wird und wodurch die Fundstellen ersetzt werden sollen. Die Syntax ist ähnlich wie die, die ein aufgezeichnetes Makro enthält, in dem der Befehl SUCHEN UND ERSETZEN verwendet wurde. Jedoch verwendet das Makro als Suchtext Zeichencodes und keinen hart-kodierten Text.

Jedes Zeichen besitzt einen eindeutigen ASCII-Code, ähnlich wie eine Seriennummer. So besitzt das kleine a beispielsweise den ASCII-Code 97. Das kleine c hat den ASCII-Code 99. Auch den unsichtbaren Zeichen sind ASCII-Codes zugeordnet:

- ✔ Das Zeichen für Zeilenschaltung hat den ASCII-Code 10.
- ✔ Das Zeichen für Wagenrücklauf hat den ASCII-Code 13.
- ✔ Das geschützte Leerzeichen hat den ASCII-Code 160.

Das Makro verwendet die Methode `Replace` und übergibt ihr den ASCII-Code der Zeichen als Suchtext. Die gefundenen Zeichen werden dann durch eine leere Zeichenfolge ersetzt.

```
Sub Makro1()

'Schritt 1: Alle Zeichen für Zeilenschaltung
' entfernen (Line Feed)
 ActiveSheet.UsedRange.Replace What:=Chr(10), _
 Replacement:=""

'Schritt 2: Alle Zeichen für Wagenrücklauf entfernen
' (Carriage Return)
 ActiveSheet.UsedRange.Replace What:=Chr(13), _
 Replacement:=""

'Schritt 3: Alle geschützten Leerzeichen entfernen
 ActiveSheet.UsedRange.Replace What:=Chr(160), _
 Replacement:=""

End Sub
```

1. In Schritt 1 wird nach dem Zeichen für Zeilenschaltung gesucht (dessen ASCII-Code ist 10) und es entfernt. Sie können das Zeichen mit dem ASCII-Code 10 angeben, indem Sie die Funktion `Chr` verwenden. Nachdem `Chr(10)` als Suchtext festgelegt wurde, wird eine leere Zeichenfolge an den Parameter `Replacement` übergeben.

   Bitte beachten Sie die Verwendung von `ActiveSheet.UsedRange`, womit Excel angewiesen wird, in allen Zellen zu suchen, die Daten enthalten. Falls erforderlich, können Sie das Objekt `UsedRange` auch durch einen konkreten Zellbereich ersetzen.
2. In Schritt 2 wird das Zeichen für Wagenrücklauf gesucht und ersetzt.

3. In Schritt 3 wird das Zeichen für geschützte Leerzeichen gesucht und ersetzt.

In diesem Makro werden nur wenige nicht druckbare Zeichen bearbeitet. Immerhin sind dies die Zeichen, die am häufigsten vorkommen. Falls Sie andere problematische Zeichen ersetzen wollen, ergänzen Sie weitere `Replace`-Methoden und geben Sie jeweils den ASCII-Code des gewünschten Zeichens ein. Geben Sie in einer Suchmaschine »nicht druckbare ASCII-Zeichen« ein, um eine Tabelle mit den verschiedenen Zeichen zu erhalten.

So verwenden Sie es

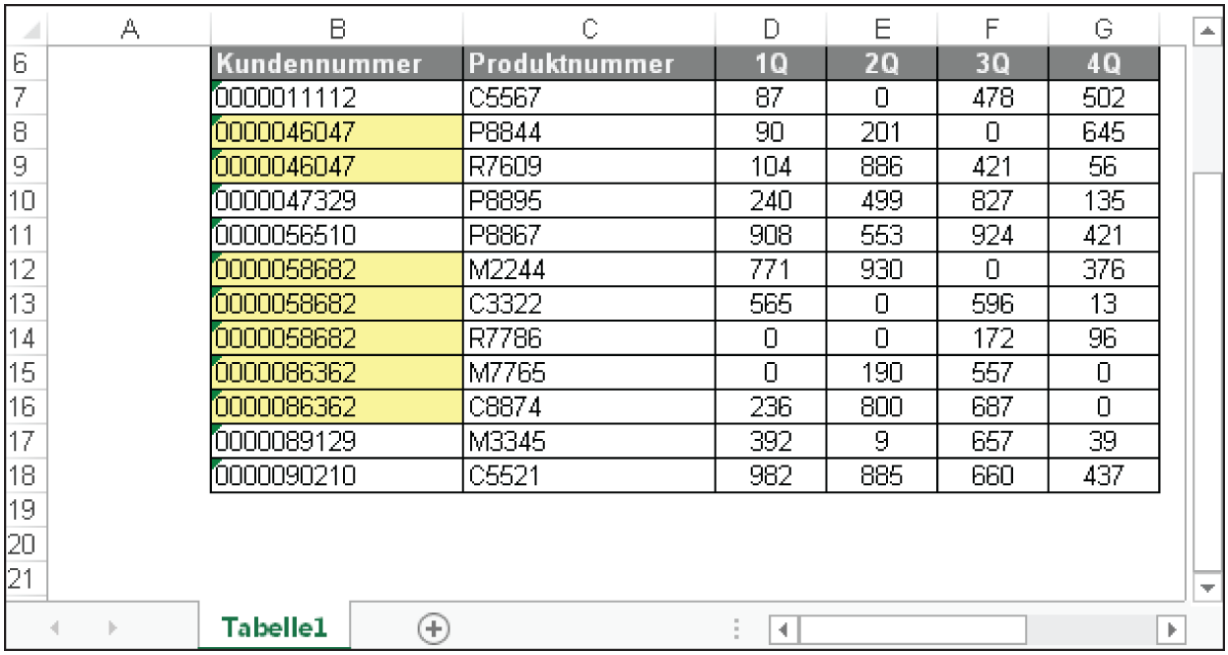
Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Duplikate in einem Datenbereich hervorheben

Mit dem Makro in diesem Abschnitt finden Sie ganz schnell Duplikate in einem Datenbereich. Es gibt auch Verfahren, um von Hand Duplikate zu finden und hervorzuheben: mit Formeln, mit bedingter Formatierung, Sortieren und so weiter. All diese Methoden müssen jedoch erst eingerichtet und, falls sich die Daten ändern, angepasst und gewartet werden.

Dieses Makro vereinfacht die Aufgabe und ermöglicht es Ihnen mit einem Mausklick, Duplikate zu finden und hervorzuheben (siehe Abbildung 7.4).



Kundennummer	Produktnummer	1Q	2Q	3Q	4Q
0000011112	C5567	87	0	478	502
0000046047	P8844	90	201	0	645
0000046047	R7609	104	886	421	56
0000047329	P8895	240	499	827	135
0000056510	P8867	908	553	924	421
0000058682	M2244	771	930	0	376
0000058682	C3322	565	0	596	13
0000058682	R7786	0	0	172	96
0000086362	M7765	0	190	557	0
0000086362	C8874	236	800	687	0
0000089129	M3345	392	9	657	39
0000090210	C5521	982	885	660	437

Abbildung 7.4: Dieses Makro sucht Duplikate in einem Bereich und heben sie hervor.

So funktioniert es

Dieses Makro durchläuft den Zielbereich in einer `For-Each`-Schleife und bearbeitet den Bereich zellenweise. Es verwendet dann die Funktion `CountIf`, um zu ermitteln, wie oft der Wert der aktiven Zelle im ausgewählten Zellbereich vorkommt. Wenn diese Zahl größer als 1 ist, wird die Zelle gelb formatiert.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim MyCell As Range

'Schritt 2: Legen Sie den Zielbereich fest
 Set MyRange = Selection

'Schritt 3: Durchlaufen Sie den Zielbereich zellenweise in
' einer Schleife
```

```
 For Each MyCell In MyRange

'Schritt 4: Mit CountIf prüfen, ob es im Zielbereich
' weitere Zellen mit dem gleichen Inhalt wie in
' der aktuellen Zelle gibt
 If WorksheetFunction.CountIf(MyRange, MyCell.Value)&gt; 1 _
 Then
 MyCell.Interior.ColorIndex = 36
 End If

'Schritt 5: Die nächste Zelle des Zielbereichs holen
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell` nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.
2. In Schritt 2 wird der Variablen `MyRange` der Zielbereich zugewiesen. In unserem Beispiel wird die aktuelle Auswahl auf dem Tabellenblatt verwendet. Sie können `MyRange` auch auf einen bestimmten Bereich setzen, wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich")`.
3. In Schritt 3 wird die Schleife gestartet und bei jedem Schleifendurchlauf eine Zelle des Zielbereichs aktiviert.
4. Das Objekt `WorksheetFunction` ermöglicht es Ihnen, in VBA viele Excel-Funktionen für Tabellenblätter zu nutzen. In Schritt 4 wird das Objekt `WorksheetFunction` verwendet, um die Funktion `CountIf` (ZÄHLENWENN) in VBA zu zu nutzen. In unserem Beispiel zählen Sie, wie oft der Wert der aktiven Zelle (`MyCell.Value`) in einem bestimmten Bereich (`MyRange`) gefunden wird. Wenn der `CountIf`-Ausdruck einen Wert liefert, der größer ist als 1, wird die Zelle gelb hinterlegt.
5. In Schritt 5 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet,

nachdem alle Zellen des Zielbereichs bearbeitet sind.

So verwenden Sie es

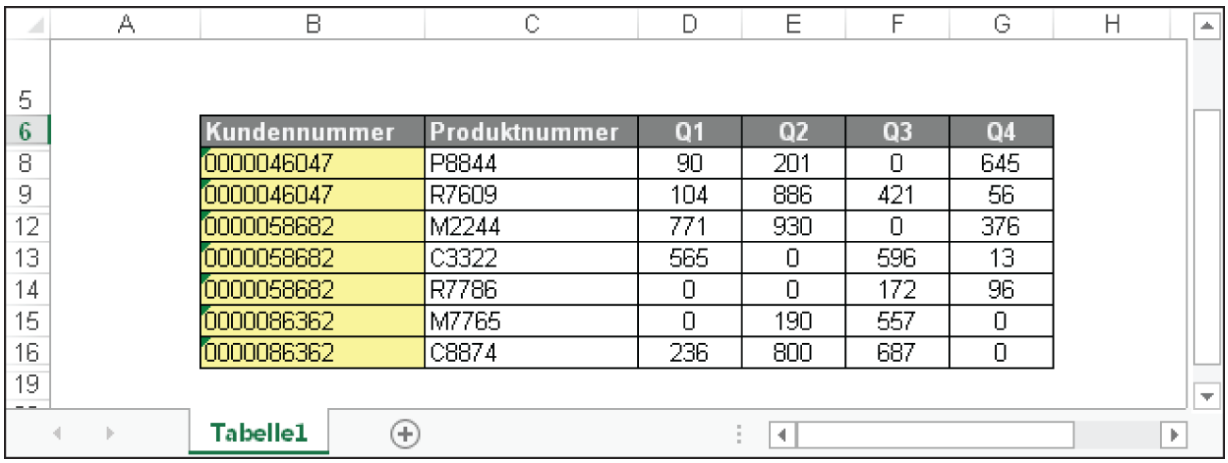
Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Alle Zeilen bis auf Zeilen mit Duplikaten ausblenden

Mit dem Makro des vorigen Abschnitts können Sie schnell Duplikate in Ihren Daten finden und hervorheben. Diese Technik ist für sich selbst schon sehr nützlich. Falls sich im Zellbereich viele Einträge befinden, wollen Sie vielleicht aber noch einen Schritt weitergehen und alle Zeilen ausblenden bis auf die, die Duplikate enthalten.

Schauen Sie sich das Beispiel in Abbildung 7.5 an. Beachten Sie, dass dort jetzt nur noch Zeilen sichtbar sind, die Duplikate enthalten.



Kundennummer	Produktnummer	Q1	Q2	Q3	Q4
0000046047	P8844	90	201	0	645
0000046047	R7609	104	886	421	56
0000058682	M2244	771	930	0	376
0000058682	C3322	565	0	596	13
0000058682	R7786	0	0	172	96
0000086362	M7765	0	190	557	0
0000086362	C8874	236	800	687	0

Abbildung 7.5: Nur Zeilen, die Duplikate enthalten, sind noch zu sehen.

So funktioniert es

Dieses Makro durchläuft den Zielbereich in einer `For-Each`-Schleife und bearbeitet den Bereich zellenweise. Sie verwenden dann die Funktion `CountIf`, um zu ermitteln, wie oft der Wert der aktiven Zelle im ausgewählten Zellbereich vorkommt. Wenn die Zahl 1 ist, blenden Sie die Zeile aus, in der sich die aktive Zelle befindet. Wenn das Ergebnis größer als 1 ist, formatieren Sie die Zelle gelb und belassen die Zeile im sichtbaren Zustand.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyRange As Range
 Dim MyCell As Range


'Schritt 2: Definieren Sie den Zielbereich
 Set MyRange = Selection


'Schritt 3: Den Zielbereich in einer Schleife bearbeiten
 For Each MyCell In MyRange


'Schritt 4: Nur nicht leere Zellen bearbeiten; Zelle
' ausblenden, wenn sie kein Duplikat enthält
 If Not IsEmpty (MyCell) Then
 If WorksheetFunction.CountIf(MyRange, MyCell)&gt; 1 Then
 MyCell.Interior.ColorIndex = 36
```

```
 MyCell.EntireRow.Hidden = False
 Else
 MyCell.EntireRow.Hidden = True
 End If
 End If


'Schritt 5: Mit der nächsten Zelle des Zielbereichs
' weitermachen
 Next MyCell

End Sub
```

1. In Schritt 1 deklarieren Sie zwei Objektvariablen des Typs `Range`. `MyRange` enthält den gesamten Zielbereich und `MyCell` nimmt in der Schleife einen Verweis auf die Zelle auf, die bearbeitet wird.
2. In Schritt 2 wird der Variablen `MyRange` der Zielbereich zugewiesen. In unserem Beispiel wird die aktuelle Auswahl auf dem Tabellenblatt verwendet. Sie können `MyRange` auch auf einen bestimmten Bereich setzen, wie beispielsweise `Range("A1:Z100")`. Falls dem Zielbereich ein Name zugewiesen wurde, können Sie auch diesen Namen verwenden: `Range("MeinBenannterBereich").`
3. In Schritt 3 wird die Schleife gestartet und bei jedem Schleifendurchlauf eine Zelle des Zielbereichs aktiviert.
4. Sie verwenden zuerst die Funktion `IsEmpty`, um zu prüfen, ob die Zelle leer ist oder nicht. Hierdurch werden leere Zellen übersprungen; dies verbessert die Ausführungsgeschwindigkeit des Makros.

   Anschließend verwenden Sie das Objekt `WorksheetFunction`, um in VBA die Funktion `CountIf` nutzen zu können. In unserem Beispiel setzen Sie `CountIf` ein, um zu prüfen, wie oft der Wert der aktiven Zelle (`MyCell.Value`) in einem bestimmten Zellbereich vorkommt (`MyRange`).

   Wenn der `CountIf`-Ausdruck einen Wert liefert, der größer ist als 1, wird der Zelle eine gelbe Hintergrundfarbe zugewiesen

und die Eigenschaft `EntireRow` auf `Hidden=False` gesetzt. Dies stellt sicher, dass die Zeile sichtbar ist.

Falls der `CountIf`-Ausdruck einen Wert ungleich 1 ergibt, macht das Makro mit dem `Else`-Zweig weiter. Hier setzen Sie die Eigenschaft `EntireRow` auf `Hidden=True`. Hierdurch wird diese Zeile ausgeblendet.

5. In Schritt 5 wird die nächste Zelle geholt und zurück an den Schleifenanfang gesprungen. Das Makro wird beendet, nachdem alle Zellen des Zielbereichs bearbeitet sind.

Da Sie den Zielbereich über die aktuelle Auswahl festlegen, müssen Sie den Bereich mit den Daten markieren, bevor Sie das Makro starten. Sie sollten beispielsweise nicht das gesamte Tabellenblatt oder eine komplette Spalte auswählen, da dann alle Zellen mit eindeutigen, nicht doppelten Werten dazu führen, dass die betreffende Zeile ausgeblendet wird. Um sicherzustellen, dass dies nie ein Problem wird, können Sie den Zielbereich auch direkt angeben, beispielsweise mit `Set MyRange = Range("A1:Z100").`

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Festlegen, in welchen Spalten AutoFilter eingesetzt werden darf

Die AutoFilter-Funktion von Excel gehört mit Sicherheit zu den nützlichsten Features, die Excel für die Datenauswertung bereitstellt. Sie erlaubt Ihnen ein extrem schnelles Filtern und Analysieren Ihrer Daten. Das einzige Problem besteht darin, dass die Standardfunktion von AutoFilter für jede Spalte des Datenbereichs eine Drop-down-Liste anzeigt, mit der Sie Daten filtern können (siehe Abbildung 7.6).

Vertriebsgebie	Q1	Q2	Q3	Q4	Produktnummer
Ost	771	930	0	376	M2244
Ost	392	9	657	39	M3345
Ost	0	190	557	0	M7765
Ost	240	499	827	135	P8895
Nord	908	553	924	421	P8867
Nord	90	201	0	645	P8844
Nord	565	0	596	13	C3322
Süd	982	885	660	437	C5521
Süd	87	0	478	502	C5567
Süd	236	800	687	0	C8874
West	0	0	172	96	R7786
West	104	886	421	56	R7609

Abbildung 7.6: Der Standard-AutoFilter fügt in jede Spalte des Datenbereichs eine Pfeilschaltfläche ein, über die Sie Daten filtern können.

Die gute Nachricht ist, dass Sie mit ein wenig VBA-Code festlegen können, in welchen Spalten mit AutoFilter gefiltert werden kann (siehe Abbildung 7.7.)

Vertriebsgebie	Q1	Q2	Q3	Q4	Produktnummer
Ost	771	930	0	376	M2244
Ost	392	9	657	39	M3345
Ost	0	190	557	0	M7765
Ost	240	499	827	135	P8895
Nord	908	553	924	421	P8867
Nord	90	201	0	645	P8844
Nord	565	0	596	13	C3322
Süd	982	885	660	437	C5521
Süd	87	0	478	502	C5567
Süd	236	800	687	0	C8874
West	0	0	172	96	R7786
West	104	886	421	56	R7609

Abbildung 7.7: Mit ein wenig VBA-Code erreichen Sie, dass nur die Spalten »Vertriebsgebiet« und »Produktnummer« zum Filtern verwendet werden können.

So funktioniert es

In VBA verwenden Sie das `AutoFilter`-Objekt, um den AutoFilter für einen bestimmten Datenbereich zu aktivieren:

```
Range("B5:G5").AutoFilter
```

Nachdem Sie den AutoFilter aktiviert haben, können Sie auf jede Spalte des AutoFilters zeigen, um sie zu konfigurieren. Um beispielsweise mit der dritten Spalte irgendeine Aktion durchzuführen, verwenden Sie folgenden Code:

```
Range("B5:G5").AutoFilter Field:=3
```

Sie können mit einem AutoFilter-Datenfeld verschiedene Aktionen durchführen. In folgendem Beispiel soll die Pfeilschaltfläche des dritten Felds ausgeblendet werden. Sie verwenden hierfür den Parameter `VisibleDropDown`. Wenn Sie diesen Parameter auf `False` setzen, blenden Sie die Pfeilschaltfläche aus.

```
Range("B5:G5").AutoFilter Field:=3, VisibleDropDown:=False
```

Hier ein Beispiel für ein Makro, das den AutoFilter einschaltet und dann lediglich in der ersten und letzten Spalte die Pfeilschaltflächen für die Drop-down-Listen sichtbar macht.

```
Sub Makro1()

 With Range("B5:G5")
 .AutoFilter
 .AutoFilter Field:=1, VisibleDropDown:=True
 .AutoFilter Field:=2, VisibleDropDown:=False
 .AutoFilter Field:=3, VisibleDropDown:=False
 .AutoFilter Field:=4, VisibleDropDown:=False
 .AutoFilter Field:=5, VisibleDropDown:=False
 .AutoFilter Field:=6, VisibleDropDown:=True
 End With
End Sub
```

In diesem Code zeigen Sie nicht nur auf einen bestimmten Datenbereich, sondern auch auf die einzelnen Felder/Spalten des AutoFilters. Wenn Sie das Makro implementieren, müssen Sie den Code ändern, damit er zu Ihrer jeweiligen Datentabelle passt.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen Sie es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Gefilterte Zeilen in eine neue Arbeitsmappe kopieren

Oft kommt es vor, dass Sie mit einer gefilterten Datentabelle arbeiten und die gefilterten Zeilen in eine neue Arbeitsmappe

kopieren möchten. Sie können dies natürlich von Hand machen und die gefilterten Zeilen kopieren, eine neue Arbeitsmappe erstellen, die Zeilen einfügen und abschließend die kopierten Zeilen so formatieren, dass die Spaltenbreite optimal ist. Falls Sie diese Schritte jedoch regelmäßig durchführen, bietet es sich an, den Vorgang mit einem Makro zu beschleunigen.

So funktioniert es

Dieses Makro kopiert den automatisch gefilterten Bereich, öffnet eine neue Arbeitsmappe und fügt dann die Daten in:

```
Sub Makro1()

'Schritt 1: Prüfen, ob AutoFilter aktiv; Prozedur beenden,
' wenn nicht
 If ActiveSheet.AutoFilterMode = False Then
 Exit Sub
 End If

'Schritt 2: Den gefilterten Bereich in eine neue
' Arbeitsmappe kopieren
 ActiveSheet.AutoFilter.Range.Copy
 Workbooks.Add.Worksheets(1).Paste

'Schritt 3: Breite der Spalten automatisch anpassen
 Cells.EntireColumn.AutoFit

End Sub
```

1. Schritt 1 verwendet die Eigenschaft `AutoFilterMode`, um zu prüfen, ob auf dem aktiven Tabellenblatt ein AutoFilter aktiv ist. Ist dies nicht der Fall, wird die Prozedur beendet.
2. Jedes `AutoFilter`-Objekt besitzt eine `Range`-Eigenschaft. Diese Eigenschaft gibt die Zeilen der Datentabelle zurück, nachdem der Filter angewendet wurde. Sie erhalten damit nur die Daten, die in der gefilterten Datentabelle sichtbar sind. In Schritt 2 werden diese Zeilen mit `Copy` kopiert und anschließend in eine neue Arbeitsmappe eingefügt. Beachten Sie, dass Sie Excel mit `Workbooks.Add.Worksheets(1)`

anweisen, die Daten in das erste Tabellenblatt der neu erstellten Arbeitsmappe einzufügen.

3. In Schritt 3 schließlich weisen Sie Excel an, die Breite der Spalten so anzupassen, dass sie zu den soeben eingefügten Daten passen.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

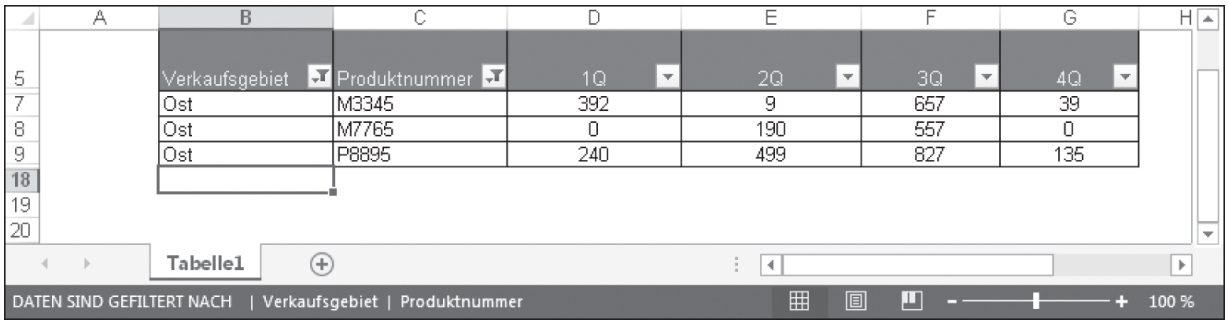
1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

In der Statusleiste die Spalten anzeigen, nach denen gefiltert wird

Wenn Sie eine große Datentabelle mit vielen Spalten haben und die AutoFilter-Funktion von Excel verwenden, kann es schwierig werden zu erkennen, nach welchen Spalten eigentlich gefiltert wurde und nach welchen nicht. Sie können zwar die Pfeilschaltflächen der Drop-down-Listen der einzelnen Spalten ansehen und dort am Symbol erkennen, dass nach ihnen gefiltert wurde, jedoch ist dies ein umständliches Verfahren.

Dieses Makro erledigt die Aufgabe für Sie und zeigt in der Statusleiste die Namen der Spalten an, nach denen die Tabelle

gefiltert wurde. Wie die Excel-Statusleiste mit diesem Makro aussieht, zeigt Abbildung 7.8.



Verkaufsgebiet	Produktnummer	1Q	2Q	3Q	4Q
Ost	M3345	392	9	657	39
Ost	M7765	0	190	557	0
Ost	P8895	240	499	827	135

Abbildung 7.8: Dieses Makro zeigt alle gefilterten Spalten in der Statusleiste an.

So funktioniert es

Dieses Makro durchläuft alle Spalten einer Datentabelle, für die ein AutoFilter aktiv ist. Es prüft in der Schleife, ob nach der aktuellen Spalte gefiltert wurde. Ist dies der Fall, speichert es den Namen der Spalte in einer Zeichenfolge. Nachdem alle Spalten bearbeitet wurden, wird die im Makro erstellte Zeichenfolge in der Statusleiste angezeigt.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim AF As AutoFilter
 Dim TargetField As String
 Dim strOutput As String
 Dim i As Integer

'Schritt 2: Prüfen, ob AutoFilter vorhanden; wenn nicht,
' Prozedur beenden
 If ActiveSheet.AutoFilterMode = False Then
 Application.StatusBar = False
 Exit Sub
 End If


'Schritt 3: AutoFilter-Variable zuweisen und Start
' der Schleife
 Set AF = ActiveSheet.AutoFilter
 For i = 1 To AF.Filters.Count
```

```
'Schritt 4: Namen der Spalten ermitteln, nach denen
' gefiltert wurde
 If AF.Filters(i).On Then
 TargetField = AF.Range.Cells(1, i).Value
 strOutput = strOutput & " | " & TargetField
 End If
 Next


'Schritt 5: Filter anzeigen, wenn vorhanden
 If strOutput = "" Then
 Application.StatusBar = False
 Else
 Application.StatusBar = "DATEN SIND GEFILTERT NACH " _
 & strOutput
 End If

End Sub
```

1. In Schritt 1 werden vier Variablen deklariert. `AF` ist eine Variable des Typs `AutoFilter`, über die Sie auf das AutoFilter-Objekt zugreifen können. `TargetField` ist eine Variable des Typs `String`, in welcher der Name der aktuellen Spalte eingefügt wird, falls nach ihr gefiltert wurde. `strOutput` ist eine weitere Variable des Typs `String`, in welcher der Text zusammengebaut wird, der in der Statusleiste erscheinen soll. Die Variable `i` schließlich dient als einfacher Zähler, mit dem Sie die einzelnen Felder des `AutoFilter`-Objekts durchlaufen.
2. Schritt 2 verwendet die Eigenschaft `AutoFilterMode`, um zu prüfen, ob auf dem aktiven Tabellenblatt ein AutoFilter aktiv ist. Ist dies nicht der Fall, setzen Sie die Eigenschaft `StatusBar` auf `False`; hierdurch wird eine benutzerdefinierte Statusleiste gelöscht und die Kontrolle über die Statusleiste wieder an Excel übertragen. Anschließend wird die Prozedur beendet.
3. Schritt 3 setzt die Variable `AF` auf den AutoFilter des aktiven Tabellenblatts. Sie stellen dann den Schleifenzähler ein; er soll bei 1 beginnen; der maximale Wert ist die Anzahl der Spalten des Bereichs, auf den der AutoFilter gelegt wurde. Auf die einzelnen Spalten des AutoFilter-Objekts kann über einen Index zugegriffen werden. Spalte 1 entspricht Index 1, Spalte

2 entspricht Index 2 und so weiter. Sie können also alle Spalten des AutoFilters in einer Schleife bearbeiten, indem Sie die Variable `i` für den Indexzugriff verwenden.

4. In Schritt 4 wird der Status des Objekts `AF.Filters` für jedes `(i)` geprüft, wobei `i` die Spalte angibt, die Sie aktuell bearbeiten. Wenn nach dieser Spalte gefiltert wurde, dann ist der Statuswert dieser Spalte `On`.

   Falls nach dieser Spalte gefiltert wurde, ermitteln Sie den Namen der Spalte und weisen ihn der Variablen `TargetField` zu. Sie erhalten den Namen, indem Sie auf die `Range`-Eigenschaft Ihres AutoFilter-Objekts `AF` zugreifen. Mit diesem Bereich können Sie das Elements `Cells` verwenden, um den Feldnamen zu ermitteln. `Cells(1,1)` liefert den Wert in Zeile 1, Spalte 1 des Bereichs. `Cells(1,2)` liefert den Wert in Zeile 1, Spalte 2 und so weiter.

   Wie Sie in Schritt 4 sehen, wurde die Zeile mit dem Wert 1 hart-kodiert und die Variable `i` für den Spaltenindex benutzt. Während das Makro die einzelnen Spalten bearbeitet, ermittelt es den Namen immer aus Zeile 1 und fügt ihn in `TargetField` ein (der Feldname befindet sich bei diesen Datentabellen immer in Zeile 1).

   Nachdem Sie den aktuellen Feldnamen ermittelt haben, übergeben Sie ihn an `strOutput`. In dieser Variablen werden alle gefundenen Feldnamen eingefügt und in ein gut lesbares Format gebracht.

5. In Schritt 5 wird geprüft, ob `strOutput` Text enthält. Falls `strOutput` leer ist, hat das Makro in der Datentabelle keine gefilterten Spalten gefunden. In diesem Fall setzt Schritt 5 die Eigenschaft `StatusBar` auf `False`, wodurch die Kontrolle über die Statusleiste an Excel zurückgegeben wird.

   Wenn `strOutput` nicht leer ist, wird in Schritt 5 die Eigenschaft `StatusBar` auf einen einleitenden Text gesetzt und an diesen der Inhalt von `strOutput` angehängt.

So verwenden Sie es

Idealerweise würden Sie das Makro immer dann ausführen, wenn nach einer Spalte gefiltert wird. Leider gibt es in Excel kein `OnAutoFilter`-Ereignis. Das Ereignis, dass Sie am ehesten verwenden können, ist das Tabellenblattereignis `Calculate`. Da AutoFilter jedoch keine Berechnungen durchführen, müssen Sie eine volatile Funktion auf das Tabellenblatt einfügen, das die gefilterte Datentabelle enthält. Eine *volatile Funktion* erzwingt eine Neuberechnung, wenn auf dem Tabellenblatt eine Änderung vorgenommen wurde.

In den Begleitdateien für dieses Buch wird in der Arbeitsmappe mit diesem Beispiel die Funktion *JETZT ()* verwendet. JETZT() ist eine volatile Funktion, die das aktuelle Datum und die aktuelle Uhrzeit zurückgibt. Wenn sich diese Funktion auf dem Tabellenblatt befindet, ist sichergestellt, dass das Tabellenblatt neu berechnet wird.

Weitere Hinweise zu den Begleitdateien für dieses Buch finden Sie in der Einleitung.

Fügen Sie die Funktion JETZT in eine beliebige Zelle des Tabellenblatts ein und tippen Sie `=JETZT()`. Kopieren Sie anschließend das Makro und fügen Sie es in die Ereignisprozedur `Worksheet_Calculate` ein:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie das Tabellenblatt an, das Sie mit dem Code verknüpfen wollen.**
4. **Wählen Sie in der Drop-down-Liste EREIGNISSE (siehe Abbildung 7.9) das Ereignis CALCULATE aus.**

5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin.**

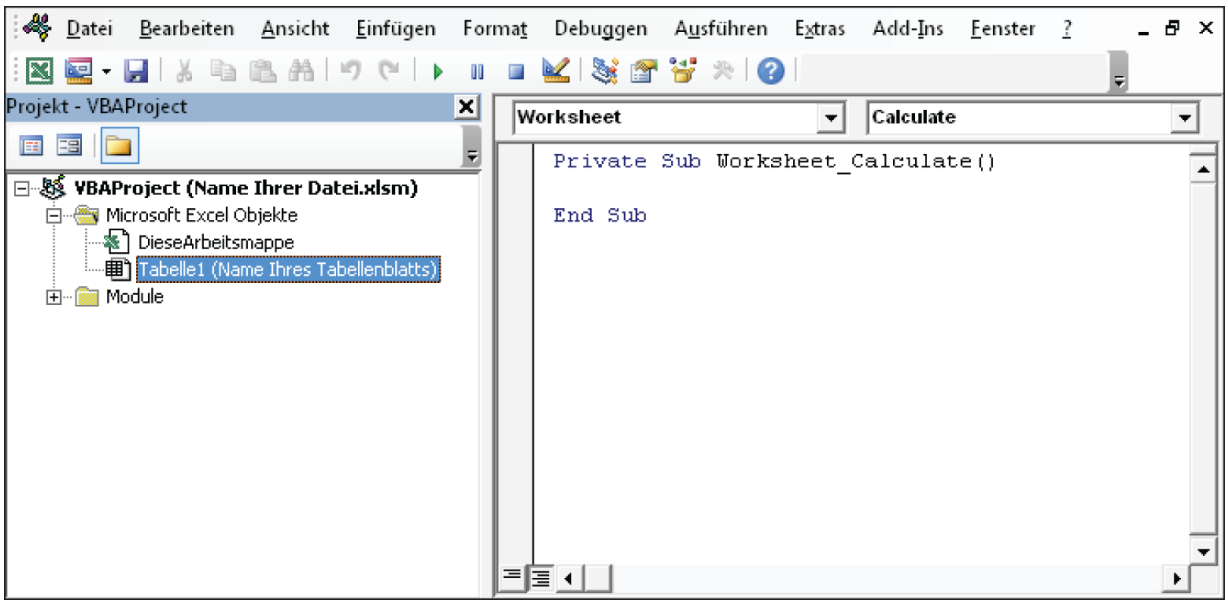


Abbildung 7.9: Tippen Sie den Code in das Tabellenblattereignis `Calculate` ein oder kopieren Sie ihn dorthin.

Damit der Code so störungsfrei wie möglich läuft, sollten Sie unterhalb von `Worksheet_Calculate` noch den folgenden Code eingeben:

```
Private Sub Worksheet_Deactivate()
 Application.StatusBar = False
End Sub

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

Geben Sie zusätzlich in die Ereignisprozedur des Arbeitsmappenereignisses `BeforeClose` noch den folgenden Code ein:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
 Application.StatusBar = False
End Sub
```

Der Code in `Worksheet_Deactivate` sorgt dafür, dass die Statusleiste zurückgesetzt wird, wenn Sie zu einem anderen

Tabellenblatt oder einer anderen Arbeitsmappe wechseln, und vermeidet so mögliche Verwirrung.

Der Code in `Worksheet_Activate` ruft `Worksheet_Calculate` auf. Hierdurch wird, wenn Sie zum gefilterten Tabellenblatt zurückkehren, in der Statusleiste wieder die Information über die Spalten angezeigt, nach denen gefiltert wurde.

Mit dem Code in der Ereignisprozedur `Workbook_BeforeClose` wird die Statusleiste auch beim Schließen der Arbeitsmappe zurückgesetzt. Dies vermeidet wiederum Verwirrungen, wenn danach eine andere Arbeitsmappe aktiviert wird.

Teil IV

Berichte, E-Mails und externe Datenquellen mit Makros steuern

IN DIESEM TEIL …

- ✔ Verstehen Sie, wie Makros bei der Automatisierung der Berichterstellung eingesetzt werden können
- ✔ Lernen Sie, wie sich die eher banalen Aspekte bei der Erstellung von Pivot-Tabellen automatisieren lassen
- ✔ Erfahren Sie, wie Makros Sie bei Diagrammen für Ihre Berichte und Dashboards unterstützen können
- ✔ Erforschen Sie einige der Techniken rund um den E-Mail-Versand aus Excel heraus
- ✔ Lernen Sie Techniken kennen, um Daten aus externen Datenbanken oder Datendateien zu bearbeiten

Kapitel 8

Die Erstellung von Berichten automatisieren

IN DIESEM KAPITEL

- Alle Pivot-Tabellen aktualisieren
- Eine Liste mit Informationen zu den Pivot-Tabellen einer Arbeitsmappe erstellen
- Titel alle Datenfelder anpassen und Felder alphabetisch sortieren
- Für alle Datenelemente Summe verwenden oder alle formatieren
- PivotTable-Felder-Beschränkungen anwenden
- Drilldown-Tabellenblätter automatisch entfernen
- Eine Pivot-Tabelle für jedes Element des Berichtsfilters drucken
- Für jedes Element des Berichtsfilters eine neue Arbeitsmappe erstellen
- Diagramme anpassen und ausrichten

Für diejenigen unter Ihnen, zu deren Aufgabenbereich die Erstellung von Dashboards und Berichten gehört, sind Pivot-Tabellen und Diagramme Bestandteile der täglichen Arbeit. Nur wenige haben jedoch das Bedürfnis, jeden Aspekt dieser Berichtswerkzeuge zu automatisieren. Einige dieser Funktionen eignen sich jedoch hervorragend für die Automatisierung. Dieses Kapitel stellt einige Makros vor, durch die Sie bei der Verwendung

von Pivot-Tabellen und Diagrammen Zeit sparen und damit effizienter arbeiten können.

Alle Pivot-Tabellen einer Arbeitsmappe aktualisieren

Es kommt oft vor, dass sich in einer Arbeitsmappe mehrere Pivot-Tabellen befinden. Diese Pivot-Tabellen sind häufig mit Daten verknüpft, die sich ändern, was wiederum eine Aktualisierung der Pivot-Tabellen erfordert. Wenn es bei Ihnen häufiger vorkommt, dass Sie mehrere Pivot-Tabellen in einer Arbeitsmappe von Hand aktualisieren müssen, können Sie dieses Makro verwenden und dann die Aktualisierung mit einem einzigen Mausklick vornehmen.

So funktioniert es

Es ist wichtig zu wissen, dass die Pivot-Tabellen in der Objekthierarchie von Excel den Tabellenblättern untergeordnet sind, auf denen sie sich befinden. Daher muss das folgende Makro zuerst alle Tabellenblätter der Arbeitsmappe durchlaufen und für jedes Tabellenblatt dann die auf ihm enthaltenen Pivot-Tabellen bearbeiten. Bei der Abarbeitung der inneren Schleife aktualisiert das Makro die gefundenen Pivot-Tabellen.

```
Sub Makro1()

'Schritt 1: Variablen deklarieren
 Dim ws As Worksheet
 Dim pt As PivotTable


'Schritt 2: Alle Tabellenblätter durchlaufen
 For Each ws In ThisWorkbook.Worksheets


'Schritt 3: Alle Pivot-Tabellen durchlaufen
 For Each pt In ws.PivotTables
 pt.RefreshTable
 Next pt
 Next ws
```

```
End Sub
```

1. In Schritt 1 wird zuerst die Objektvariable `ws` deklariert. Hierdurch wird ein Speicherbereich erzeugt, in dem beim Durchlaufen der Schleife das jeweils aktuell bearbeitete Tabellenblatt abgelegt wird. Die zweite Objektvariable, `pt`, wird verwendet, um einen Verweis auf die derzeit in der Schleife bearbeitete Pivot-Tabelle abzulegen.
2. In Schritt 2 wird die Schleife gestartet und Excel angewiesen, alle Tabellenblätter in der aktuellen Arbeitsmappe zu bearbeiten. Beachten Sie die Verwendung von `ThisWorkbook` anstelle von `ActiveWorkbook`. Das Objekt `ThisWorkbook` verweist auf die Arbeitsmappe, in der sich der Code befindet. Das Objekt `ActiveWorkbook` hingegen würde auf die Arbeitsmappe verweisen, die derzeit aktiv ist. Oft liefern beide Objekte die gleiche Arbeitsmappe zurück; falls jedoch die Arbeitsmappe mit dem Code nicht die aktive Arbeitsmappe ist, verweisen Sie auf unterschiedliche Objekte. Da Sie in unserem Beispiel nicht riskieren wollen, dass Pivot-Tabellen in anderen Arbeitsmappen aktualisiert werden, verwenden Sie `ThisWorkbook`.
3. In Schritt 3 werden alle Pivot-Tabellen des jeweils aktuellen Tabellenblatts bearbeitet und für jede Pivot-Tabelle die Methode `RefreshTable` aufgerufen. Nachdem alle Pivot-Tabellen aktualisiert sind, machen Sie mit dem nächsten Tabellenblatt weiter. Das Makro wird beendet, nachdem alle Tabellenblätter untersucht wurden und deren Pivot-Tabellen auf dem neuesten Stand sind.

Als alternatives Verfahren zur Aktualisierung der Pivot-Tabellen können Sie auch die Methode `ThisWorkbook.RefreshAll` verwenden. Diese Methode aktualisiert alle PivotTable-Berichte in der Arbeitsmappe, aber auch alle externen Datenbereiche. Falls die Arbeitsmappe

also Datentabellen enthält, deren Daten aus externen Quellen oder dem Web stammen, werden diese durch `RefreshAll` ebenfalls auf den neuesten Stand gebracht. Falls dies kein Problem darstellt, können Sie also auch einfach in eine Prozedur eines Standardmoduls den Ausdruck `ThisWorkbook.RefreshAll` eingeben.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Eine Liste mit Informationen zu den Pivot-Tabellen einer Arbeitsmappe erstellen

Falls Ihre Arbeitsmappe mehrere PivotTable-Berichte enthält, ist es oft nützlich, eine Liste mit allen Berichten zu erstellen, in der dann auch weitere Detailinformationen zu den Pivot-Tabellen enthalten sein können (ähnlich wie die Liste in Abbildung 8.1).

In dieser Übersicht sehen Sie die Namen der Berichte, wo sich der Bericht in der Arbeitsmappe befindet, woher die Quelldaten stammen und welchen Pivot-Cache-Index er verwendet.

Das folgende Makro erzeugt ein Verzeichnis der PivotTable-Berichte.

	A	B	C	D	E	F
1	Pivot-Name	Tabellenblatt	Location	Cache-Ind	Speicherort der Quelldaten	Anzahl der Zeilen
2	PivotTable10	Produktkategorien	$A$3:$I$11	1	[8.2 Inhaltsverzeichnis der PivotTables erstellen.xlsm]	59466
3	PivotTable9	Onlineverkäufe	$A$3:$D$11	2	[8.2 Inhaltsverzeichnis der PivotTables erstellen.xlsm]	59465
4	PivotTable11	Verkaufte Einheiten	$A$3:$I$11	3	[8.2 Inhaltsverzeichnis der PivotTables erstellen.xlsm]	59467
5	PivotTable12	Umsatz nach Jahr	$A$3:$I$41	4	[8.2 Inhaltsverzeichnis der PivotTables erstellen.xlsm]	59466

Abbildung 8.1: Verzeichnis aller PivotTable-Berichte

So funktioniert es

Wenn Sie eine `PivotTable`-Objektvariable erstellen, haben Sie Zugriff auf die Eigenschaften des PivotTable-Berichts. Hierzu gehören der Name des Berichts, das Tabellenblatt, auf dem er sich befindet, und der Cache-Index. Im folgenden Makro durchlaufen Sie in einer Schleife alle Pivot-Tabellen einer Arbeitsmappe und tragen die verschiedenen Eigenschaften der Berichte in ein neues Tabellenblatt ein.

Da in der Objekthierarchie von Excel jeder PivotTable-Bericht dem Tabellenblatt untergeordnet ist, auf dem er sich befindet, durchlaufen Sie in einer ersten Schleife zunächst alle Tabellenblätter der Arbeitsmappe und anschließend alle Pivot-Tabellen des jeweiligen Tabellenblatts.

Nehmen Sie sich einen Moment Zeit und schauen Sie sich die einzelnen Schritte des Makros an:

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim ws As Worksheet
 Dim pt As PivotTable
 Dim MyCell As Range


'Schritt 2: Neues Tabellenblatt erstellen und
' Spaltenüberschriften einfügen
 Worksheets.Add

 Range("A1:F1") = Array("Pivot-Name", "Tabellenblatt", _
 "Zellbereich", "Cache-Index", _
 "Speicherort der Quelldaten", _
 "Anzahl der Zeilen")
```

```
'Schritt 3: Die Liste mit Informationen zu den
' Pivot-Tabellen beginnt in Zelle A2
 Set MyCell = ActiveSheet.Range("A2")


'Schritt 4: Alle Tabellenblätter der Arbeitsmappe
' durchlaufen
 For Each ws In Worksheets


'Schritt 5: Informationen zu einer Pivot-Tabelle ermitteln
 For Each pt In ws.PivotTables
 MyCell.Offset(0, 0) = pt.Name
 MyCell.Offset(0, 1) = pt.Parent.Name
 MyCell.Offset(0, 2) = pt.TableRange2.Address
 MyCell.Offset(0, 3) = pt.CacheIndex
 MyCell.Offset(0, 4) = Application.ConvertFormula _
 (pt.PivotCache.SourceData, xlR1C1, xlA1)
 MyCell.Offset(0, 5) = pt.PivotCache.RecordCount


'Schritt 6: Zellmarkierung eine Zeile nach unten
 Set MyCell = MyCell.Offset(1, 0)


'Schritt 7: Alle Pivot-Tabellen auf allen Tabellenblättern
' durchlaufen
 Next pt
 Next ws


'Schritt 8: Spaltenbreite automatisch anpassen
 ActiveSheet.Cells.EntireColumn.AutoFit

End Sub
```

1. In Schritt 1 wird zuerst die Objektvariable `ws` deklariert. Hierdurch wird ein Speicherbereich erzeugt, in dem beim Durchlaufen der Schleife das jeweils aktuell bearbeitete Tabellenblatt abgelegt wird. Die zweite Objektvariable, `pt`, wird verwendet, um einen Verweis auf die derzeit in der Schleife bearbeitete Pivot-Tabelle abzulegen. Abschließend erzeugen Sie die Variable `MyCell` des Typs `Range`; diese verweist auf die aktuelle Zelle in dem Verzeichnis, in das Sie die Informationen zu den Pivot-Tabellen eintragen.

2. Schritt 2 erstellt ein neues Tabellenblatt und fügt die Spaltenüberschriften in den Bereich A1:F1 ein. Beachten Sie, dass Sie die Spaltenüberschriften einfach einfügen können, indem Sie dem Zellbereich ein Array mit den Zeichenfolgen der einzelnen Überschriften zuweisen. Von diesem Punkt an ist dieses Tabellenblatt das aktive Tabellenblatt.
3. Wenn Sie von Hand Daten in eine Zelle eingeben wollen, verschieben Sie die Zellmarkierung in die gewünschte Zelle. Das Gleiche passiert in Schritt 3: Dort wird `MyCell` auf Zelle A2 des aktiven Tabellenblatts gesetzt. Die ist Ihr Ausgangspunkt, von dem aus Sie die anderen Zellen erreichen.

   Im weiteren Verlauf des Makros wird die Eigenschaft `Offset` verwendet. Mit `Offset` können Sie die Zellmarkierung um *x* Zeilen und *y* Spalten, bezogen auf den vorher festgelegten Ausgangspunkt, verschieben. So bewegen Sie die Zellmarkierung mit `Range(A2).Offset(0,1)` von A2 aus um eine Spalte nach rechts. Wenn Sie die Markierung von dort aus eine Zeile nach unten bewegen wollen, verwenden Sie `Range(A2).Offset(1,0)`.

   In diesem Makro bewegen Sie sich durch das Tabellenblatt, indem Sie die Eigenschaft `Offset` von `MyCell` aus verwenden. So bewegt beispielsweise `MyCell.Offset(0,4)` die Zellmarkierung um vier Spalten nach rechts. Wenn sich die Markierung an der gewünschten Stelle befindet, geben Sie die Daten ein.
4. In Schritt 4 wird die Schleife gestartet und Excel angewiesen, alle Tabellenblätter in der aktuellen Arbeitsmappe zu bearbeiten.
5. In Schritt 5 werden alle Pivot-Tabellen des jeweils aktuellen Tabellenblatts bearbeitet und für jede gefundene Pivot-Tabelle die verschiedenen Eigenschaften abgerufen und anschließend in das Verzeichnis an der aktuellen Position der Zellmarkierung (siehe Schritt 3) eingetragen.

   Sie erfassen sechs `PivotTable`-Eigenschaften: `Name`, `Parent.Range`, `TableRange2.Address`, `CacheIndex`,

`PivotCache.SourceData` und `PivotCache.RecordCount`.

- Die Eigenschaft `Name` gibt den Namen des PivotTable-Berichts zurück.
- Über die Eigenschaft `Parent.Range` erhalten Sie den Namen des Tabellenblatts, auf dem sich der PivotTable-Bericht befindet.
- Mit `TableRange2.Address` erhalten Sie den Zellbereich, in dem sich der PivotTable-Bericht befindet.
- Mit der Eigenschaft `CacheIndex` erhalten Sie den Index des Pivot-Caches der Pivot-Tabelle. Der Pivot-Cache ist ein Speicherbereich, in dem sich alle Daten befinden, die in der Pivot-Tabelle ausgewertet werden. Wenn Sie eine neue Pivot-Tabelle erstellen, erstellt Excel einen Snapshot der Quelldaten und erzeugt einen Pivot-Cache. Jedes Mal, wenn Sie die Pivot-Tabelle aktualisieren, geht Excel zu den Quelldaten, erstellt einen weiteren Snapshot und aktualisiert den Pivot-Cache.
- Jedes Element im Pivot-Cache besitzt die Eigenschaft `SourceData`, die auf den Speicherort der Daten verweist, die zur Erstellung des Caches verwendet wurden. Wenn Sie die Pivot-Tabelle aktualisieren, werden die Daten erneut aus dem Zellbereich abgerufen, auf den `PivotCache.SourceData` zeigt.
- Sie können die Anzahl der Zeilen der Quelldaten abrufen, indem Sie die Eigenschaft `PivotCache.RecordCount` verwenden.

6. In Schritt 6 wird die Zellmarkierung mittels `MyCell` um eine Zeile nach unten bewegt, wenn eine neue Pivot-Tabelle gefunden wurde. Auf diese Weise wird für jede Pivot-Tabelle im Verzeichnis eine neue Zeile erzeugt.
7. In Schritt 7 wird, falls vorhanden, die nächste Pivot-Tabelle abgerufen. Falls das aktuelle Tabellenblatt keine weitere Pivot-Tabelle mehr enthält, wird mit dem nächsten Tabellenblatt

weitergemacht, und zwar so lange, bis alle Tabellenblätter bearbeitet wurden.

8. Schritt 8 nimmt am erstellten Verzeichnis eine kleine Formatierungsaufgabe vor: es passt die Breite der Spalten an die enthaltenen Daten an.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Die Titel aller Datenfelder der Pivot-Tabelle anpassen

Wenn Sie eine Pivot-Tabelle erstellen, versucht Excel, Sie zu unterstützen, und stellt den Beschriftungen der Datenfelder den Text *Summe von*, *Anzahl von* oder den Namen einer anderen Funktion voran, die Sie für die Zusammenfassung verwenden. Das ist nicht in allen Fällen wünschenswert. Sie möchten Beschriftungen verwenden, die den Feldern Ihrer Datenquelle so nahe wie möglich kommen. Sie können die Beschriftungen zwar auch einzeln von Hand ändern, mit diesem Makro hingegen wird diese Aufgabe in einem Rutsch erledigt.

So funktioniert es

Idealerweise stimmt der Name jedes Datenelements mit dem Feldnamen der Datenquelle überein, deren Daten in der Pivot-

Tabelle ausgewertet werden.

Leider ist es jedoch so, dass der PivotTable-Feldname mit dem Namen des Felds, wie er in der Datenquelle verwendet wird, nicht exakt übereinstimmen darf. Um diese Beschränkung zu umgehen, können Sie dem PivotTable-Feldnamen ein Leerzeichen nachstellen. Für Excel ist der Feldname mit dem Leerzeichen ein anderer Wert als der ursprüngliche Feldname aus der Datenquelle. Die Leser Ihres PivotTable-Berichts werden das nachgestellte Leerzeichen dagegen höchstwahrscheinlich nicht bemerken.

Dieses Makro verwendet diese Technik und benennt die PivotTable-Feldnamen um. In einer Schleife werden alle Datenfelder bearbeitet und dabei die Beschriftungen so geändert, dass sie dem Namen des Felds in der Datenquelle plus einem nachgestellten (geschützten) Leerzeichen entsprechen.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim pt As PivotTable
 Dim pf As PivotField


'Schritt 2: Zeigen Sie auf die Pivot-Tabelle der aktiven Zelle
 On Error Resume Next
 Set pt = _
 ActiveSheet.PivotTables(ActiveCell.PivotTable.Name)


'Schritt 3: Beenden, wenn sich die aktive Zelle nicht in
' einer Pivot-Tabelle befindet
 If pt Is Nothing Then
 MsgBox "Sie müssen die Zellmarkierung in eine " & _
 "PivotTable setzen."
 Exit Sub
 End If


'Schritt 4: Alle Pivot-Felder in einer Schleife
' durchlaufen und Titel anpassen
 For Each pf In pt.DataFields
 pf.Caption = pf.SourceName & Chr(160)
```

```
    Next pf

End Sub
```

1. In Schritt 1 werden zwei Variablen deklariert. Die Objektvariable `pt` wird verwendet, um einen Verweis auf die Pivot-Tabelle abzulegen; in `pf` befindet sich das Datenfeld, das derzeit bearbeitet wird.

   Das Makro ist so aufgebaut, dass es die Pivot-Tabelle bearbeitet, in der sich beim Makrostart die Zellmarkierung befindet. Daher muss die Zellmarkierung in einer Pivot-Tabelle stehen, wenn Sie das Makro starten.

2. In Schritt 2 wird `pt` auf den Namen der Pivot-Tabelle gesetzt, in der sich die aktive Zelle befindet. Sie verwenden hierzu die Eigenschaft `ActiveCell.PivotTable.Name` der Pivot-Tabelle, die bearbeitet werden soll.

   Wenn sich die aktive Zelle nicht in einer Pivot-Tabelle befindet, löst Excel einen Fehler aus. Um den Abbruch des Makros zu unterdrücken, verwenden Sie die Anweisung `On Error Resume Next`. Dann setzt Excel die Makroausführung auch dann fort, wenn solch ein Fehler auftritt.

3. In Schritt 3 überprüfen Sie, ob sich in der Variablen `pt` ein `PivotTable`-Objekt befindet. Wenn in der Variablen `pt` der Wert `Nothing` steht, dann befand sich die aktive Zelle nicht in einer Pivot-Tabelle; folglich konnte der Variablen auch keine Pivot-Tabelle zugewiesen werden. In diesem Fall informieren Sie den Anwender mit einer Meldung und beenden die Prozedur.

4. Wenn das Makro bei Schritt 4 ankommt, wurde eine Pivot-Tabelle gefunden, die nun bearbeitet werden kann. Das Makro verwendet eine `For-Each`-Anweisung, um alle Datenfelder zu durchlaufen. Immer wenn ein neues Datenfeld ausgewählt wird, ändert das Makro dessen Beschriftung. Hierzu wird die Eigenschaft `Caption` so geändert, dass sie der Eigenschaft `SourceName` des Felds entspricht. Die Eigenschaft `SourceName` gibt den Namen des Felds zurück, wie er in den Quelldaten enthalten ist.

Das Makro hängt an diesen Namen dann noch ein geschütztes Leerzeichen an: `Chr(160)`.

Jedes Zeichen besitzt einen eindeutigen ASCII-Code, ähnlich wie eine Seriennummer. So besitzt das kleine a beispielsweise den ASCII-Code 97. Das kleine c hat den ASCII-Code 99. Auch den unsichtbaren Zeichen ist ein ASCII-Code zugeordnet. Das geschützte Leerzeichen beispielsweise besitzt den ASCII-Code 160. Sie können diese unsichtbaren Zeichen in Ihren Makros verwenden, indem Sie den Zeichencode an die Funktion `Chr` übergeben.

Nachdem der Name geändert wurde, macht das Makro mit dem nächsten Datenfeld weiter. Das Makro wird beendet, nachdem alle Datenfelder bearbeitet wurden.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Für alle Datenelemente Summe verwenden

Wenn Sie eine neue Pivot-Tabelle erstellen, verwendet Excel für die Zusammenfassung der Daten entweder die Funktion *Summe* oder die Funktion *Anzahl*. Die Logik, die Excel für die Entscheidung verwendet, ob die Felder gezählt werden oder deren Summe gebildet wird, ist ganz einfach. Falls alle Zellen

einer Spalte numerische Werte enthalten, wird die Summe gebildet. Wenn das Feld, das Sie in die Pivot-Tabelle einfügen, ein Leerzeichen oder Text enthält, wird als Aggregatfunktion Anzahl benutzt.

Auch wenn dieses Prinzip auf den ersten Blick sinnvoll erscheint, kann es Situationen geben, in denen für ein Pivot-Tabelle-Feld, das auch leere Zellen enthält, die Summe gebildet werden muss. In diesen Fällen müssen Sie die Pivot-Tabelle von Hand bearbeiten und festlegen, dass die Werte nach der Summe zusammengefasst werden sollen. Es muss Ihnen jedoch überhaupt erst einmal auffallen, dass Excel die Anzahl und nicht die Summe berechnet. Es kommt ziemlich oft vor, dass man einfach übersieht, dass ein PivotTable-Feld gezählt und nicht addiert wird.

Das Makro in diesem Abschnitt hilft in solchen Situationen und setzt die Funktion, nach der die Werte zusammengefasst werden, für alle Datenelemente auf *Summe*.

So funktioniert es

Dieses Makro durchläuft in einer Schleife alle Datenfelder einer Pivot-Tabelle und setzt die Eigenschaft `Function` auf `xlSum`. Sie können das Makro abändern und dabei alle zur Verfügung stehenden Zusammenfassungsfunktionen verwenden: `xlCount`, `xlAverage`, `xlMin`, `xlMax` und so weiter. Wenn Sie in den Code-Editor gehen und dort `pf.Function =` eingeben, sehen Sie eine Drop-down-Liste mit allen Optionen, die Ihnen zur Verfügung stehen (siehe Abbildung 8.2).

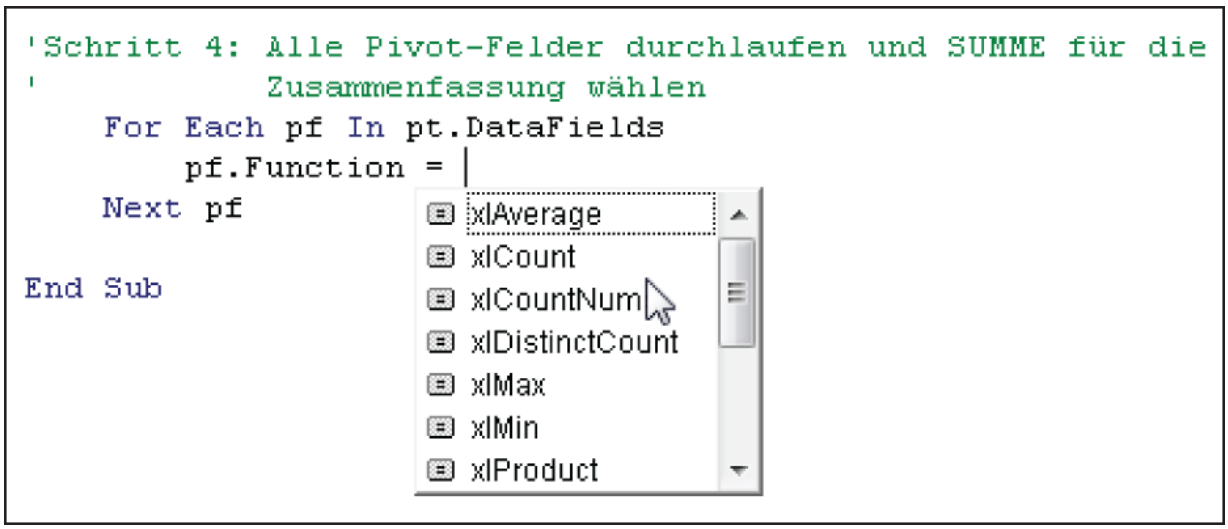


Abbildung 8.2: Excel zeigt eine Liste der Funktionen an, die Sie für die Zusammenfassung der Daten verwenden können.

```
Sub Makro1()

'Schritt1 1: Deklarieren Sie Ihre Variablen
 Dim pt As PivotTable
 Dim pf As PivotField

'Schritt 2: Zeigen Sie auf die Pivot-Tabelle der aktiven Zelle
 On Error Resume Next
 Set pt = _
 ActiveSheet.PivotTables(ActiveCell.PivotTable.Name)


'Schritt 3: Beenden, wenn sich die aktive Zelle nicht in
' einer Pivot-Tabelle befindet
 If pt Is Nothing Then
 MsgBox "Sie müssen die Zellmarkierung in eine " & _
 "Pivot-Tabelle setzen."
 Exit Sub
 End If


'Schritt 4: Alle Pivot-Felder durchlaufen und SUMME für die
' Zusammenfassung wählen
 For Each pf In pt.DataFields
 pf.Function = xlSum
 Next pf

End Sub
```

1. In Schritt 1 werden zwei Variablen deklariert. Die Objektvariable `pt` wird verwendet, um einen Verweis auf die Pivot-Tabelle abzulegen; in `pf` befindet sich das Datenfeld, das derzeit bearbeitet wird. Dieser Schritt erlaubt es Ihnen, alle Datenfelder einer Pivot-Tabelle in einer Schleife zu bearbeiten.

   Das Makro bearbeitet die Pivot-Tabelle, in der sich beim Makrostart die Zellmarkierung befindet. Daher muss die Zellmarkierung in einer Pivot-Tabelle stehen, wenn Sie das Makro starten.

2. In Schritt 2 wird `pt` auf den Namen der Pivot-Tabelle gesetzt, in der sich die aktive Zelle befindet. Sie verwenden hierzu die Eigenschaft `ActiveCell.PivotTable.Name` der Pivot-Tabelle, die bearbeitet werden soll.

   Wenn sich die aktive Zelle nicht in einer Pivot-Tabelle befindet, löst Excel einen Fehler aus. Um den Abbruch des Makros zu unterdrücken, verwenden Sie die Anweisung `On Error Resume Next`. Excel setzt die Makroausführung auch dann fort, wenn solch ein Fehler auftritt.

3. In Schritt 3 überprüfen Sie, ob sich in der Variablen `pt` ein `PivotTable`-Objekt befindet. Wenn in der Variablen `pt` der Wert `Nothing` steht, dann befand sich die aktive Zelle nicht in einer Pivot-Tabelle; folglich konnte der Variablen auch keine Pivot-Tabelle zugewiesen werden. In diesem Fall informieren Sie den Anwender mit einer Meldung und beenden die Prozedur.

4. Wenn das Makro bei Schritt 4 ankommt, wurde eine Pivot-Tabelle gefunden, die nun bearbeitet werden kann. Das Makro verwendet eine `For-Each`-Anweisung, um alle Datenfelder zu durchlaufen. Immer wenn ein neues Datenfeld ausgewählt wird, ändert das Makro die Eigenschaft `Function`, mit der die Funktion festgelegt wird, die für die Zusammenfassung dieses Felds verwendet werden soll. In unserem Beispiel soll für alle Datenfelder die Funktion *Summe* verwendet werden.

   Nachdem der Wert der Eigenschaft `Function` geändert wurde, macht das Makro mit dem nächsten Datenfeld weiter. Das

Makro wird beendet, nachdem alle Datenfelder bearbeitet wurden.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Alle Datenelemente als Zahlen formatieren

Eine Pivot-Tabelle speichert im Pivot-Cache keine Informationen zur Zahlenformatierung, da dies zusätzlichen Speicherplatz erfordert. Um so wenig Platz wie möglich zu beanspruchen, enthält der Pivot-Cache nur die Daten. Als Folge davon müssen Sie für jedes Feld, das Sie in eine Pivot-Tabelle einfügen, selbst die Zahlenformatierung vornehmen. Für jedes vorhandene Datenfeld brauchen Sie hierfür zwischen acht und zehn Mausklicks. Bei einer Pivot-Tabelle mit fünf oder mehr Datenfeldern reden wir also über mehr als 40 Mausklicks.

Idealerweise sollte eine Pivot-Tabelle in der Lage sein, sich die Quelldaten anzusehen und die dort vorhandene Formatierung zu übernehmen. Das Makro in diesem Abschnitt führt genau diese Aufgabe durch. Es erkennt die Zahlenformatierung in der Datenquelle der Pivot-Tabelle und weist den einzelnen PivotTable-Feldern diese Formatierung automatisch zu.

So funktioniert es

Bevor Sie das Makro starten, müssen Sie zwei Dinge kontrollieren:

- ✔ **Die Datenquelle der Pivot-Tabelle muss verfügbar sein.** Das Makro muss sich die Quelldaten ansehen können, anderenfalls kann es die dort vorhandene Formatierung nicht sehen.
- ✔ **Die Quelldaten sind korrekt formatiert.** Währungsfelder müssen als Währung formatiert sein, Wertefelder als Zahlen und so weiter.

Das Makro verwendet die Eigenschaft `SourceData` der Pivot-Tabelle, um den Speicherort der Quelldaten zu ermitteln. Anschließend durchläuft es in einer Schleife alle Spalten der Daten und ermittelt die Spaltenüberschrift und die Formatierung des ersten Werts unterhalb der Spaltenüberschrift. Nachdem diese Informationen vorhanden sind, überprüft das Makro, ob die aus der Spalte ermittelten Informationen zu einem beliebigen Datenfeld der Pivot-Tabelle passen. Ist dies der Fall, wird die ermittelte Zahlenformatierung auf dieses Datenfeld angewendet.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim pt As PivotTable
 Dim pf As PivotField
 Dim SrcRange As Range
 Dim strFormat As String
 Dim strLabel As String
 Dim i As Integer

'Schritt 2: Zeigen Sie auf die Pivot-Tabelle der aktiven Zelle
 On Error Resume Next
 Set pt = _
 ActiveSheet.PivotTables(ActiveCell.PivotTable.Name)


'Schritt 3: Beenden, wenn sich die aktive Zelle nicht in
' einer Pivot-Tabelle befindet
 If pt Is Nothing Then
```

```
 MsgBox "Sie müssen die Zellmarkierung in eine " & _
 "Pivot-Tabelle setzen."
 Exit Sub
 End If

'Schritt 4: Den Quellbereich ermitteln
 Set SrcRange = _
 Range(Application.ConvertFormula(pt.SourceData, _
 xlR1C1, xlA1))


'Schritt 5: Alle Spalten im Quelldatenbereich durchlaufen
 For i = 1 To SrcRange.Columns.Count


'Schritt 6: Den Namen der Spalte und das Zahlenformat
' im Quelldatenbereich ermitteln
 strLabel = SrcRange.Cells(1, i).Value
 strFormat = SrcRange.Cells(2, i).NumberFormat


'Schritt 7: Alle Felder im Datenbereich der Pivot-Tabelle
' durchlaufen
 For Each pf In pt.DataFields


'Schritt 8: Prüfen, ob SourceName mit dem vorher ermitteltem
' Spaltennamen übereinstimmt; wenn dies zutrifft,
' Zahlenformat zuweisen
 If pf.SourceName = strLabel Then
 pf.NumberFormat = strFormat
 pf.Caption = strLabel & Chr(160)
 End If
 Next pf
 Next i

End Sub
```

1. In Schritt 1 werden sechs Variablen deklariert. Die Objektvariable `pt` wird verwendet, um einen Verweis auf die Pivot-Tabelle abzulegen; in `pf` befindet sich das Datenfeld, das derzeit bearbeitet wird. In `SrcRange` befindet sich der Datenbereich der Quelldaten. Die Variablen `strFormat` und `strLabel` besitzen den Typ `String`; in ihnen werden die Spaltenüberschrift und das Zahlenformat der Quelldaten

abgelegt. Die Variable `i` dient als Zähler und hilft dabei, die einzelnen Spalten in der Datenquelle zu durchlaufen.

2. Das Makro ist so aufgebaut, dass es die Pivot-Tabelle bearbeitet, in der sich beim Makrostart die Zellmarkierung befindet. Daher muss die Zellmarkierung in einer Pivot-Tabelle stehen, wenn Sie das Makro starten.

   In Schritt 2 wird `pt` auf den Namen der Pivot-Tabelle gesetzt, in der sich die aktive Zelle befindet. Sie verwenden hierzu die Eigenschaft `ActiveCell.PivotTable.Name` der Pivot-Tabelle, die bearbeitet werden soll.

   Wenn sich die aktive Zelle nicht in einer Pivot-Tabelle befindet, löst Excel einen Fehler aus. Um den Abbruch des Makros zu unterdrücken, verwenden Sie die Anweisung `On Error Resume Next`. Dann setzt Excel die Makroausführung auch dann fort, wenn solch ein Fehler auftritt.

3. Schritt 3 überprüft, ob sich in der Variablen `pt` ein `PivotTable`-Objekt befindet. Wenn in der Variablen `pt` der Wert `Nothing` steht, dann befand sich die aktive Zelle nicht in einer Pivot-Tabelle; folglich konnte der Variablen auch keine Pivot-Tabelle zugewiesen werden. In diesem Fall informieren Sie den Anwender mit einer Meldung und beenden die Prozedur.

4. Wenn das Makro bei Schritt 4 ankommt, wurde eine Pivot-Tabelle gefunden, die nun bearbeitet werden kann. Sie weisen der Variablen `SrcRange` den Quelldatenbereich der Pivot-Tabelle zu.

   Alle Pivot-Tabellen besitzen die Eigenschaft `SourceData`, die auf die Adresse der Quelldaten zeigt. Leider wird die Adresse dort in der R1C1-Schreibweise angegeben, wie beispielsweise `'RohDaten'!R3C1:R59470C14`. Bei `Range`-Objekten können Sie die R1C1-Schreibweise nicht verwenden und Sie müssen die Adresse vor der Verwendung konvertieren: `'RohDaten'!$A$3:$N$59470`.

   Um die Konvertierung durchzuführen, übergeben Sie den Wert der Eigenschaft `SourceData` an die Funktion `Application.ConvertFormula`, die einen Zellbereich in die

R1C1-Schreibweise und auch wieder zurück in die Excel-Standardnotation konvertieren kann.

5. Nachdem Sie den Bereich zugewiesen haben, bearbeitet das Makro in einer Schleife alle Spalten der Quelldaten. Als Schleifenzähler verwenden Sie die Variable `i`, die mit 1 initialisiert wird. Die Schleife wird beendet, wenn alle Spalten der Quelldaten bearbeitet wurden (`SrcRange.Columns.Count`).
6. Beim Durchlaufen der einzelnen Spalten der Quelldaten extrahieren Sie für die jeweils aktuelle Spalte die Spaltenüberschrift und das Zahlenformat.

   Sie verwenden hierfür das `Cells`-Element. Mit dem `Cells`-Element ist es recht einfach, im Code einzelne Zellen auszuwählen. Sie brauchen lediglich die Nummer der Spalte und der Zeile anzugeben. Mit `Cells(1,1)` greifen Sie auf die Zelle in der ersten Zeile und ersten Spalte zu (was der Spaltenüberschrift der ersten Spalte entspricht). `Cells(2,1)` verweist auf die Zelle der zweiten Zeile in der ersten Spalte (womit Sie auf die erste Zelle in dieser Spalte zugreifen, die einen Wert enthält).

   Der Variablen `strLabel` wird die Spaltenüberschrift zugewiesen, die der ersten Zeile der aktuellen Spalte entnommen wird. In `strFormat` wird das Zahlenformat abgespeichert, das der zweiten Zeile der aktuellen Spalte entnommen wird.
7. Das Makro hat nun eine Verbindung zu den Quelldaten hergestellt und den Text der Spaltenbeschriftung sowie das Zahlenformat der ersten Spalte ermittelt. In Schritt 7 initialisiert das Makro eine Schleife, in der alle Datenfelder der Pivot-Tabelle untersucht werden.
8. In Schritt 8 wird jedes Datenfeld untersucht und geprüft, ob der Feldname der Quelle mit dem Text übereinstimmt, der sich in `strLabel` befindet. Ist dies der Fall, wird diesem Datenfeld die Formatierung zugewiesen, die in der betreffenden Spalte der Quelldaten gefunden und in `strFormat` abgespeichert wurde.

9. Nachdem alle Datenfelder untersucht wurden, inkrementiert das Makro den Wert von `i` und macht dann mit der nächsten Spalte der Quelldaten weiter. Das Makro wird beendet, nachdem alle Spalten bearbeitet wurden.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Alle Datenfelder alphabetisch sortieren

Wenn Sie in Ihre Pivot-Tabellen regelmäßig Daten einfügen, werden Sie bemerken, dass die neuen PivotTable-Felder nicht automatisch zwischen die bereits vorhandenen Felder einsortiert werden. Stattdessen werden sie nach den bereits vorhandenen Feldern eingefügt. Hierdurch werden neue Daten am Ende der Drop-down-Liste angezeigt, vorhandene Daten hingegen erscheinen alphabetisch sortiert.

So funktioniert es

Dieses Makro setzt die Sortierung aller Datenfelder zurück, wodurch neue Daten in die bereits vorhandenen einsortiert werden. Sie sollten das Makro daher nach jeder Aktualisierung der Pivot-Tabelle ausführen. Im Code durchlaufen Sie die

einzelnen Datenfelder der Pivot-Tabelle und führen die Sortierung durch.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim pt As PivotTable
 Dim pf As PivotField


'Schritt 2: Zeigen Sie auf die Pivot-Tabelle der aktiven Zelle
 On Error Resume Next
 Set pt = _
 ActiveSheet.PivotTables(ActiveCell.PivotTable.Name)


'Schritt 3: Beenden, wenn sich die aktive Zelle nicht in
' einer Pivot-Tabelle befindet
 If pt Is Nothing Then
 MsgBox "Sie müssen die Zellmarkierung in eine " & _
 "Pivot-Tabelle setzen."
 Exit Sub
 End If


'Schritt 4: Alle PivotTable-Felder durchlaufen und
' sortieren
 For Each pf In pt.PivotFields
 pf.AutoSort xlAscending, pf.Name
 Next pf

End Sub
```

1. In Schritt 1 werden zwei Variablen deklariert. Die Objektvariable `pt` wird verwendet, um einen Verweis auf die Pivot-Tabelle abzulegen; in `pf` befindet sich das Datenfeld, das derzeit bearbeitet wird. Dieser Schritt erlaubt es Ihnen, alle Datenfelder einer Pivot-Tabelle in einer Schleife zu bearbeiten.
2. Das Makro ist so aufgebaut, dass es die Pivot-Tabelle bearbeitet, in der sich beim Makrostart die Zellmarkierung befindet. Folglich muss die Zellmarkierung in einer Pivot-Tabelle stehen, wenn Sie das Makro starten.

In Schritt 2 wird `pt` auf den Namen der Pivot-Tabelle gesetzt, in der sich die aktive Zelle befindet. Sie verwenden hierzu die Eigenschaft `ActiveCell.PivotTable.Name` der Pivot-Tabelle, die bearbeitet werden soll.

Wenn sich die aktive Zelle nicht in einer Pivot-Tabelle befindet, löst Excel einen Fehler aus. Um den Abbruch des Makros zu unterdrücken, verwenden Sie die Anweisung `On Error Resume Next`. Excel setzt somit die Makroausführung auch dann fort, wenn solch ein Fehler auftritt.

3. In Schritt 3 überprüfen Sie, ob sich in der Variablen `pt` ein `PivotTable`-Objekt befindet. Wenn in der Variablen `pt` der Wert `Nothing` steht, dann befand sich die aktive Zelle nicht in einer Pivot-Tabelle; folglich konnte der Variablen auch keine Pivot-Tabelle zugewiesen werden. In diesem Fall informieren Sie den Anwender mit einer Meldung und beenden die Prozedur.
4. Wenn das Makro bei Schritt 4 ankommt, wurde eine Pivot-Tabelle gefunden, die nun bearbeitet werden kann. Das Makro verwendet eine `For-Each`-Anweisung, um alle Datenfelder zu durchlaufen. Immer wenn ein neues Datenfeld ausgewählt wird, benutzt das Makro die Methode `AutoSort`, um die automatischen Sortierungsregeln für dieses Feld zurückzusetzen. In unserem Beispiel sortieren Sie alle Felder in aufsteigender Reihenfolge. Nachdem alle Datenfelder bearbeitet sind, wird das Makro beendet.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**

4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Datenelemente benutzerdefiniert sortieren

Es kann gelegentlich vorkommen, dass Sie die Datenelemente nicht alphabetisch sortieren, sondern eine benutzerdefinierte Sortierreihenfolge verwenden wollen. Angenommen, Sie arbeiten für ein Unternehmen in Kalifornien und wollen, dass das Vertriebsgebiet West vor den Gebieten Nord und Süd stehen soll. In diesen Situationen ist weder die aufsteigende noch die absteigende alphabetische Sortierreihenfolge geeignet.

So funktioniert es

Sie können die benutzerdefinierte Sortierung Ihrer Datenelemente automatisieren, indem Sie die Eigenschaft `Position` des Objekts `PivotItems` verwenden. Mit der Eigenschaft `Position` legen Sie die Reihenfolge fest, in der Sie die einzelnen Pivot-Elemente sehen wollen.

Im folgenden Beispielcode zeigen Sie zuerst auf das Pivot-Feld `Region` der Pivot-Tabelle `Pvt1`. Anschließend geben Sie die verschiedenen Datenelemente und die Position an, an der sie angezeigt werden sollen.

```
Sub Makro1()

With Sheets("Tabelle1").PivotTables("Pvt1"). _
 PivotFields("Region ")
 .PivotItems("West").Position = 1
 .PivotItems("Nord").Position = 2
 .PivotItems("Süd").Position = 3
End With

End Sub
```

Die andere Lösung besteht darin, eine benutzerdefinierte Liste zu erstellen. Benutzerdefinierte Listen sind definierte Listen, die in Ihrer Instanz von Excel gespeichert sind. Um eine benutzerdefinierte Liste zu erstellen, wählen Sie DATEI | OPTIONEN | ERWEITERT und klicken auf BENUTZERDEFINIERTE LISTEN BEARBEITEN. Geben Sie in das Feld LISTENEINTRäGE **West**, **Nord** und **Süd** ein und klicken Sie auf HINZUFüGEN.

Nachdem Sie diese benutzerdefinierte Liste erstellt haben, erkennt Excel, dass es sich bei den Datenelementen des Felds *Region* um eine benutzerdefinierte Liste handelt und nimmt die Sortierung so vor, wie Sie sie festgelegt haben.

Um eine benutzerdefinierte Sortierliste zu erstellen, gehen Sie zum Dialogfeld EXCEL-OPTIONEN und wählen Sie BENUTZERDEFINIERTE LISTEN BEARBEITEN. Hier können Sie West, Nord und Süd in das Feld LISTENEINTRäGE eingeben und auf die Schaltfläche HINZUFüGEN klicken.

Nachdem Sie eine benutzerdefinierte Liste erstellt haben, können Sie ein Datenelement im Zielfeld (in diesem Fall Region) auswählen und dann im Menüband auf DATEN | SORTIEREN UND FILTERN | SORTIEREN klicken. Dadurch wird das Dialogfeld SORTIEREN geöffnet. Hier können Sie auf der Drop-down-Liste REIHENFOLGE öFFNEN und Ihre benutzerdefinierte Liste als Sortierfolge auswählen.

So brillant diese Option auch ist, so weist sie doch einen entscheidenden Nachteil auf: Benutzerdefinierte Listen werden nicht in der Arbeitsmappe, sondern in den Excel-Einstellungen gespeichert. Wenn Sie eine Arbeitsmappe gemeinsam verwenden, ist daher das obige Makro die bessere Wahl, da Sie kaum davon ausgehen können, dass Ihre Kunden oder Teamkollegen exakt die gleichen benutzerdefinierten Listen erstellt haben wie Sie.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Einschränkungen für Pivot-Tabellen festlegen

Häufig geben wir PivotTable-Berichte an Kunden, Kollegen, Manager oder andere Personen weiter. In bestimmten Fällen wollen wir die Aktionen einschränken, die andere an den Berichten vornehmen können. Dieses Makro demonstriert einige der Schutzmöglichkeiten, auf die Sie von VBA aus zugreifen können.

So funktioniert es

Das `PivotTable`-Objekt stellt Ihnen (dem Programmierer) verschiedene Eigenschaften zur Verfügung, mit denen Sie die Nutzung von bestimmten Features und Komponenten von Pivot-Tabellen einschränken können:

- ✔ **EnableWizard:** Wird diese Eigenschaft auf `False` gesetzt, werden die kontextbezogenen Registerkarten PIVOTTABLE-TOOLS deaktiviert. Diese werden von Excel normalerweise eingeblendet, wenn Sie die Zellmarkierung in eine Pivot-Tabelle setzen. In Excel 2003 deaktiviert diese Einstellung den PivotTable- und den PivotChart-Assistenten.

- **EnableDrilldown:** Wird diese Eigenschaft auf `False` gesetzt, wird das Drilldown deaktiviert. Normalerweise wird ein Drilldown aktiviert, indem Sie doppelt auf ein Datenfeld klicken; Sie erhalten dann auf einem neuen Tabellenblatt detaillierte Informationen zu den Daten.
- **EnableFiedList:** Wenn Sie diese Eigenschaft auf `False` setzen, kann die Liste mit den PivotTable-Feldern nicht mehr angezeigt und kein Pivot-Feld mehr verschoben werden.
- **EnableFieldDialog:** Wenn Sie diese Eigenschaft auf `False` setzen, können die Benutzer das Pivot-Feld nicht mehr über das Dialogfeld WERTFELDEINSTELLUNGEN konfigurieren.
- **PivotCache.EnableRefresh:** Wenn Sie diese Eigenschaft auf `False` setzen, lässt sich die Pivot-Tabelle nicht mehr aktualisieren.

Sie können all diese Eigenschaften unabhängig voneinander auf `True` oder `False` setzen. Im folgenden Makro werden alle oben beschriebenen Einschränkungen für die Pivot-Tabelle aktiviert (also die entsprechenden Optionen deaktiviert!), in der sich die Zellmarkierung befindet:

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim pt As PivotTable


'Schritt 2: Zeigen Sie auf die Pivot-Tabelle der aktiven Zelle
 On Error Resume Next
 Set pt = _
 ActiveSheet.PivotTables(ActiveCell.PivotTable.Name)


'Schritt 3: Beenden, wenn sich die aktive Zelle nicht in
' einer Pivot-Tabelle befindet
 If pt Is Nothing Then
 MsgBox "Sie müssen die Zellmarkierung in eine " & _
 "Pivot-Tabelle setzen."
 Exit Sub
 End If
```

```
'Schritt 4: Weisen Sie die Einschränkungen zu
 With pt
 .EnableWizard = False
 .EnableDrilldown = False
 .EnableFieldList = False
 .EnableFieldDialog = False
 .PivotCache.EnableRefresh = False
 End With

End Sub
```

1. In Schritt 1 wird die Objektvariable `pt` deklariert; sie wird verwendet, um einen Verweis auf die Pivot-Tabelle abzulegen.
2. In Schritt 2 wird `pt` auf den Namen der Pivot-Tabelle gesetzt, in der sich die aktive Zelle befindet. Sie verwenden hierzu die Eigenschaft `ActiveCell.PivotTable.Name` der Pivot-Tabelle, die bearbeitet werden soll.
3. In Schritt 3 wird geprüft, ob sich in der Variablen `pt` ein `PivotTable`-Objekt befindet. Steht in der Variablen `pt` der Wert `Nothing`, dann befand sich die aktive Zelle nicht in einer Pivot-Tabelle; folglich konnte der Variablen auch keine Pivot-Tabelle zugewiesen werden. In diesem Fall informieren Sie den Anwender mit einer Meldung und beenden die Prozedur.
4. Im letzten Schritt der Prozedur aktivieren Sie die sechs angesprochenen Einschränkungen für die Pivot-Tabelle.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**

4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Einschränkungen für PivotTable-Felder festlegen

Ähnlich wie Sie es im vorigen Abschnitt bei den Einschränkungen für Pivot-Tabellen gesehen haben, können Sie auch für PivotTable-Felder Einschränkungen festlegen und somit steuern, welche Aktionen der Anwender mit den verschiedenen PivotTable-Feldern vornehmen kann. Das Makro in diesem Abschnitt zeigt einige der Schutzeinstellungen, die Sie mit VBA-Code vornehmen können.

So funktioniert es

Das Objekt `PivotField` stellt Ihnen, dem Entwickler, verschiedene Eigenschaften zur Verfügung, mit denen Sie unterschiedliche Features beim Erstellen einer Pivot-Tabelle einschränken können:

- ✔ **DragToPage:** Wenn Sie diese Eigenschaft auf `False` setzen, können die Anwender keine PivotTable-Felder in den Berichtsfilterbereich einer Pivot-Tabelle ziehen.
- ✔ **DragToRow:** Wenn Sie diese Eigenschaft auf `False` setzen, können die Anwender keine PivotTable-Felder in den Zeilenbereich einer Pivot-Tabelle ziehen.
- ✔ **DragToColumn:** Wenn Sie diese Eigenschaft auf `False` setzen, können die Anwender keine PivotTable-Felder in den Spaltenbereich einer Pivot-Tabelle ziehen.
- ✔ **DragToData:** Wenn Sie diese Eigenschaft auf `False` setzen, können die Anwender keine PivotTable-Felder in den Datenbereich einer Pivot-Tabelle ziehen.
- ✔ **DragToHide:** Wenn Sie diese Eigenschaft auf `False` setzen, können die Anwender keine PivotTable-Felder mehr aus der Pivot-Tabelle ziehen, um sie auf diese Weise zu entfernen.

Außerdem können im Kontextmenü die Befehle zum Löschen und Ausblenden von PivotTable-Feldern nicht mehr ausgewählt werden.

- **EnableItemSelection:** Wenn Sie diese Eigenschaft auf `False` setzen, stehen die Drop-down-Listen in den einzelnen PivotTable-Feldern nicht mehr zur Verfügung.

Sie können all diese Eigenschaften unabhängig voneinander auf `True` oder `False` setzen. Im folgenden Makro werden alle oben beschriebenen Einschränkungen für alle PivotTable-Felder der Pivot-Tabelle aktiviert (also die entsprechenden Optionen deaktiviert!), in der sich die Zellmarkierung befindet:

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim pt As PivotTable
 Dim pf As PivotField


'Schritt 2: Zeigen Sie auf die Pivot-Tabelle der aktiven Zelle
 On Error Resume Next
 Set pt = _
 ActiveSheet.PivotTables(ActiveCell.PivotTable.Name)


'Schritt 3: Beenden, wenn sich die aktive Zelle nicht in
' einer Pivot-Tabelle befindet
 If pt Is Nothing Then
 MsgBox "Sie müssen die Zellmarkierung in eine " & _
 "Pivot-Tabelle setzen."
 Exit Sub
 End If


'Schritt 4: Legen Sie die Einschränkungen für die Pivot-
' Felder fest
 For Each pf In pt.PivotFields
 pf.EnableItemSelection = False
 pf.DragToPage = False
 pf.DragToRow = False
 pf.DragToColumn = False
 pf.DragToData = False
 pf.DragToHide = False
```

```
    Next pf

End Sub
```

1. In Schritt 1 werden zwei Objektvariablen deklariert. Die Variable `pt` wird verwendet, um einen Verweis auf die Pivot-Tabelle abzulegen. In `pf` befindet sich das PivotTable-Feld, das derzeit bearbeitet wird. Dieser Schritt erlaubt es Ihnen, alle PivotTable-Felder einer Pivot-Tabelle in einer Schleife zu bearbeiten.
2. In Schritt 2 wird `pt` auf den Namen der Pivot-Tabelle gesetzt, in der sich die aktive Zelle befindet. Sie verwenden hierzu die Eigenschaft `ActiveCell.PivotTable.Name` der Pivot-Tabelle, die bearbeitet werden soll.
3. In Schritt 3 wird geprüft, ob sich in der Variablen `pt` ein `PivotTable`-Objekt befindet. Steht in der Variablen `pt` der Wert `Nothing`, dann befand sich die aktive Zelle nicht in einer Pivot-Tabelle; folglich konnte der Variablen auch keine Pivot-Tabelle zugewiesen werden. In diesem Fall informieren Sie den Anwender mit einer Meldung und beenden die Prozedur.
4. In Schritt 4 des Makros wird die `For-Each`-Anweisung verwendet, um die einzelnen Felder der Pivot-Tabelle zu bearbeiten. Die Felder werden nacheinander ausgewählt und die oben beschriebenen Einschränkungen aktiviert.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**

4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Drilldown-Tabellenblätter automatisch entfernen

Eines der coolsten Features von PivotTable-Berichten ist die Möglichkeit, eine Zahl im Datenbereich doppelt anzuklicken, um so zu untersuchen, aus welchen Detailwerten sich die angeklickte Zahl zusammensetzt. Die Details werden auf einem neuen Tabellenblatt angezeigt. In den meisten Fällen brauchen Sie diese Tabellenblätter nicht aufzubewahren. Sie werden vielmehr zu einem Ärgernis, da es Zeit braucht, sie manuell zu löschen.

Das Ganze ist insbesondere dann ein Problem, wenn Sie Ihre PivotTable-Berichte an Benutzer weitergeben, die sich regelmäßig mit der Drilldown-Funktion Detailinformationen anzeigen lassen. Es gibt keine Garantie dafür, dass die User daran denken, die zusätzlichen Tabellenblätter wieder zu entfernen. Auch wenn diese Tabellenblätter normalerweise keine Probleme verursachen, führen sie dennoch dazu, dass die Arbeitsmappe unübersichtlicher wird.

Hier ist eine Technik, die Sie implementieren können, damit sich Ihre Arbeitsmappe automatisch darum kümmert, die Drilldown-Tabellenblätter wieder zu entfernen.

So funktioniert es

Das Grundprinzip des Makros ist recht einfach. Wenn der Benutzer doppelt klickt, um ein Drilldown-Tabellenblatt zu erzeugen, benennt das Makro das neue Tabellenblatt um und sorgt dafür, dass der Name des Blatts mit *PivotDrill* beginnt. Bevor die Arbeitsmappe geschlossen wird, sucht ein weiteres Makro nach Tabellenblättern, die mit dem Namen *PivotDrill* beginnen, und löscht diese.

Die Implementierung ist ein klein wenig aufwändiger, da Sie zwei Prozeduren erstellen müssen. Das erste Stück VBA-Code wird in die Tabellenblatt-Ereignisprozedur `WorkSheet_BeforeDoubleClick` eingefügt, das zweite in die Arbeitsmappen-Ereignisprozedur `Workbook_BeforeClose`.

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target _
 As Range, Cancel As Boolean)

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim pt As String


'Schritt 2: Beenden, wenn der Doppelklick nicht in
' einer Pivot-Tabelle erfolgte
 On Error Resume Next
 If IsEmpty(Target) And _
 ActiveCell.PivotField.Name &lt;&gt; "" Then
 Cancel = True
 Exit Sub
 End If


'Schritt 3: Das PivotTable-Objekt abrufen
 pt = ActiveSheet.Range(ActiveCell.Address).PivotTable


'Schritt 4: Wenn Drilldowns aktiviert sind,
' Drilldown-Tabellenblatt erzeugen und in dessen Namen
' »Tabelle« durch »PivotDrill« ersetzen
 If ActiveSheet.PivotTables(pt).EnableDrilldown Then
 Selection.ShowDetail = True

 ActiveSheet.Name = Replace(ActiveSheet.Name, _
 "Tabelle", "PivotDrill")
 End If

End Sub
```

1. In Schritt 1 wird eine Objektvariable für die Pivot-Tabelle deklariert.
2. In Schritt 2 wird geprüft, ob sich die Zelle, die doppelt angeklickt wurde, in einer Pivot-Tabelle befindet. Sollte dies

nicht der Fall sein, können Sie das Doppelklickereignis ignorieren.

3. Falls die angeklickte Zelle zu einer Pivot-Tabelle gehört, wird in Schritt 3 die Variable `pt` auf die Pivot-Tabelle gesetzt.
4. Schritt 4 prüft die Eigenschaft `EnableDrilldown`. Falls sie aktiviert ist, lösen Sie die Methode `ShowDetail` aus. Hierdurch werden die Details in ein neues Tabellenblatt eingefügt.

   Das Makro folgt dann der soeben erzeugten Ausgabe und verwendet die Funktion `Replace`, um das erstellte Tabellenblatt so umzubenennen, dass die ersten zehn Zeichen des Namens *PivotDrill* lauten. Die Funktion `Replace` ersetzt den angegebenen Text in einem Ausdruck durch einen anderen Text. In unserem Beispiel ersetzen Sie das Wort *Tabelle* durch *PivotDrill*, indem Sie `Replace(ActiveSheet.Name, "Tabelle", "PivotDrill")` verwenden.

   So wird aus *Tabelle1 PivotDrill1* und aus dem Tabellenblatt mit dem Namen *Tabelle12* wird *PivotDrill12* und so weiter.

   Anschließend wird die Ereignisprozedur `Workbook_BeforeClose` implementiert. Wie der Name schon sagt, wird diese Prozedur beim Schließen der Arbeitsmappe ausgeführt.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

'Schritt 5: Deklarieren Sie Ihre Variablen
 Dim ws As Worksheet


'Schritt 6: Alle Tabellenblätter durchlaufen
 For Each ws In ThisWorkbook.Worksheets


'Schritt 7: Tabellenblatt löschen, wenn der Name
' mit PivotDrill beginnt
 If Left(ws.Name, 10) = "PivotDrill" Then
 Application.DisplayAlerts = False
 ws.Delete
 Application.DisplayAlerts = True
 End If
 Next ws
```

```
End Sub
```

1. In Schritt 5 wird die `Worksheet`-Variable `ws` deklariert. Hier wird das jeweils aktuelle Tabellenblatt abgelegt, während Sie die Arbeitsmappe in einer Schleife bearbeiten.
2. Die Schleife wird in Schritt 6 gestartet; Sie weisen Excel an, alle Tabellenblätter in dieser Arbeitsmappe zu durchlaufen.
3. Im letzten (und entscheidenden) Schritt untersuchen Sie den Namen des Tabellenblatts, das derzeit in der Schleife bearbeitet wird. Falls die ersten zehn Zeichen des Namens *PivotDrill* lauten, löschen Sie das Tabellenblatt. Nachdem alle Tabellenblätter bearbeitet und alle Drilldown-Tabellenblätter gelöscht sind, wird das Makro beendet.

So verwenden Sie es

Um den ersten Teil dieses Makros zu implementieren, müssen Sie im Codefenster eine Ereignisprozedur für das Ereignis `Worksheet_BeforeDoubleClick` einfügen. Da Sie das Makro diesem Ereignis zuordnen, wird es automatisch ausgeführt, wenn auf dem Tabellenblatt ein Doppelklick ausgeführt wird.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie das Tabellenblatt an, das Sie mit dem Code verknüpfen wollen.**
4. **Wählen Sie in der Drop-down-Liste EREIGNISSE (siehe Abbildung 8.3) das Ereignis BEFOREDOUBLECLICK aus.**
5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin.**

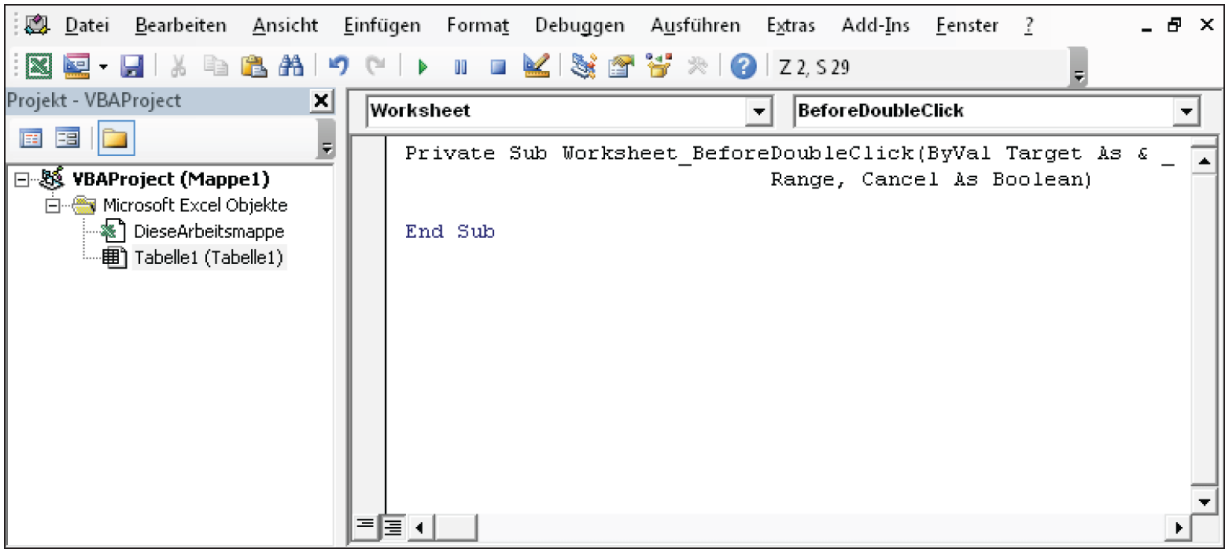


Abbildung 8.3: Tippen Sie den Code in das Tabellenblattereignis `BeforeDoubleClick` ein oder kopieren Sie ihn dorthin.

Um den zweiten Teil dieses Makros zu implementieren, müssen Sie im Codefenster eine Ereignisprozedur für das Ereignis `Workbook_BeforeClose` einfügen. Indem Sie das Makro auf diese Art und Weise erstellen, gewährleisten Sie, dass es immer dann ausgeführt wird, wenn die Arbeitsmappe geschlossen wird:

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Suchen Sie im Projekt-Explorer nach dem Namen Ihres Projekts/Ihrer Arbeitsmappe und klicken Sie das Pluszeichen an, um alle Tabellenblätter zu sehen.**
3. **Klicken Sie auf DIESEARBEITSMAPPE.**
4. **Wählen Sie in der Drop-down-Liste EREIGNISSE (siehe Abbildung 8.4) das Ereignis BEFORECLOSE aus.**
5. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn dorthin.**

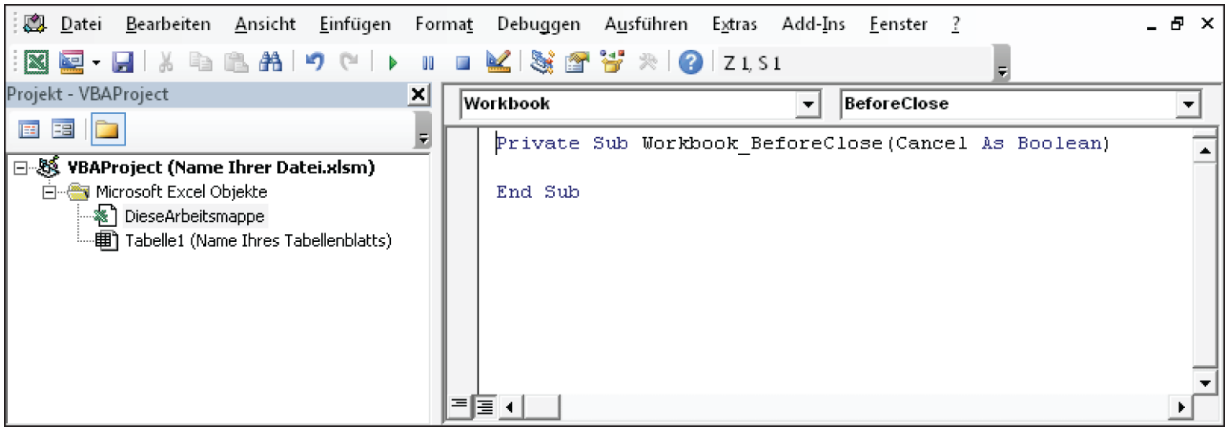


Abbildung 8.4: Geben Sie den Code in die Prozedur für das Arbeitsmappenereignis `BeforeClose` ein oder kopieren Sie ihn dorthin.

Eine Pivot-Tabelle für jedes Element des Berichtsfilters drucken

Pivot-Tabellenles stellen ein ausgezeichnetes Verfahren dar, um große Datensätze in druckbare, gefilterte Berichte aufzuteilen. Sie erstellen einen PivotTable-Bericht inklusive Analysen und Zusammenfassungen und fügen dann ein Feld wie das *Verkaufsgebiet* in den Berichtsfilter ein. Anschließend wählen Sie im Berichtsfilter nacheinander die einzelnen Elemente aus und drucken dann die PivotTable-Berichte aus.

Das Makro in diesem Abschnitt zeigt, wie Sie alle Elemente eines Berichtsfilters in einer Schleife durchlaufen und die einzelnen Berichte ausdrucken lassen.

So funktioniert es

Im Excel-Objektmodell greifen Sie über `PageField` auf die Drop-down-Liste des Berichtsfilters zu. Um für jeden Eintrag in der Drop-down-Lliste des Berichtsfilters eine Pivot-Tabelle auszudrucken, müssen Sie in einer Schleife die Sammlung `PivotItems` des `PageField`-Objekts durchlaufen. In der Schleife ändern Sie dynamisch die Auswahl im Berichtsfilter und rufen

dann die Methode `ActiveSheet.PrintOut` auf. Dann drucken Sie den ausgewählten Bereich.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim pt As PivotTable
 Dim pf As PivotField
 Dim pi As PivotItem


'Schritt 2: Zeigen Sie auf die Pivot-Tabelle der aktiven Zelle
 On Error Resume Next
 Set pt = _
 ActiveSheet.PivotTables(ActiveCell.PivotTable.Name)


'Schritt 3: Beenden, wenn sich die aktive Zelle nicht in
' einer Pivot-Tabelle befindet
 If pt Is Nothing Then
 MsgBox "Sie müssen die Zellmarkierung in eine " & _
 "Pivot-Tabelle setzen."
 Exit Sub
 End If


'Schritt 4: Beenden, wenn es mehr als einen
' Berichtsfilter gibt
 If pt.PageFields.Count&gt; 1 Then
 MsgBox "Zu viele Berichtsfilter-Felder gefunden. " & _
 "Limit ist 1."
 Exit Sub
 End If


'Schritt 5: Schleife über die Berichtsfilter und die
' dort enthaltenen Elemente
 For Each pf In pt.PageFields
 For Each pi In pf.PivotItems


'Schritt 6: Auswahl im Berichtsfilter ändern
 pt.PivotFields(pf.Name).CurrentPage = pi.Name


'Schritt 7: Druckbereich festlegen und drucken
 ActiveSheet.PrintOut Copies:=1
```

```
    Next pi
    Next pf

End Sub
```

1. Für dieses Makro werden in Schritt 1 drei Variablen deklariert. Die Variable `pt` wird verwendet, um einen Verweis auf die Pivot-Tabelle abzulegen. In `pf` befindet sich das PivotTable-Feld, das derzeit bearbeitet wird, und in `pi` das Element des Berichtsfilters, das derzeit ausgewählt ist.
2. Damit das Makro funktioniert, muss sich die aktive Zelle in einer Pivot-Tabelle befinden. Wenn sich die Zellmarkierung in einer bestimmten Pivot-Tabelle befindet, können Sie davon ausgehen, dass das Makro auf diese Pivot-Tabelle angewendet werden soll.

   In Schritt 2 wird `pt` auf den Namen der Pivot-Tabelle gesetzt, in der sich die aktive Zelle befindet. Sie verwenden hierzu die Eigenschaft `ActiveCell.PivotTable.Name` der Pivot-Tabelle, die bearbeitet werden soll.

   Wenn sich die aktive Zelle nicht in einer Pivot-Tabelle befindet, löst Excel einen Fehler aus. Sie verwenden die Anweisung `On Error Resume Next`, um Excel anzuweisen, auch beim Eintreten eines Fehlers die Makroausführung fortzusetzen.
3. In Schritt 3 wird geprüft, ob sich in der Variablen `pt` ein `PivotTable`-Objekt befindet. Steht in der Variablen `pt` der Wert `Nothing`, dann befand sich die aktive Zelle nicht in einer Pivot-Tabelle; folglich konnte der Variablen auch keine Pivot-Tabelle zugewiesen werden. In diesem Fall wird der Anwender mit einer Meldung informiert und die Prozedur beendet.
4. Schritt 4 prüft, ob die Pivot-Tabelle mehr als einen Berichtsfilter enthält. (Wenn die Eigenschaft `Count` von `PageFields` größer als 1 ist, dann ist mehr als ein Berichtsfilter vorhanden.) Diese Überprüfung führen Sie aus einem einfachen Grund aus: Wenn die Pivot-Tabelle zahlreiche Berichtsfilter enthält, kann es schnell passieren, dass Sie Hunderte von Seiten drucken. Wenn die Anzahl der

Berichtsfilter größer ist als 1, wird daher die Makroausführung beendet und eine Meldung angezeigt.

Falls Sie diese Einschränkung entfernen wollen, können Sie einfach den Code von Schritt 4 entfernen oder auskommentieren oder auch einen anderen Grenzwert als 1 eingeben.

5. In Schritt 5 werden zwei ineinander verschachtelte Schleifen gestartet. Die äußere Schleife weist Excel an, alle Berichtsfilter zu durchlaufen. In der inneren Schleife werden alle Pivot-Elemente des aktuellenBerichtsfilters durchlaufen.
6. Das Makro ruft für jedes Pivot-Element den Namen ab und verwendet ihn, um die Auswahl im Berichtsfilter zu ändern. Hierdurch wird praktisch der Filter angewendet und die Pivot-Tabelle passt sich dem Filter an.
7. In Schritt 7 wird das aktuelle Tabellenblatt gedruckt und dann mit dem nächsten Eintrag der Berichtsfilter-Drop-down-Liste weitergemacht. Nachdem Sie alle Elemente des aktuellen Berichtsfilters bearbeitet haben, macht das Makro mit dem nächsten `PageField` (Berichtsfilter) weiter. Das Makro wird beendet, nachdem alle Berichtsfilter bearbeitet wurden.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Für jedes Element des Berichtsfilters eine neue Arbeitsmappe erstellen

Pivot-Tabellen stellen ein ausgezeichnetes Verfahren dar, um große Datensätze in druckbare, gefilterte Berichte aufzuteilen. Sie erstellen einen PivotTable-Bericht inklusive Analysen und Zusammenfassungen und fügen dann ein Feld wie das *Verkaufsgebiet* in den Berichtsfilter ein. Anschließend wählen Sie im Berichtsfilter nacheinander die einzelnen Elemente aus und können dann die Daten der Pivot-Tabelle in eine neue Arbeitsmappe exportieren.

Das Makro in diesem Abschnitt zeigt, wie Sie alle Elemente eines Berichtsfilters in einer Schleife durchlaufen und die Berichte, die Sie so erhalten, in neue Arbeitsmappen exportieren.

So funktioniert es

Im Excel-Objektmodell greifen Sie über `PageField` auf die Drop-down-Liste des Berichtsfilters zu. Um für jeden Eintrag in der Drop-down-Liste des Berichtsfilters eine Kopie zu erstellen, müssen Sie in einer Schleife die Sammlung `PivotItems` des `PageField`-Objekts durchlaufen. In der Schleife ändern Sie dynamisch die Auswahl im Berichtsfilter und exportieren dann den PivotTable-Bericht in eine neue Arbeitsmappe.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim pt As PivotTable
 Dim pf As PivotField
 Dim pi As PivotItem


'Schritt 2: Zeigen Sie auf die Pivot-Tabelle der aktiven Zelle
 On Error Resume Next
 Set pt = _
 ActiveSheet.PivotTables(ActiveCell.PivotTable.Name)
```

```
'Schritt 3: Beenden, wenn sich die aktive Zelle nicht in
' einer Pivot-Tabelle befindet
 If pt Is Nothing Then
 MsgBox "Sie müssen die Zellmarkierung in eine " & _
 "Pivot-Tabelle setzen."
 Exit Sub
 End If


'Schritt 4: Beenden, wenn es mehr als einen
' Berichtsfilter gibt
 If pt.PageFields.Count&gt; 1 Then
 MsgBox "Zu viele Berichtsfilter-Felder gefunden. " & _
 "Limit ist 1."
 Exit Sub
 End If


'Schritt 5: Schleife über die Berichtsfilter und die
' dort enthaltenen Elemente
 For Each pf In pt.PageFields
 For Each pi In pf.PivotItems


'Schritt 6: Auswahl des Berichtsfilters ändern
 pt.PivotFields(pf.Name).CurrentPage = pi.Name


'Schritt 7: Den Datenbereich in eine neue
' Arbeitsmappe kopieren
 pt.TableRange1.Copy _
 Workbooks.Add.Worksheets(1).Paste
 Application.DisplayAlerts = False
 ActiveWorkbook.SaveAs _
 Filename:="C:\Temp\" & pi.Name & ".xlsx"
 ActiveWorkbook.Close
 Application.DisplayAlerts = True


'Schritt 8: Nächstes Element bzw. nächsten Filter holen
 Next pi
 Next pf

End Sub
```

1. In Schritt 1 werden drei Variablen deklariert. Die Variable `pt` wird verwendet, um einen Verweis auf die Pivot-Tabelle abzulegen. In `pf` befindet sich das PivotTable-Feld, das derzeit bearbeitet wird, und in `pi` das Element des Berichtsfilters, das gerade ausgewählt ist.
2. Damit das Makro funktioniert, muss sich die aktive Zelle in einer Pivot-Tabelle befinden. Wenn sich die Zellmarkierung in einer bestimmten Pivot-Tabelle befindet, können Sie davon ausgehen, dass das Makro auf diese Pivot-Tabelle angewendet werden soll.

   In Schritt 2 wird `pt` auf den Namen der Pivot-Tabelle gesetzt, in der sich die aktive Zelle befindet. Sie verwenden hierzu die Eigenschaft `ActiveCell.PivotTable.Name` der Pivot-Tabelle, die bearbeitet werden soll.

   Wenn sich die aktive Zelle nicht in einer Pivot-Tabelle befindet, löst Excel einen Fehler aus. Sie verwenden die Anweisung `On Error Resume Next`, um Excel anzuweisen, die Makroausführung auch beim Eintreten eines Fehlers fortzusetzen.
3. In Schritt 3 wird geprüft, ob sich in der Variablen `pt` ein `PivotTable`-Objekt befindet. Enthält die Variable `pt` den Wert `Nothing`, dann befand sich die aktive Zelle nicht in einer Pivot-Tabelle; folglich konnte der Variablen auch keine Pivot-Tabelle zugewiesen werden. In diesem Fall informieren Sie den Anwender mit einer Meldung und beenden die Prozedur.
4. Schritt 4 prüft, ob die Pivot-Tabelle mehr als einen Berichtsfilter enthält. (Wenn die Eigenschaft `Count` von `PageFields` größer als 1 ist, dann ist mehr als ein Berichtsfilter vorhanden.) Diese Überprüfung führen Sie aus einem einfachen Grund aus: Wenn die Pivot-Tabelle zahlreiche Berichtsfilter enthält kann es schnell passieren, dass Sie Hunderte von neuen Arbeitsmappen exportieren. Wenn die Anzahl der Berichtsfilter größer ist als 1, wird die Makroausführung beendet und eine Meldung angezeigt.

Falls Sie diese Einschränkung entfernen wollen, können Sie einfach den Code von Schritt 4 entfernen oder auskommentieren oder auch einen größeren Grenzwert als 1 eingeben.

5. In Schritt 5 werden zwei ineinander verschachtelte Schleifen gestartet. Die äußere Schleife weist Excel an, alle Berichtsfilter zu durchlaufen. In der inneren Schleife werden alle Pivot-Elemente des Berichtsfilters durchlaufen, der derzeit bearbeitet wird.
6. Das Makro ruft für jedes Pivot-Element den Namen ab und verwendet ihn, um die Auswahl im Berichtsfilter zu ändern. Hierdurch wird praktisch der Filter angewendet und die Pivot-Tabelle passt sich dem Filter an.
7. In Schritt 7 kopieren Sie das Objekt `TableRange1` des `PivotTable`-Objekts. Bei `TableRange1` handelt es sich um ein eingebautes `Range`-Objekt, das auf den Hauptdatenbereich des PivotTable-Berichts verweist. Sie fügen dann die kopierten Daten in eine neue Arbeitsmappe ein und speichern diese. Beachten Sie, dass Sie den Ordner so anpassen müssen, dass er zu Ihrer Arbeitsumgebung passt.
8. In Schritt 8 machen Sie mit dem nächsten Eintrag der Berichtsfilter-Drop-down-Liste weiter. Nachdem Sie alle Elemente des aktuellen Berichtsfilters bearbeitet haben, macht das Makro mit dem nächsten `PageField` (Berichtsfilter) weiter. Das Makro wird beendet, nachdem alle Berichtsfilter bearbeitet wurden.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**

3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Die Größe aller Diagramme eines Tabellenblatts ändern

Wenn Sie ein Dashboard erstellen, sind auch die Optik und das Layout wichtig, damit die Informationen übersichtlich dargestellt werden. Um dies zu erreichen, ist es manchmal erforderlich, Standardabmessungen für Diagramme zu verwenden. Mit diesem Makro lassen sich alle Diagramme eines Tabellenblatts schnell auf die gleiche Größe bringen.

So funktioniert es

Alle Diagramme gehören automatisch zu der Sammlung `ChartObjects`. Um an allen Diagrammen eines Tabellenblatts eine bestimmte Aktion durchzuführen, durchlaufen Sie alle Diagramme, die sich in `ChartObjects` befinden. Jedes Diagramm in `ChartObjects` besitzt einen Index, den Sie verwenden, um das Diagramm auszuwählen. So greifen Sie beispielsweise mit `ChartObjects(1)` auf das erste Diagramm des Tabellenblatts zu.

Im folgenden Makro wird dieses Konzept umgesetzt und mittels eines einfachen Zählers werden die einzelnen Diagramme nacheinander bearbeitet. Nachdem Sie ein Diagramm ausgewählt haben, ändern Sie dessen Größe, indem Sie seine Breite und Höhe auf die vorher festgelegten Werte setzen.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim i As Integer


'Schritt 2: Alle Diagramme in einer Schleife durchlaufen
 For i = 1 To ActiveSheet.ChartObjects.Count
```

```
'Schritt 3: Jedes Diagramm einzeln aktivieren und
' seine Größe ändern
 With ActiveSheet.ChartObjects(i)
 .Width = 300
 .Height = 200
 End With


'Schritt 4: Schleifenzähler inkrementieren
' und mit nächstem Diagramm weitermachen
 Next i

End Sub
```

1. In Schritt 1 wird die Variable `i` des Typs `Integer` definiert; sie wird für die Schleife benötigt.
2. Die Schleife wird in Schritt 2 gestartet. Die Variable `i` wird mit 1 initialisiert und die Schleifenende-Bedingung wird festgelegt. Die maximale Anzahl der Schleifendurchläufe entspricht der Anzahl der Diagramme in `ChartObjects`, die Sie über die Eigenschaft `Count` abfragen. Bei jedem Schleifendurchlauf wird der Wert von `i` um 1 erhöht, bis der Wert von `i` der Anzahl der Diagramme auf dem Tabellenblatt entspricht.
3. In Schritt 3 wird der Wert von `i` als Index von `ChartObjects` verwendet und auf diese Weise ein Diagramm ausgewählt. Anschließend werden die Breite und die Höhe des Diagramms auf die im Code definierten Werte gesetzt. Sie können diese beiden Werte so ändern, dass sie zu Ihren Anforderungen passen.
4. In Schritt 4 wird der Schleifenzähler `i` inkrementiert; sollten noch Diagramme vorhanden sein, wird die Schleife ein weiteres Mal durchlaufen. Das Makro wird beendet, nachdem alle Diagramme bearbeitet wurden.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Diagramm platzieren

Neben der Größenänderung von Diagrammen verbringen viele von uns ziemlich viel Zeit damit, Diagramme so zu positionieren, dass sie an der richtigen Stelle im Dashboard stehen. Dieses Makro verschiebt die Diagramme zu vorher festgelegten Positionen, wodurch sie sich immer am richtigen Platz befinden.

So funktioniert es

Jedes Diagramm besitzt vier Eigenschaften, mit denen seine Größe und seine Position festgelegt werden. Diese Eigenschaften sind: `Width`, `Height`, `Top` und `Left`. Interessant ist, dass jedes `Range`-Objekt die gleichen vier Eigenschaften aufweist. Wenn Sie also diese vier Eigenschaften eines Diagramms so einstellen, dass sie den entsprechenden vier Eigenschaften eines Bereichs entsprechen, wird das Diagramm an diesem Bereich ausgerichtet.

Nachdem das Layout Ihres Dashboards steht, notieren Sie die Bereiche, die den verschiedenen Elementen Ihres Dashboards entsprechen. Verwenden Sie dann diese Bereiche, um die einzelnen Diagramme an der gewünschten Stelle zu positionieren. In diesem Beispiel passen Sie vier Diagramme so an, dass ihre Abmessungen und ihre Positionen einem bestimmten Zellbereich entsprechen.

Beachten Sie, dass Sie auf die Diagramme über deren Namen zugreifen. Die Namen von Diagrammen werden aus dem Wort *Diagramm* und einer fortlaufenden Zahl gebildet, beispielsweise

Diagramm 1, *Diagramm 2* und so weiter. Sie können die Namen der Diagramme sehen, indem Sie auf der Registerkarte START in der Gruppe BEARBEITEN auf SUCHEN UND AUSWäHLEN und dann auf AUSWAHLBEREICH klicken. Sie sehen dann den Arbeitsbereich AUSWAHL (siehe Abbildung 8.5), der alle Objekte des Tabellenblatts anzeigt.

Abbildung 8.5: Im Auswahlbereich werden alle Diagrammobjekte sowie deren Namen angezeigt.

Sie können den Arbeitsbereich AUSWAHL verwenden, um für Ihre Version des Makros die Namen der Diagramme zu ermitteln.

```
Sub Makro1()
```

```
'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim SnapRange As Range


Set SnapRange = ActiveSheet.Range("B6:G19")

 With ActiveSheet.ChartObjects("Diagramm 1")
 .Height = SnapRange.Height
 .Width = SnapRange.Width
 .Top = SnapRange.Top
 .Left = SnapRange.Left
 End With

Set SnapRange = ActiveSheet.Range("B21:G34")
 With ActiveSheet.ChartObjects("Diagramm 2")
 .Height = SnapRange.Height
 .Width = SnapRange.Width
 .Top = SnapRange.Top
 .Left = SnapRange.Left
 End With

Set SnapRange = ActiveSheet.Range("I6:Q19")
 With ActiveSheet.ChartObjects("Diagramm 3")
 .Height = SnapRange.Height
 .Width = SnapRange.Width
 .Top = SnapRange.Top
 .Left = SnapRange.Left
 End With


Set SnapRange = ActiveSheet.Range("I21:Q34")
 With ActiveSheet.ChartObjects("Diagramm 4")
 .Height = SnapRange.Height
 .Width = SnapRange.Width
 .Top = SnapRange.Top
 .Left = SnapRange.Left
 End With


End Sub
```

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Verknüpfung zwischen Diagramm und Daten lösen

Wenn Sie Diagramme aus einer Arbeitsmappe kopieren und diese in eine andere Arbeitsmappe, in PowerPoint oder Outlook einfügen, macht es oft Sinn, die Verknüpfung zwischen dem Diagramm und seiner ursprünglichen Datenquelle zu lösen. Sie vermeiden so die Excel-Fehlermeldungen, die Sie über fehlende Verknüpfungen informieren. Das Makro in diesem Abschnitt kopiert alle Diagramme des aktiven Tabellenblatts, fügt diese in eine neue Arbeitsmappe ein und hebt abschließend die Verknüpfung zwischen dem Diagramm und den Quelldaten auf.

So funktioniert es

Dieses Makro verwendet die Methode `ShapeRange.Group`, um alle Diagramme auf dem aktuellen Tabellenblatt als Formen zu gruppieren. Diese Aktion entspricht der Vorgehensweise, die Sie manuell vornehmen würden, um mehre Formen zu gruppieren. Nachdem Sie aus den Diagrammen eine Gruppe gemacht haben, kopieren Sie die Gruppe und fügen sie in eine neue Arbeitsmappe ein. Die Methode `BreakLink` sorgt abschließend dafür, dass Excel die Diagrammdaten in Arrayformeln konvertiert.

```
Sub Makro()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim wbLinks As Variant
```

```
'Schritt 2: Diagramme gruppieren, Gruppe kopieren
' und Gruppierung wieder aufheben
 With ActiveSheet.ChartObjects.ShapeRange.Group
 .Copy
 .Ungroup
 End With


' Schritt 3: In neue Arbeitsmappe einfügen
' und Gruppierung aufheben
Workbooks.Add.Sheets(1).Paste
Selection.ShapeRange.Ungroup


' Schritt 4: Verknüpfungen zur Datenquelle lösen
wbLinks = _
 ActiveWorkbook.LinkSources(Type:=xlLinkTypeExcelLinks)
ActiveWorkbook.BreakLink Name:=wbLinks(1), _
 Type:=xlLinkTypeExcelLinks


End Sub
```

1. In Schritt 1 wird die Variable `wbLinks` des Typs `Variant` deklariert. Das Makro verwendet diese Variable in Schritt 4, um beim Lösen der Verknüpfung die Datenquelle anzugeben.
2. Schritt 2 verwendet `ChartObjects.ShapeRange.Group`, um alle Diagramme des Tabellenblatts zu gruppieren. Das Makro kopiert dann diese Gruppe in die Zwischenablage. Nach dem Kopieren wird die Gruppierung wieder aufgehoben.
3. Schritt 3 erstellt eine neue Arbeitsmappe und fügt dann die kopierte Gruppe auf *Tabelle1* ein. Nachdem die Gruppe kopiert wurde, können Sie die Gruppierung aufheben, damit jedes Diagramm wieder einzeln bearbeitet werden kann. Beachten Sie, dass die neu erstellte Arbeitsmappe nun das aktive Objekt darstellt; Sie greifen daher mit `ActiveWorkbook` auf die neue Arbeitsmappe zu.
4. Schritt 4 weist der Variablen `wbLinks` die Quelle der Verknüpfung zu. Anschließend weist das Makro Excel an, diese Verknüpfung zu lösen.

Bei dieser Vorgehensweise werden die verknüpften Quelldaten in eine Arrayformel konvertiert. Dies kann schiefgehen, wenn Ihr Diagramm zu viele Datenpunkte enthält. Wie viel ist zu viel? Die genaue Anzahl hängt von der Speicherkapazität des Computers ab, den Sie einsetzen.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Alle Diagramme eines Tabellenblatts drucken

Um ein eingebettetes Diagramm auf einem Tabellenblatt manuell zu drucken, klicken Sie es an und wählen den Befehl DATEI | DRUCKEN. Durch diese Aktion erreichen Sie, dass lediglich das Diagramm gedruckt wird und keine anderen Daten auf dem Tabellenblatt. Auch wenn es ganz einfach ist, ein Diagramm von Hand zu drucken, ist es doch etwas aufwändig, wenn Sie ein Tabellenblatt mit vielen Diagrammen haben. Das Makro dieses Abschnitts erledigt die Arbeit für Sie.

So funktioniert es

Alle Diagramme gehören zu der Sammlung `ChartObjects`. Um an allen Diagrammen eines Tabellenblatts eine bestimmte Aktion

durchzuführen, durchlaufen Sie alle Diagramme, die sich in `ChartObjects` befinden. Jedes Diagramm in `ChartObjects` besitzt einen Index, mit dem Sie das Diagramm auswählen. So greifen Sie beispielsweise mit `ChartObjects(1)` auf das erste Diagramm des Tabellenblatts zu.

Im folgenden Makro wird dieses Konzept umgesetzt und mittels eines einfachen Zählers werden die einzelnen Diagramme nacheinander bearbeitet. Nachdem Sie ein Diagramm ausgewählt haben, drucken Sie es aus.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim i As Integer


'Schritt 2: Alle Diagramme in einer Schleife durchlaufen
 For i = 1 To ActiveSheet.ChartObjects.Count


'Schritt 3: Die Diagramme einzeln aktivieren und drucken
 ActiveSheet.ChartObjects(i).Activate
 ActiveChart.PrintOut Copies:=1


'Schritt 4: Schleifenzähler inkrementieren
' und mit nächstem Diagramm weitermachen
 Next i


End Sub
```

1. In Schritt 1 wird die Variable `i` des Typs `Integer` definiert; sie wird für die Schleife benötigt.
2. Die Schleife wird in Schritt 2 gestartet. Die Variable `i` wird mit 1 initialisiert und die Schleifenende-Bedingung wird festgelegt. Die maximale Anzahl der Schleifendurchläufe entspricht der Anzahl der Diagramme in `ChartObjects`, die Sie über die Eigenschaft `Count` abfragen. Bei jedem Schleifendurchlauf wird der Wert von `i` um 1 erhöht.

3. In Schritt 3 wird der Wert von `i` als Index von `ChartObjects` verwendet und so ein Diagramm ausgewählt. Anschließend verwenden Sie die Methode `ActiveChart.PrintOut`, um den Druckvorgang zu starten. Beachten Sie, dass Sie mit den Konstanten `xlLandspace` (Querformat) und `xlPortrait` (Hochformat) die gewünschte Seitenorientierung einstellen können.
4. In Schritt 4 wird der Schleifenzähler `i` inkrementiert; sollten noch Diagramme vorhanden sein, wird die Schleife ein weiteres Mal durchlaufen. Das Makro wird beendet, nachdem alle Diagramme bearbeitet wurden.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Kapitel 9

E-Mails mit Excel versenden

IN DIESEM KAPITEL

- Aktive Arbeitsmappen als Dateianlage mailen
- Einen bestimmten Bereich als Dateianlage mailen
- Ein einzelnes Tabellenblatt das Dateianlage mailen
- Den Link zu einer Arbeitsmappe mailen
- Bestimmte Tabellenblätter oder Zellbereiche als Dateianhang mailen
- E-Mails an alle Adressen in einer Liste senden
- Alle E-Mail-Anhänge in einem Ordner speichern
- Bestimmte E-Mail-Anhänge in einem Ordner speichern

Haben Sie schon gewusst, dass Excel und Outlook gut zusammenarbeiten können? Falls Sie bereits schon einmal eine Excel-Arbeitsmappe per Outlook versendet oder in einer E-Mail erhalten haben, haben Sie die beiden Programme bereits zusammenarbeiten lassen, wenn auch manuell. Dieses Kapitel zeigt anhand einiger Beispiele, wie Sie die Zusammenarbeit von Excel und Outlook mithilfe von Makros automatisieren.

Die Makros in diesem Kapitel dienen dazu, Outlook zu automatisieren. Damit die vorgestellten Makros funktionieren, muss auf Ihrem Computer Microsoft Outlook installiert sein.

Die aktive Arbeitsmappe als E-Mail-Anhang senden

Die elementarste Outlook-Aufgabe, die Sie automatisiert erledigen können, ist das Versenden einer E-Mail. Im Codebeispiel wird die aktive Arbeitsmappe an zwei E-Mail-Empfänger versendet.

Möglicherweise fragen Sie sich, warum ich nicht die Excel-Methode `SendMail` verwende, mit der Sie direkt aus Excel heraus einfache E-Mails versenden können. Die Methode `SendMail` ist nicht so robust wie die Automation von Outlook. Mit `SendMail` können Sie zum Beispiel keine Dateianhänge versenden. Außerdem können Sie die Felder CC und BCC mit `SendMail` nicht verwenden. Aufgrund dieser Beschränkungen ist das Verfahren, das im Makro dieses Abschnitts verwendet wird, die bessere Wahl.

So funktioniert es

Da der folgende Code aus Excel heraus gestartet wird, müssen Sie in Excel einen Verweis auf die Objektbibliothek von Microsoft Outlook erstellen. Hierzu öffnen Sie aus Excel heraus den Visual-Basic-Editor und wählen im Menü den Befehl EXTRAS | VERWEISE. Führen Sie einen Bildlauf nach unten durch, bis Sie den Eintrag MICROSOFT OUTLOOK XX OBJECT LIBRARY sehen, wobei statt XX die Versionsnummer von Outlook angezeigt wird, die Sie verwenden. Schalten Sie das Kontrollkästchen neben diesem Eintrag ein.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim OLApp As Outlook.Application
 Dim OLMail As Object
```

```
'Schritt 2: Outlook öffnen und neues Mail-Element erzeugen
 Set OLApp = New Outlook.Application
 Set OLMail = OLApp.CreateItem(olMailItem)
 OLApp.Session.Logon


'Schritt 3: Verschiedene Eigenschaften der Mail festlegen
' und Mail versenden
 With OLMail
 .To = "admin@datapigtechnologies.com; " & _
 "mike@datapigtechnologies.com"
 .CC = ""
 .BCC = ""
 .Subject = "Das ist die Betreffzeile"
 .Body = "Dies ist der E-Mail-Inhalt."
 .Attachments.Add ActiveWorkbook.FullName
 .Display '».Display« in ».Send« ändern, damit
'Mail direkt versendet wird
 End With


'Schritt 4: Speicher aufräumen
 Set OLMail = Nothing
 Set OLApp = Nothing

End Sub
```

1. In Schritt 1 werden zwei Variablen deklariert: `OLApp` ist eine Objektvariable, über die Sie auf das `Outlook Application`-Objekt zugreifen können. `OLMail` ist eine weitere Objektvariable, in der sich die Informationen zu der E-Mail-Nachricht befinden.
2. In Schritt 2 wird Outlook aktiviert und eine neue Outlook-Sitzung gestartet. Beachten Sie, dass Sie `OLApp.Session.Logon` verwenden, um sich mit Standardrechten in die aktuelle MAPI-Sitzung einzuloggen. In Schritt 2 wird außerdem eine neue E-Mail-Nachricht erzeugt. Diese Aktion entspricht dem Anklicken der Outlook-Schaltfläche NEUE E-MAIL.
3. In Schritt 3 werden die E-Mail-Nachricht zusammengebaut und die Empfänger (To, CC und BCC), der Betreff, der Nachrichtentext und die Anhänge festgelegt. Bitte beachten Sie, dass die Namen aller Empfänger in Anführungszeichen

eingeschlossen und die einzelnen Empfängeradressen durch Semikolons getrennt werden. Die Standardsyntax für einen Dateianhang lautet `.Attachments.Add "Pfad zur Datei"`. In unserem Beispiel geben Sie den Pfad zur aktuellen Arbeitsmappe mit `ActiveWorkbook.Fullname` an, wodurch die aktive Arbeitsmappe als Dateianhang spezifiziert wird. Nachdem die E-Mail-Nachricht zusammengestellt wurde, verwenden Sie die Methode `.Display`, damit das Fenster zum Versenden einer E-Mail angezeigt wird; dort können Sie die durch den Code vorgenommenen Einstellungen überprüfen und gegebenenfalls ändern. Sie können `.Display` auch durch `.Send` ersetzen; dann wird die E-Mail-Nachricht sofort versendet; das Fenster zum Erstellen einer E-Mail wird nicht angezeigt.

4. Es ist ein bewährtes Verfahren, die Objekte, die Sie Ihren Variablen zugewiesen haben, wieder freizugeben, wenn Sie sie nicht mehr benötigen. Hierdurch vermeiden Sie mögliche Probleme, die sich durch noch im Speicher befindliche, fehlerhafte Objekte ergeben können. Um die Objekte freizugeben, weisen Sie ihnen einfach den Wert `Nothing` zu, so wie hier geschehen.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Einen bestimmten Zellbereich als E-Mail-Anhang senden

Vermutlich wollen Sie nicht in allen Fällen immer die gesamte Arbeitsmappe als Dateianhang versenden. Das Makro in diesem Abschnitt zeigt, wie Sie einen bestimmten Zellbereich per Mail versenden.

So funktioniert es

Da der folgende Code aus Excel heraus gestartet wird, müssen Sie in Excel einen Verweis auf die Objektbibliothek von Microsoft Outlook erstellen. Sie können den Verweis im Visual-Basic-Editor festlegen, indem Sie im Menü den Befehl EXTRAS | VERWEISE wählen. Führen Sie einen Bildlauf nach unten durch, bis Sie den Eintrag MICROSOFT OUTLOOK XX OBJECT LIBRARY sehen, wobei statt XX die Versionsnummer von Outlook angezeigt wird, die Sie verwenden. Schalten Sie das Kontrollkästchen neben diesem Eintrag ein.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim OLApp As Outlook.Application
 Dim OLMail As Object


'Schritt 2: Zellbereich kopieren, in neue temporäre
' Arbeitsmappe einfügen und speichern
 Sheets("Umsatztabelle").Range("A1:E7").Copy
 Workbooks.Add
 Range("A1").PasteSpecial xlPasteValues
 Range("A1").PasteSpecial xlPasteFormats
 ActiveWorkbook.SaveAs ThisWorkbook.Path & _
 "\TempRangeForEmail.xlsx"


'Schritt 3: Outlook öffnen und neues Mail-Element erzeugen
 Set OLApp = New Outlook.Application
 Set OLMail = OLApp.CreateItem(0)
 OLApp.Session.Logon
```

```
'Schritt 4: Verschiedene Eigenschaften der Mail festlegen
' und Mail versenden
 With OLMail
 .To = "admin@datapigtechnologies.com; " & _
 "mike@datapigtechnologies.com"
 .CC = ""
 .BCC = ""
 .Subject = "Das ist die Betreffzeile"
 .Body = "Dies ist der E-Mail-Inhalt."
 .Attachments.Add (ThisWorkbook.Path & _
 "\TempRangeForEmail.xlsx")
.Display '».Display« in ».Send« ändern, damit
'Mail direkt versendet wird
 End With


'Schritt 5: Temporäre Excel-Datei löschen
 ActiveWorkbook.Close SaveChanges:=True
 Kill ThisWorkbook.Path & "\TempRangeForEmail.xlsx"


'Schritt 6: Speicher aufräumen
 Set OLMail = Nothing
 Set OLApp = Nothing

End Sub
```

1. In Schritt 1 werden zwei Variablen deklariert: `OLApp` ist eine Objektvariable, über die Sie auf das `Outlook Application`-Objekt zugreifen können. `OLMail` ist eine weitere Objektvariable, in der sich die Informationen zu der E-Mail-Nachricht befinden.
2. In Schritt 2 wird der angegebene Bereich kopiert und die kopierten Werte und Formate werden in einer temporären Excel-Datei abgelegt. Anschließend speichert das Makro die temporäre Datei und legt dazu den Ordner sowie den Dateinamen fest.
3. In Schritt 3 wird Outlook aktiviert und eine neue Outlook-Sitzung gestartet. Beachten Sie, dass Sie `OLApp.Session.Logon` verwenden, um sich mit Standardrechten in die aktuelle MAPI-

Sitzung einzuloggen. In Schritt 2 wird außerdem eine neue E-Mail-Nachricht erzeugt. Diese Aktion entspricht dem Anklicken der Outlook-Schaltfläche NEUE E-MAIL.

4. In Schritt 4 wird die E-Mail-Nachricht zusammengebaut und die Empfänger (To, CC und BCC), der Betreff, der Nachrichtentext und die Anhänge festgelegt. Bitte beachten Sie, dass die Namen aller Empfänger in Anführungszeichen eingeschlossen und die einzelnen Empfängeradressen durch Semikolons getrennt werden.

   Anschließend legen Sie den Pfad und den Dateinamen der in Schritt 2 erzeugten Datei als Dateianhang für diese E-Mail fest. Nachdem die E-Mail-Nachricht zusammengestellt wurde, verwenden Sie die Methode `.Display`, damit das Fenster zum Versenden einer E-Mail angezeigt wird; dort können Sie die durch den Code vorgenommenen Einstellungen überprüfen und gegebenenfalls ändern. Optional können Sie `.Display` auch durch `.Send` ersetzen; dann wird die E-Mail-Nachricht sofort versendet; das Fenster zum Erstellen einer E-Mail wird nicht angezeigt.

5. Um keine überflüssigen temporären Dateien auf Ihrer Festplatte liegen zu haben, löschen Sie nach dem Versand der E-Mail in Schritt 5 die von Ihnen erstellte temporäre Excel-Datei.

6. Es ist ein bewährtes Verfahren, die Objekte, die Sie Ihren Variablen zugewiesen haben, wieder freizugeben, wenn Sie sie nicht mehr benötigen. Hierdurch vermeiden Sie mögliche Probleme, die sich durch noch im Speicher befindliche, fehlerhafte Objekte ergeben können. Um die Objekte freizugeben, weisen Sie ihnen einfach den Wert `Nothing` zu, so wie hier in Schritt 6 geschehen.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Ein bestimmtes Tabellenblatt als E-Mail-Anhang senden

Dieses Beispiel zeigt, wie Sie anstatt der gesamten Arbeitsmappe lediglich ein bestimmtes Tabellenblatt per Mail versenden.

So funktioniert es

Da der folgende Code aus Excel heraus gestartet wird, müssen Sie in Excel einen Verweis auf die Objektbibliothek von Microsoft Outlook erstellen. Hierzu öffnen Sie aus Excel heraus den Visual-Basic-Editor und wählen im Menü den Befehl EXTRAS | VERWEISE. Führen Sie einen Bildlauf nach unten durch, bis Sie den Eintrag MICROSOFT OUTLOOK XX OBJECT LIBRARY sehen, wobei statt XX die Versionsnummer von Outlook angezeigt wird, die Sie verwenden. Schalten Sie das Kontrollkästchen neben diesem Eintrag ein.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie die Variablen
 Dim OLApp As Outlook.Application
 Dim OLMail As Object


'Schritt 2: Tabellenblatt kopieren, in neue temporäre
' Arbeitsmappe einfügen und speichern
 Sheets("Umsatztabelle").Copy
 ActiveWorkbook.SaveAs ThisWorkbook.Path & _
 "\TempSheetForEmail.xlsx"
```

```
 'Schritt 3: Outlook öffnen und neues Mail-Element erzeugen
 Set OLApp = New Outlook.Application
 Set OLMail = OLApp.CreateItem(0)
 OLApp.Session.Logon


'Schritt 4: Verschiedene Eigenschaften der Mail festlegen
' und Mail versenden
 With OLMail
 .To = "admin@datapigtechnologies.com; " & _
 "mike@datapigtechnologies.com"
 .CC = ""
 .BCC = ""
 .Subject = "Das ist die Betreffzeile"
 .Body = "Das neue Tabellenblatt finden Sie im Anhang."
 .Attachments.Add (ThisWorkbook.Path & _
 "\TempSheetForEmail.xlsx")
.Display '».Display« in ».Send« ändern, damit
'Mail direkt versendet wird
 End With


'Schritt 5: Temporäre Excel-Datei löschen
 ActiveWorkbook.Close SaveChanges:=True
 Kill ThisWorkbook.Path & _
 "\TempSheetForEmail.xlsx"


'Schritt 6: Speicher aufräumen
 Set OLMail = Nothing
 Set OLApp = Nothing

End Sub
```

1. In Schritt 1 werden zwei Variablen deklariert: `OLApp` ist eine Objektvariable, über die Sie auf das `Outlook Application`-Objekt zugreifen können. `OLMail` ist eine weitere Objektvariable, in der sich die Informationen zu der E-Mail-Nachricht befinden.
2. In Schritt 2 wird das angegebene Tabellenblatt kopiert und die kopierten Werte und Formate in einer temporären Excel-Datei gespeichert. Anschließend speichert das Makro die temporäre Datei und legt dabei den Ordner sowie den Dateinamen fest.

3. In Schritt 3 wird Outlook aktiviert und eine neue Outlook-Sitzung gestartet. Beachten Sie, dass Sie `OLApp.Session.Logon` verwenden, um sich mit Standardrechten in die aktuelle MAPI-Sitzung einzuloggen. Außerdem wird eine neue E-Mail-Nachricht erzeugt. Diese Aktion entspricht dem Anklicken der Outlook-Schaltfläche NEUE E-MAIL.
4. In Schritt 4 wird die E-Mail-Nachricht zusammengebaut und die Empfänger (To, CC und BCC), der Betreff, der Nachrichtentext und die Anhänge festgelegt. Bitte beachten Sie, dass die Namen aller Empfänger in Anführungszeichen eingeschlossen und die einzelnen Empfängeradressen durch Semikolons getrennt werden.

   Anschließend legen Sie den Pfad und den Dateinamen der in Schritt 2 erzeugten Datei als Dateianhang für diese E-Mail fest. Nachdem die E-Mail-Nachricht zusammengestellt wurde, verwenden Sie die Methode `.Display`, damit das Fenster zum Versenden einer E-Mail angezeigt wird; dort können Sie die durch den Code vorgenommenen Einstellungen überprüfen und gegebenenfalls ändern. Sie können `.Display` auch durch `.Send` ersetzen; dann wird die E-Mail-Nachricht sofort versendet; das Fenster zum Erstellen einer E-Mail wird nicht angezeigt.
5. Um keine überflüssigen temporären Dateien auf Ihrer Festplatte liegen zu haben, löschen Sie nach dem Versand der E-Mail die von Ihnen erstellte temporäre Excel-Datei.
6. Es ist ein bewährtes Verfahren, die Objekte, die Sie Ihren Variablen zugewiesen haben, wieder freizugeben, wenn Sie sie nicht mehr benötigen. Hierdurch vermeiden Sie mögliche Probleme, die sich durch noch im Speicher befindliche, fehlerhafte Objekte ergeben können. Um die Objekte freizugeben, weisen Sie ihnen wie im Code zu sehen einfach den Wert `Nothing` zu.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Mail mit einem Link zu Ihrer Arbeitsmappe senden

In manchen Fällen ist es nicht erforderlich, die Arbeitsmappe als Anhang zu versenden. Stattdessen können Sie auch eine Mail senden, die einen Link zu der Arbeitsmappe enthält. Dieses Makro zeigt, wie das geht.


Die E-Mail-Empfänger müssen für den Netzwerkspeicherort, der im Link verwendet wird, mindestens Leseberechtigung besitzen, damit sie die Excel-Datei öffnen können, auf die der Link zeigt.

So funktioniert es

Da der folgende Code aus Excel heraus gestartet wird, müssen Sie in Excel einen Verweis auf die Objektbibliothek von Microsoft Outlook erstellen. Hierzu öffnen Sie aus Excel heraus den Visual-Basic-Editor und wählen im Menü den Befehl EXTRAS | VERWEISE. Führen Sie einen Bildlauf nach unten durch, bis Sie den Eintrag MICROSOFT OUTLOOK XX OBJECT LIBRARY sehen, wobei statt XX die Versionsnummer von Outlook angezeigt wird, die Sie verwenden. Schalten Sie das Kontrollkästchen neben diesem Eintrag ein.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim OLApp As Outlook.Application
 Dim OLMail As Object

'Schritt 2: Outlook öffnen und neues Mail-Element erzeugen
 Set OLApp = New Outlook.Application
 Set OLMail = OLApp.CreateItem(0)
 OLApp.Session.Logon

'Schritt 3: Verschiedene Eigenschaften der Mail festlegen
' und Mail versenden
 With OLMail
 .To = "admin@datapigtechnologies.com; " & _
 "mike@datapigtechnologies.com"
 .CC = ""
 .BCC = ""
 .Subject = "Monatsbericht, E-Mail mit Link"
 .HTMLBody = _
 "&lt;p&gt;Der Monatsbericht ist fertig. " & _
 "Zum Herunterladen Link anklicken.&lt;/p&gt;" & _
 "&lt;p&gt;&lt;a href=" & Chr(34) & _
 "Z:\Downloads\Monatsbericht.xlsx" & _
 Chr(34) & "&gt;Jetzt herunterladen&lt;/a&gt;&lt;/p&gt;"
.Display '».Display« in ».Send« ändern, damit
'Mail direkt versendet wird
 End With

'Schritt 4: Speicher aufräumen
 Set OLMail = Nothing
 Set OLApp = Nothing

End Sub
```

1. In Schritt 1 werden zwei Variablen deklariert: `OLApp` ist eine Objektvariable, über die Sie auf das `Outlook Application`-Objekt zugreifen können. `OLMail` ist eine weitere Objektvariable, in der sich die Informationen zu der E-Mail-Nachricht 0.befinden.
2. In Schritt 2 wird Outlook aktiviert und eine neue Outlook-Sitzung gestartet. Beachten Sie, dass Sie `OLApp.Session.Logon` verwenden, um sich mit Standardrechten in die aktuelle MAPI-Sitzung einzuloggen. In Schritt 2 wird außerdem eine neue E-

Mail-Nachricht erzeugt. Dies entspricht dem Anklicken der Outlook-Schaltfläche NEUE E-MAIL.

3. In Schritt 3 wird die E-Mail-Nachricht zusammengebaut und die Empfänger (To, CC und BCC), die Betreffzeile und der Nachrichtentext im HTML-Format festgelegt.

   Um den Hyperlink zu erstellen, verwenden Sie die Eigenschaft `HTMLBody`; dies erlaubt Ihnen, im Nachrichtentext HTML-Tags zu verwenden. Sie müssen den Speicherort der Excel-Datei im Makro durch einen Pfad ersetzen, der in Ihrem Netzwerk gültig ist. Nachdem die E-Mail-Nachricht zusammengestellt wurde, verwenden Sie die Methode `.Display`, damit das Fenster zum Versenden einer E-Mail angezeigt wird; dort können Sie die durch den Code vorgenommenen Einstellungen überprüfen und gegebenenfalls ändern. Sie können `.Display` auch durch `.Send` ersetzen; dann wird die E-Mail-Nachricht sofort versendet; das Fenster zum Erstellen einer E-Mail wird nicht angezeigt.

4. Es ist ein bewährtes Verfahren, die Objekte, die Sie Ihren Variablen zugewiesen haben, wieder freizugeben, wenn Sie sie nicht mehr benötigen. Hierdurch vermeiden Sie mögliche Probleme, die sich durch noch im Speicher befindliche, fehlerhafte Objekte ergeben können. Um die Objekte freizugeben, weisen Sie ihnen, wie im Code in Schritt 4 zu sehen, den Wert `Nothing` zu.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**

4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

E-Mail an alle Adressen in einer Liste mit Kontakten senden

Haben Sie schon einmal eine Massenmail wie einen Newsletter oder ein Memo versenden müssen? Anstatt jede E-Mail-Adresse einzeln in den VBA-Code einzufügen, können Sie auch die folgende Prozedur verwenden. Diese Prozedur versendet eine E-Mail, wobei die E-Mail-Adressen aus einer Liste mit Kontakten entnommen werden, die sich auf einem separaten Tabellenblatt befindet.

So funktioniert es

Da der folgende Code aus Excel heraus gestartet wird, müssen Sie in Excel einen Verweis auf die Objektbibliothek von Microsoft Outlook erstellen. Hierzu öffnen Sie aus Excel heraus den Visual-Basic-Editor und wählen im Menü den Befehl EXTRAS | VERWEISE. Führen Sie einen Bildlauf nach unten durch, bis Sie den Eintrag MICROSOFT OUTLOOK XX OBJECT LIBRARY sehen, wobei statt XX die Versionsnummer von Outlook angezeigt wird, die Sie verwenden. Schalten Sie das Kontrollkästchen neben diesem Eintrag ein.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim OLApp As Outlook.Application
 Dim OLMail As Object
 Dim MyCell As Range
 Dim MyContacts As Range


'Schritt 2: Zellbereich für Schleife festlegen
 Set MyContacts = Sheets("Kontakte").Range("H2:H21")


'Schritt 3: Outlook öffnen
```

```
 Set OLApp = New Outlook.Application
 Set OLMail = OLApp.CreateItem(0)
 OLApp.Session.Logon

'Schritt 4: Jede Adresse aus der Kontakte-Liste einfügen
 With OLMail
 .BCC = ""
 For Each MyCell In MyContacts
 .BCC = .BCC & Chr(59) & MyCell.Value
 Next MyCell
 .Subject = "Beispieldatei Kapitel 9"
 .Body = "Die Beispieldatei befindet sich im Anhang."
 .Attachments.Add ActiveWorkbook.FullName
.Display '».Display« in ».Send« ändern, damit
'Mail direkt versendet wird
 End With


'Schritt 5: Speicher aufräumen
 Set OLMail = Nothing
 Set OLApp = Nothing

End Sub
```

1. In Schritt 1 werden vier Objektvariablen deklariert: Über `OLApp` greifen Sie auf das `Outlook Application`-Objekt zu, `OLMail` wird für das Erstellen der E-Mail-Nachricht verwendet, bei `MyCell` und `MyContacts` handelt es sich um Excel-Bereiche.
2. In Schritt 2 wird `MyContacts` der Zellbereich zugewiesen, in dem sich die E-Mail-Adressen befinden. Dies ist der Zellbereich, der später in einer Schleife bearbeitet wird und aus dem die einzelnen E-Mail-Adressen in die E-Mail-Nachricht eingefügt werden.
3. In Schritt 3 wird Outlook aktiviert und eine neue Outlook-Sitzung gestartet. Beachten Sie, dass Sie `OLApp.Session.Logon` verwenden, um sich mit Standardrechten in die aktuelle MAPI-Sitzung einzuloggen. In Schritt 3 wird außerdem eine neue E-Mail-Nachricht erzeugt. Diese Aktion entspricht dem Anklicken der Outlook-Schaltfläche NEUE E-MAIL.
4. In Schritt 4 wird die E-Mail-Nachricht zusammengebaut. Beachten Sie, dass Sie jede Zelle im Bereich `MyContacts`

bearbeiten und die dort enthaltenen E-Mail-Adressen in die Eigenschaft `BCC` einfügen. Sie verwenden die Eigenschaft `BCC`, damit jeder Empfänger eine E-Mail erhält, die so aussieht, als wäre sie nur an ihn geschickt. Die Empfänger können die anderen E-Mail-Adressen nicht sehen, da sie sich im Feld `BCC` (Blindkopie) befinden.

Nachdem die E-Mail-Nachricht zusammengestellt wurde, verwenden Sie die Methode `.Display`, damit das Fenster zum Versenden einer E-Mail angezeigt wird; dort können Sie die durch den Code vorgenommenen Einstellungen überprüfen und gegebenenfalls ändern. Sie können `.Display` auch durch `.Send` ersetzen, dann wird die E-Mail-Nachricht sofort versendet; das Fenster zum Erstellen einer E-Mail wird nicht angezeigt.

5. Es ist ein bewährtes Verfahren, die Objekte, die Sie Ihren Variablen zugewiesen haben, wieder freizugeben, wenn Sie sie nicht mehr benötigen. Hierdurch vermeiden Sie mögliche Probleme, die sich durch noch im Speicher befindliche, fehlerhafte Objekte ergeben können. Um die Objekte freizugeben, weisen Sie ihnen, wie im Code in Schritt 5 zu sehen, den Wert `Nothing` zu.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Alle E-Mail-Anhänge in einem Ordner speichern

In vielen Geschäftsprozessen bietet es sich an, Daten per E-Mail auszutauschen. Angenommen, Sie versenden eine Vorlage für das Budget an die Leiter aller Niederlassungen, damit diese sie ausfüllen und an Sie zurücksenden. Falls die Mail an 150 Niederlassungsleiter versendet wird, ist es ein mühsames Unternehmen, die erhaltenen E-Mail-Anhänge manuell zu speichern.

Die folgende Prozedur zeigt eine mögliche Lösung dieses Problems. In dieser Prozedur verwenden Sie Automatisierung, um alle Anhänge in Ihrem Posteingang zu suchen und diese dann in einem bestimmten Ordner zu speichern.

So funktioniert es

Da der folgende Code aus Excel heraus gestartet wird, müssen Sie in Excel einen Verweis auf die Objektbibliothek von Microsoft Outlook erstellen. Hierzu öffnen Sie aus Excel heraus den Visual-Basic-Editor und wählen im Menü den Befehl EXTRAS | VERWEISE. Führen Sie einen Bildlauf nach unten durch, bis Sie den Eintrag MICROSOFT OUTLOOK XX OBJECT LIBRARY sehen, wobei statt XX die Versionsnummer von Outlook angezeigt wird, die Sie verwenden. Schalten Sie das Kontrollkästchen neben diesem Eintrag ein.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim ns As Namespace
 Dim MyInbox As MAPIFolder
 Dim MItem As MailItem
 Dim Atmt As Attachment
 Dim FileName As String


'Schritt 2: Einen Verweis auf Ihren Posteingang erzeugen
 Set ns = GetNamespace("MAPI")
```

```
 Set MyInbox = ns.GetDefaultFolder(olFolderInbox)


'Schritt 3: Prüfen, ob Mails im Posteingang vorhanden sind;
' Prozedur beenden, falls nicht
 If MyInbox.Items.Count = 0 Then
 MsgBox "Keine Mails im Ordner vorhanden."
 Exit Sub
 End If


'Schritt 4: Ordner für das Speichern der Anhänge erzeugen
 On Error Resume Next
 MkDir "C:\OffTheGrid\MyAttachments\"



'Schritt 5: Alle Mail-Elemente in einer Schleife ' bearbeiten
 For Each MItem In MyInbox.Items


'Schritt 6: Die einzelnen Anhänge der aktuellen Mail
' speichern
 For Each Atmt In MItem.Attachments
 FileName = "C:\OffTheGrid\MyAttachments\" & _
 Atmt.FileName
 Atmt.SaveAsFile FileName
 Next Atmt


'Schritt 7: Mit dem nächsten Mail-Element weitermachen
 Next MItem


'Schritt 8: Speicher aufräumen
 Set ns = Nothing
 Set MyInbox = Nothing

End Sub
```

1. In Schritt 1 werden fünf Variablen deklariert: Über `ns` greifen Sie auf den MAPI-Namensraum zu, über `MyInbox` haben Sie Zugriff auf den Mailordner. `MItem` stellt die Eigenschaften einer erhaltenen E-Mail bereit. `Atmt` ist eine weitere Objektvariable, über die Sie auf alle Anhänge einer E-Mail-Nachricht

zugreifen. In `FileName`, einer Variablen des Typs `String`, speichern Sie den Dateinamen eines E-Mail-Anhangs ab.

2. In Schritt 2 wird der Variablen `MyInbox` der Posteingang des Standard-E-Mail-Programms zugewiesen.
3. In Schritt 3 wird erst einmal überprüft, ob der Posteingang Nachrichten enthält. Falls gar keine E-Mail-Nachrichten vorhanden sind, wird ein Meldungsfeld angezeigt und die Prozedur beendet.
4. In Schritt 4 wird das Verzeichnis erstellt, in dem die gefundenen E-Mail-Anhänge gespeichert werden sollen. Sie können hier einen vorhandenen Ordner verwenden. Es ist jedoch besser, für das Speichern der Anhänge ein neues Verzeichnis zu erstellen. Sie erstellen hier ein neues Verzeichnis und verwenden vorher `On Error Resume Next`. Auf diese Weise vermeiden Sie, dass die Makroausführung bei einem Fehler beendet wird; Excel erzeugt einen Fehler, falls das Verzeichnis, das Sie im Code erstellen wollen, bereits existiert.
5. Die Schleife wird in Schritt 5 gestartet. In der Schleife wird jedes Mail-Element im angegebenen Ordner bearbeitet.
6. In Schritt 6 wird geprüft, ob die aktuelle E-Mail Anhänge enthält. Ist dies der Fall, wird jeder E-Mail-Anhang in dem vorher erstellten Ordner in einer eigenen Datei gespeichert.
7. In Schritt 7 wird das nächste Mail-Element abgerufen und dann die Schleife bei Schritt 5 weiter abgearbeitet.
8. Es ist ein bewährtes Verfahren, die Objekte, die Sie Ihren Variablen zugewiesen haben, wieder freizugeben, wenn Sie sie nicht mehr benötigen. Hierdurch vermeiden Sie mögliche Probleme, die sich durch noch im Speicher befindliche, fehlerhafte Objekte ergeben können. Um die Objekte freizugeben, weisen Sie ihnen, wie im Code in Schritt 8 zu sehen, den Wert `Nothing` zu.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**
4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Bestimmte E-Mail-Anhänge in einem Ordner speichern

In der Prozedur des vorigen Beispiels haben Sie nach *allen* Anhängen in Ihrem Posteingang gesucht und diese dann in einem bestimmten Ordner gespeichert. In den meisten Fällen wollen Sie aber vermutlich gar nicht alle Anhänge speichern, sondern nur bestimmte, wie beispielsweise nur die Anhänge, in deren Mail-Betreffzeile bestimmte Schlüsselwörter auftauchen. Das Makro in diesem Abschnitt zeigt, wie Sie die Betreffzeile einer E-Mail-Nachricht untersuchen und nur ausgewählte Anhänge speichern.

So funktioniert es

Da der folgende Code aus Excel heraus gestartet wird, müssen Sie in Excel einen Verweis auf die Objektbibliothek von Microsoft Outlook erstellen. Hierzu öffnen Sie aus Excel heraus den Visual-Basic-Editor und wählen im Menü den Befehl EXTRAS | VERWEISE. Führen Sie einen Bildlauf nach unten durch, bis Sie den Eintrag MICROSOFT OUTLOOK XX OBJECT LIBRARY sehen, wobei statt XX die Versionsnummer von Outlook angezeigt wird, die Sie verwenden. Schalten Sie das Kontrollkästchen neben diesem Eintrag ein.

```
Sub Makro1()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim ns As Namespace
 Dim MyInbox As MAPIFolder
 Dim MItem As Object
 Dim Atmt As Attachment
 Dim FileName As String
 Dim i As Integer

'Schritt 2: Einen Verweis auf Ihren Posteingang erzeugen
 Set ns = GetNamespace("MAPI")
 Set MyInbox = ns.GetDefaultFolder(olFolderInbox)


'Schritt 3: Prüfen, ob Mails im Posteingang vorhanden sind;
' Prozedur beenden, falls nicht
 If MyInbox.Items.Count = 0 Then
 MsgBox "Keine Mails im Ordner vorhanden."
 Exit Sub
 End If


'Schritt 4: Ordner für das Speichern der Anhänge erstellen
 On Error Resume Next
 MkDir "C:\OffTheGrid\MyAttachments\"


'Schritt 5: Alle Mail-Elemente in einer Schleife
' bearbeiten
 For Each MItem In MyInbox.Items


'Schritt 6: Prüfen, ob die Betreffzeile das Wort
' »Datenübermittlung« enthält
 If InStr(1, MItem.Subject, "Datenübermittlung") &lt; 1 Then
 GoTo SkipIt
 End If

'Schritt 7: Die Anhänge mit einer eindeutigen Kennziffer
' versehen und speichern, dann mit dem nächsten Anhang
' weitermachen
 i = 0
 For Each Atmt In MItem.Attachments
 FileName = _
 "C:\OffTheGrid\MyAttachments\Anhang-" & i & _
 "-" & Atmt.FileName
```

```
 Atmt.SaveAsFile FileName
 i = i + 1
 Next Atmt


'Schritt 8: Mit dem nächsten Mail-Element weitermachen
SkipIt:
 Next MItem


'Schritt 9: Speicher aufräumen
 Set ns = Nothing
 Set MyInbox = Nothing

End Sub
```

1. In Schritt 1 werden sechs Variablen deklariert: Über `ns` greifen Sie auf den MAPI-Namensraum zu, über `MyInbox` haben Sie Zugriff auf den Mailordner. `MItem` stellt die Eigenschaften einer erhaltenen E-Mail bereit. `Atmt` ist eine weitere Objektvariable, über die Sie auf alle Anhänge einer E-Mail-Nachricht zugreifen. In `FileName`, einer Variablen des `Typs` String, speichern Sie den Dateinamen eines E-Mail-Anhangs ab. `i` ist eine Variable des Typs `Integer` und wird verwendet, um sicherzustellen, dass jeder Anhang unter einem eindeutigen Dateinamen gespeichert wird.
2. In Schritt 2 wird der Variablen `MyInbox` der Posteingang des Standard-E-Mail-Programms zugewiesen.
3. In Schritt 3 wird zunächst einmal überprüft, ob der Posteingang Nachrichten enthält. Falls keine E-Mails vorhanden sind, wird ein Meldungsfeld angezeigt und die Prozedur beendet.
4. In Schritt 4 wird das Verzeichnis erstellt, in dem die gefundenen E-Mail-Anhänge gespeichert werden sollen. Sie erstellen hier ein neues Verzeichnis und verwenden vorher `On Error Resume Next`. Auf diese Weise vermeiden Sie, dass die Makroausführung bei einem Fehler beendet wird; Excel erzeugt einen Fehler, falls das Verzeichnis, das Sie im Code erstellen wollen, bereits existiert.

5. Die Schleife wird in Schritt 5 gestartet. In der Schleife wird jedes Mail-Element im angegebenen Ordner bearbeitet.
6. In Schritt 6 wird geprüft, ob die Betreffzeile der aktuellen E-Mail den Text *Datenübermittlung* enthält. Falls dieses Wort nicht gefunden wird, müssen Sie die Mail nicht weiter bearbeiten; die Codeausführung wird dann an der Sprungmarke `SkipIt` fortgesetzt. Die Zeile, die sich nach der Sprungmarke befindet, sorgt dafür, dass mit der nächsten E-Mail im Posteingang weitergemacht wird.
7. In Schritt 7 wird geprüft, ob die E-Mail Anhänge besitzt, und es werden die einzelnen Anhänge im von Ihnen erstellten Ordner gespeichert. Hierbei wird der Dateiname jedes Anhangs um eine fortlaufende Nummer ergänzt; dies gewährleistet, dass die Dateinamen eindeutig sind und nicht versehentlich eine bereits vorhandene Datei überschrieben wird.
8. In Schritt 8 wird das nächste Mail-Element abgerufen und dann die Schleife bei Schritt 5 weiter abgearbeitet.
9. Es ist ein bewährtes Verfahren, die Objekte, die Sie Ihren Variablen zugewiesen haben, wieder freizugeben, wenn Sie sie nicht mehr benötigen. Hierdurch vermeiden Sie mögliche Probleme, die sich durch noch im Speicher befindliche, fehlerhafte Objekte ergeben können. Um die Objekte freizugeben, weisen Sie ihnen, wie im Code in Schritt 9 zu sehen, den Wert `Nothing` zu.

So verwenden Sie es

Um dieses Makro zu implementieren, kopieren Sie es und fügen es in ein Standardmodul ein.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie Alt + F11 drücken.**
2. **Klicken Sie im Projekt-Explorer den Namen des Projekts/der Arbeitsmappe mit der rechten Maustaste an.**
3. **Wählen Sie EINFüGEN | MODUL.**

4. **Geben Sie den Code in das neu erstellte Modul ein oder kopieren Sie ihn.**

Kapitel 10

Externe Daten mit Makros bearbeiten

IN DIESEM KAPITEL

- Arbeiten mit externen Datenverbindungen
- Verbindungen zu externen Datenbanken herstellen
- Mit Makros externe Daten abrufen
- Arbeiten mit externen Textdateien

Externe Daten sind genau das, wonach es klingt: Daten, die sich nicht in der Excel-Arbeitsmappe befinden, in der Sie arbeiten. Einige Beispiele für externe Datenquellen sind Textdateien, CSV-Dateien, Access-Tabellen und sogar SQL-Server-Tabellen.

Es gibt zahlreiche Möglichkeiten, Daten in Excel zu importieren. Es ist sogar so, dass in der Benutzeroberfläche von Excel und in VBA/Code weit mehr Techniken zur Verfügung stehen, als in diesem Kapitel behandelt werden können. Daher konzentriert sich dieses Kapitel auf eine Handvoll Techniken, die Sie in den meisten Situationen implementieren können und die nicht zu viele Fallstricke und Stolpersteine aufweisen.

Externe Datenverbindungen verwenden

Jede Excel-Arbeitsmappe kann die Syntax speichern, die es der Arbeitsmappe ermöglicht, Daten aus externen Datenbanken, Dateien und Websites abzurufen. Diese Fähigkeit wird durch das

Excel-Feature »Externe Datenverbindungen« ermöglicht. Sie können eine beliebige Anzahl von Datenverbindungen erstellen, um die gewünschten externen Daten abzurufen.

Der folgende Abschnitt führt Sie in die Details bei der Arbeit mit externen Datenverbindungen ein.

Eine Verbindung manuell erstellen

In Excel ist es einfach, manuell eine Verbindung mit externen Datenquellen wie Microsoft Access, SQL Server oder einer anderen ODBC-Ressource herzustellen, die Sie regelmäßig verwenden.

Ab Excel 2010/2013 bis zur aktuellen Excel-Version in Microsoft 365 wurde die Funktionalität der externen Datenverbindungen kontinuierlich weiterentwickelt. In der aktuellen Excel-Version wird das Modell »Daten abrufen und transformieren« verwendet, bei dem die Daten über Power Query importiert und auch dort transformiert, beispielsweise gefiltert werden. Die folgenden Schritte beschreiben die Vorgehensweise, wie Sie bis Excel 2016 verwendet wurde. Wenn Sie Excel 2019 oder Excel Microsoft 365 nutzen, müssen Sie die folgenden Aktionen durchführen, damit Sie die Schritte in diesem Abschnitt nachvollziehen können:

1. **Wählen Sie DATEI | OPTIONEN.**
2. **Klicken Sie in der Kategorienleiste an der linken Seite des Dialogfeldes EXCEL-OPTIONEN auf DATEN.**
3. **Schalten Sie im Abschnitt LEGACY-DATENIMPORT-ASSISTENTEN ANZEIGEN das Kontrollkästchen AUS ACCESS (LEGACY) ein.**
4. **Klicken Sie auf OK.**

Sie können die Access-Datenbank *Facility Services.accdb* verwenden, die Sie in den Beispieldateien zum Buch finden, um die vorgestellten Schritte nachzuvollziehen. In den Begleitdateien befindet sich ebenfalls die Datei *10.1*

DynamischeDatenverbindung.xlsm, die die Beispielmakros dieses Abschnitts enthält. Weitere Informationen zu den Begleitdateien finden Sie in der Einleitung des Buchs.

Um beispielswiese eine Verbindung zu einer Access-Datenbank herzustellen, führen Sie diese Schritte durch:

1. **Erstellen Sie eine neue Excel-Arbeitsmappe und klicken Sie im Menüband auf Daten.**
2. **Führen Sie eine der folgenden Aktionen durch:**
    - Wenn Sie eine Excel-Version bis einschließlich Excel 2016 verwenden, klicken Sie in der Gruppe EXTERNE DATEN ABRUFEN auf AUS ACCESS.
    - Wenn Sie Excel 2019 oder neuer verwenden, wählen Sie in der Gruppe DATEN ABRUFEN UND TRANSFORMIEREN den Befehl DATEN ABRUFEN | LEGACY-ASSISTENTEN | AUS ACCESS (LEGACY).

    Das Dialogfeld DATENQUELLE AUSWäHLEN wird geöffnet, wie in Abbildung 10.1 dargestellt. Wenn es sich bei der Datenbank, aus der Sie Daten importieren möchten, um eine lokale Datenbank handelt, navigieren Sie zum Speicherort der Datei und wählen Sie sie aus. Wenn sich Ihre Access-Zieldatenbank auf einem Netzlaufwerk an einem anderen Speicherort befindet, benötigen Sie die entsprechende Berechtigung, um sie auszuwählen.

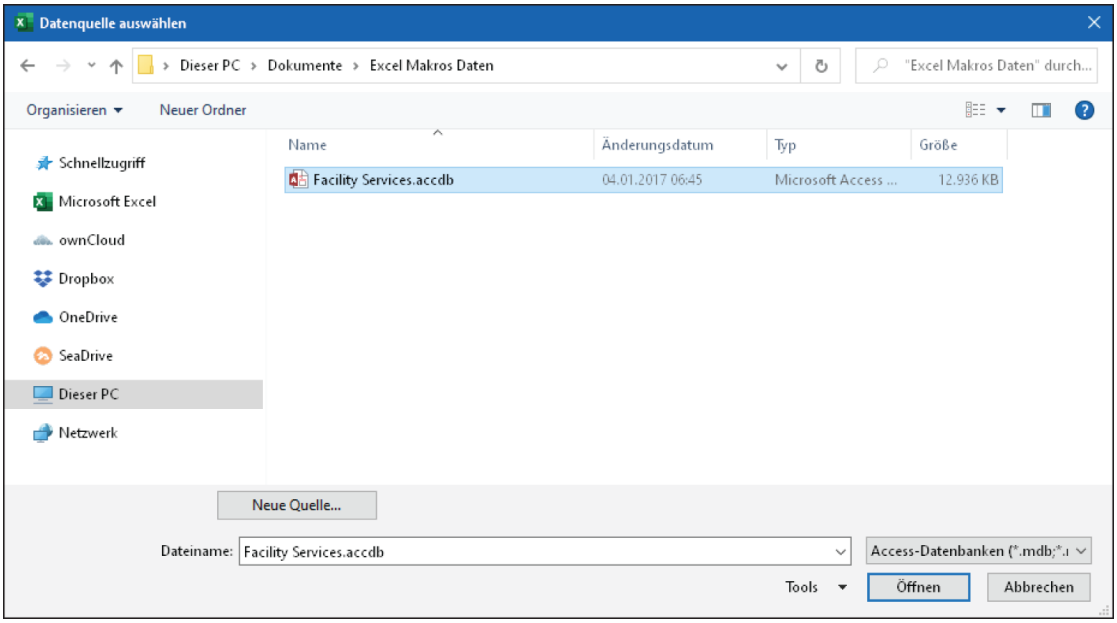


Abbildung 10.1: Wählen Sie die Datenbankquelle aus, die die Daten enthält, die Sie importieren möchten.

3. **Navigieren Sie zu Ihrer Beispieldatenbank, markieren Sie sie und klicken Sie auf Öffnen.**

   Das Dialogfeld TABELLE AUSWäHLEN (siehe Abbildung 10.2) wird geöffnet. In diesem Dialogfeld werden alle in der ausgewählten Datenbank verfügbaren Tabellen und Abfragen aufgelistet.

   Das Dialogfeld TABELLE AUSWäHLEN enthält eine Spalte mit dem Namen Typ. Es gibt zwei Arten von Access-Objekten, mit denen Sie arbeiten können: Sichten (VIEW) und Tabellen (TABLE). VIEW zeigt an, dass es sich bei dem angezeigten Dataset um eine Access-Abfrage handelt, und TABLE gibt an, dass es sich bei dem Dataset um eine Access-Tabelle handelt. In diesem Beispiel ist *Sales_By_Employee* eigentlich eine Access-Abfrage. Dies bedeutet, dass Sie die Ergebnisse der Abfrage importieren. Dies ist echte Arbeitsteilung: Access übernimmt die gesamte Backend-Datenverwaltung und -Aggregation und Excel kümmert sich um Analyse und Präsentation.

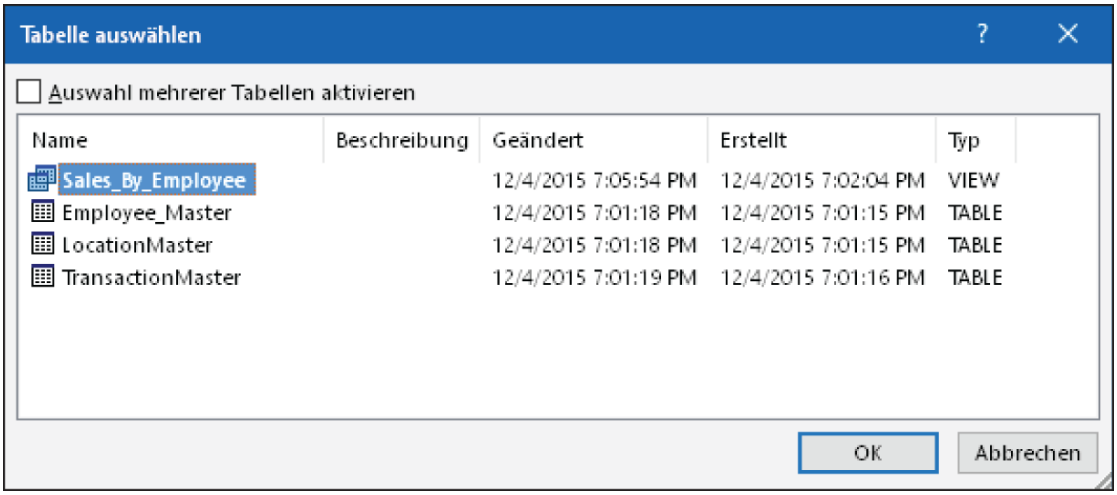


Abbildung 10.2: Wählen Sie das Access-Objekt aus, das Sie importieren wollen.

4. **Wählen Sie die Tabelle oder Abfrage aus, die Sie verwenden wollen, und klicken Sie auf OK.**

In Fällen, in denen Ihre Access-Datenbank kennwortgeschützt ist, aktiviert Schritt 3 eine Reihe von Dialogfeldern zu Datenverknüpfungseigenschaften, in denen nach Anmeldeinformationen gefragt wird (das heißt Benutzername und Kennwort). Die meisten Access-Datenbanken benötigen keine Anmeldeinformationen, aber wenn Ihre Datenbank einen Benutzernamen und ein Kennwort erfordert, geben Sie sie in das Dialogfeld DATENVERKNüPFUNGSEIGENSCHAFTEN ein.

Das Dialogfeld DATEN IMPORTIEREN (siehe Abbildung 10.3) wird geöffnet. Hier legen Sie fest, wo und wie die Tabelle importiert werden soll. Sie können die Daten nun in eine Tabelle, einen PivotTable-Bericht oder ein PivotChart importieren. Sie haben auch die Möglichkeit, nur die Verbindung zu erstellen. Sie können diese Verbindung dann zu einem späteren Zeitpunkt verwenden.

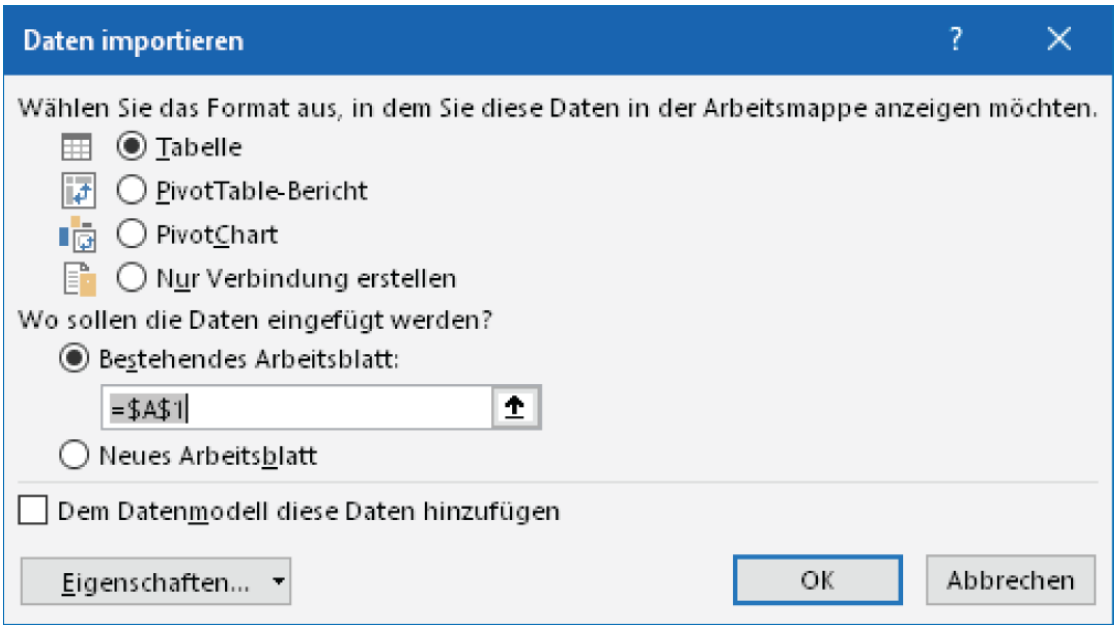


Abbildung 10.3: Legen Sie fest, wo und wie Sie die Access-Daten sehen wollen.

Beachten Sie, dass die Daten, wenn Sie PIVOTCHART oder PIVOTTABLE-BERICHT auswählen, in einem Pivot-Cache gespeichert werden, ohne dass die tatsächlichen Daten in das Arbeitsblatt geschrieben werden. So kann Ihre Pivot-Tabelle normal funktionieren, ohne dass Sie möglicherweise Hunderttausende von Datenzeilen zweimal importieren müssen (einmal für den Pivot-Cache und einmal für die Kalkulationstabelle).

5. **Wählen Sie TABELLE als Ausgabeformat und legen Sie fest, dass die Daten ab Zelle A1 ausgegeben werden sollen, wie in Abbildung 10.3 zu sehen.**
6. **Klicken Sie auf OK.**

Als Belohnung für all Ihre Mühen wird eine Tabelle eingefügt (siehe Abbildung 10.4), die die importierten Daten aus Ihrer Access-Datenbank enthält.

Wenn Sie Daten auf diese Weise importieren, ist das unglaublich Mächtige dabei die Möglichkeit zur Aktualisierung der Werte. Wenn Sie mit dieser Technik Daten aus Access importieren, erstellt Excel eine Tabelle, die Sie aktualisieren können, indem Sie sie mit der rechten Maustaste anklicken und im Kontextmenü AKTUALISIEREN auswählen, wie in Abbildung 10.5 dargestellt. Wenn Sie die importierten Daten aktualisieren, stellt Excel erneut

eine Verbindung zu Ihrer Access-Datenbank her und importiert die Daten erneut. Solange eine Verbindung zu Ihrer Datenbank verfügbar ist, können Sie die Daten auf diese Weise per Mausklick aktualisieren.

	A	B	C	D	E
1	Region	Market	Branch_Number	Employee_Number	Last_Name
2	MIDWEST	TULSA	401612	1336	RACHTIR
3	MIDWEST	TULSA	401612	1336	RACHTIR
4	MIDWEST	TULSA	401612	60224	HERVIY
5	MIDWEST	TULSA	401612	60224	HERVIY
6	MIDWEST	TULSA	401612	55662	WHATILIY
7	MIDWEST	TULSA	401612	60224	HERVIY
8	MIDWEST	TULSA	401612	1336	RACHTIR
9	MIDWEST	TULSA	401612	55662	WHATILIY
10	MIDWEST	TULSA	401612	55662	WHATILIY
11	MIDWEST	TULSA	401612	1336	RACHTIR
12	MIDWEST	TULSA	401612	55662	WHATILIY
13	MIDWEST	TULSA	401612	55662	WHATILIY

Abbildung 10.4: Daten, die aus Access importiert wurden.

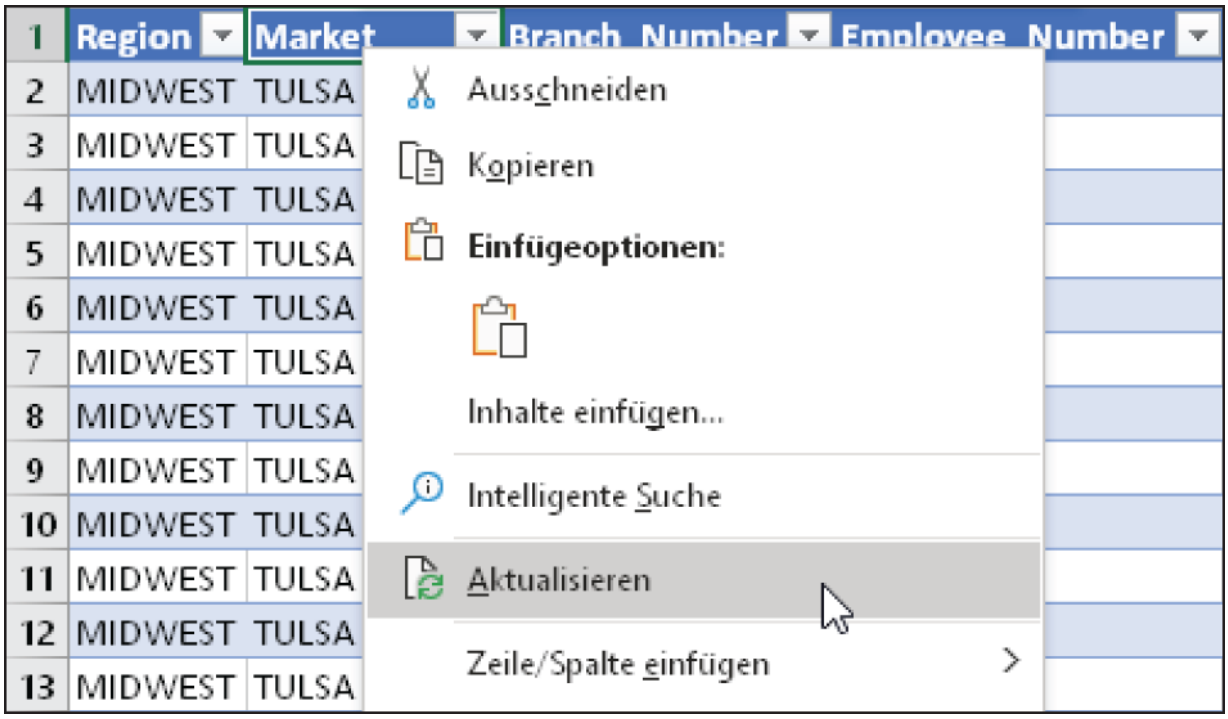


Abbildung 10.5: Solange eine Verbindung mit Ihrer Datenbank verfügbar ist, können Sie die Tabelle mit den neuesten Daten aktualisieren.

Ein großer Vorteil bei der Verwendung der Gruppe EXTERNE DATEN ABRUFEN (bis Excel 2016) bzw. DATEN ABRUFEN UND TRANSFORMIEREN (ab Excel 2019) besteht wiederum darin, dass Sie eine Datenverbindung zwischen Excel und Access herstellen können. In den meisten Fällen können Sie die Verbindung einmal einrichten und die Datenverbindung bei Bedarf einfach aktualisieren. Sie können sogar ein Excel-Makro aufzeichnen, um die Daten für einen Trigger oder beim Eintritt eines Ereignisses aktualisieren zu lassen, was für die Automatisierung das Datentransfers von Access nach Excel ideal ist.

Datenverbindungen manuell bearbeiten

Nachdem Sie die Verbindung erstellt haben, können Sie die Verbindungseigenschaften verwenden, um auf eine andere Datenbanktabelle oder Abfrage zu verweisen. Sie können sogar eigene SQL-Anweisungen schreiben. SQL (Structured Query Language) ist die gebräuchlichste Sprache, um mit relationalen Datenbanksystemen (zum Beispiel Microsoft Access) nützliche Aufgaben zu erledigen. Sie können Anweisungen direkt aus Excel in Form von SQL-Anweisungen übergeben. Hierdurch erhalten Sie mehr Kontrolle über die Daten, die Sie in Ihr Excel-Modell importieren.

In der aktuellen Excel-Version (Excel 2019 und Excel Microsoft 365) werden externe Daten standardmäßig mit dem Power-Query-Editor importiert. Anders als in den folgenden Abschnitten beschrieben, bearbeiten Sie die Datenverbindung nicht mehr direkt in Excel, sondern im Power-Query-Editor. Damit Sie die Datenverbindung wie hier beschrieben bearbeiten können, müssen Sie die Verbindung zur Access-Datenbank über den Legacy-Assistenten herstellen, wie im Abschnitt »Eine Verbindung manuell erstellen« weiter oben in diesem Kapitel beschrieben.

Obwohl eine ausführliche Erörterung von SQL den Rahmen dieses Buchs bei Weitem sprengen würde, lassen Sie uns ein wenig außerhalb unserer Komfortzone gehen und unsere externe Datenverbindung mit einer einfachen SQL-Anweisung bearbeiten, um einen anderen Satz von Daten abzurufen.

1. **Führen Sie eine der beiden folgenden Aktionen durch:**
    - **Excel 2016 und älter:** Öffnen Sie die Registerkarte DATEN und klicken Sie auf VERBINDUNGEN. Hierdurch wird das Dialogfeld ARBEITSMAPPENVERBINDUNGEN geöffnet.
    - **Excel 2019 und neuer:** Öffnen Sie die Registerkarte DATEN und klicken Sie auf ABFRAGEN UND VERBINDUNGEN. Hierdurch wird der gleichnamige Aufgabenbereich geöffnet (siehe Abbildung 10.6).

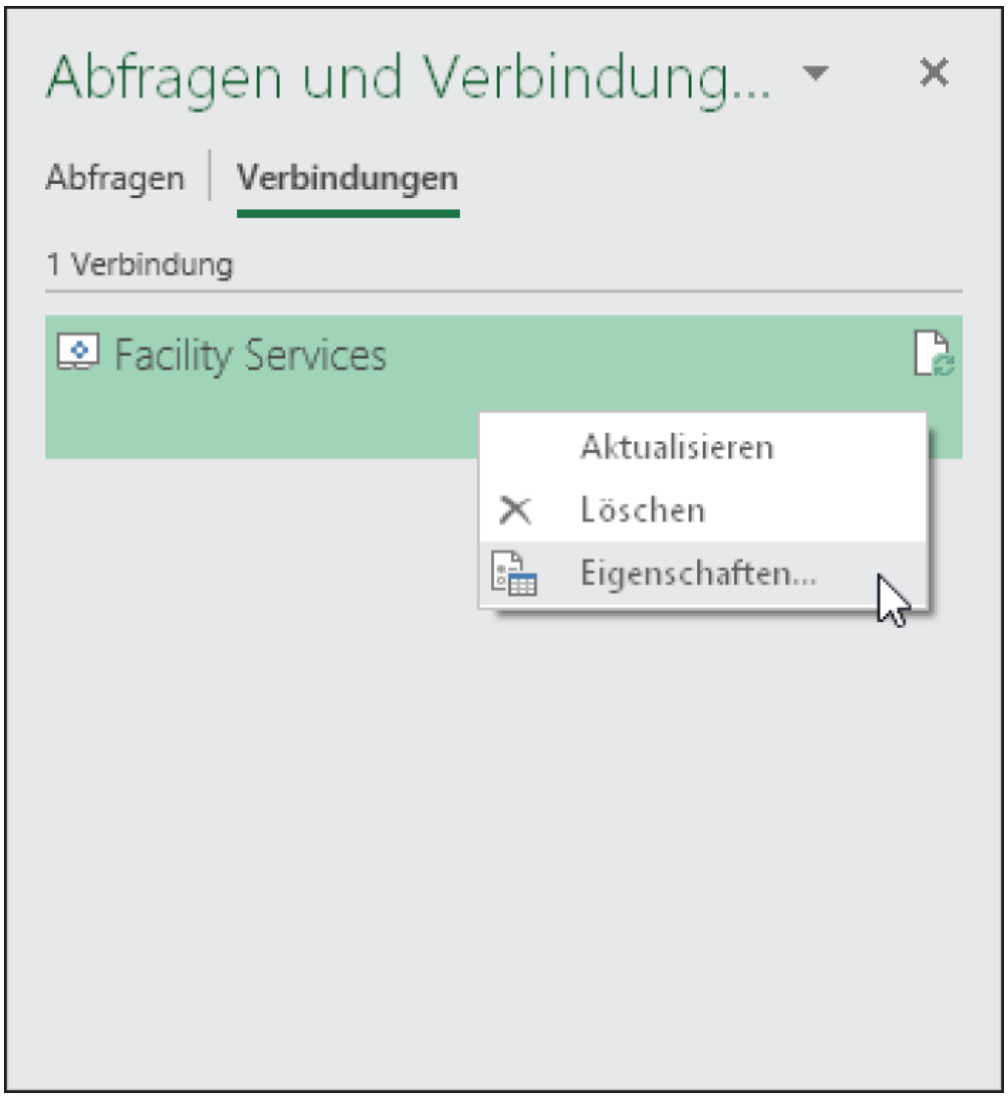


Abbildung 10.6: Im Excel 2019-Aufgabenbereich ABFRAGEN UND VERBINDUNGEN werden auf der Registerkarte VERBINDUNGEN alle Verbindungen zu externen Datenbanken angezeigt.

2. **Führen Sie eine der folgenden Aktionen durch:**
   - **Excel 2016 und älter:** Markieren Sie die Verbindung an, die Sie bearbeiten wollen, und klicken Sie dann auf EIGENSCHAFTEN.
   - **Excel 2019 und neuer:** Klicken Sie die Verbindung, die Sie bearbeiten wollen, mit der rechten Maustaste an, und wählen Sie im Kontextmenü den Befehl EIGENSCHAFTEN.

3. **Das Dialogfeld VERBINDUNGSEIGENSCHAFTEN wird angezeigt. Klicken Sie dort auf die Registerkarte DEFINITION (siehe Abbildung 10.7).**
4. **Ändern Sie den Befehlstyp in SQL ab und geben Sie dann Ihre SQL-Anweisung ein. Verwenden Sie für unser Beispiel diese SQL-Anweisung:**

```
SELECT &lt;sup&gt;*&lt;/sup&gt; FROM [Sales_By_Employee]
WHERE ([Market] = 'Tulsa';)
```

Diese Anweisung weist Excel an, aus der Tabelle *Sales_By_Employee* alle Datensätze abzurufen, bei denen die Spalte Market, in der das Vertriebsgebiet steht, den Wert Tulsa enthält.

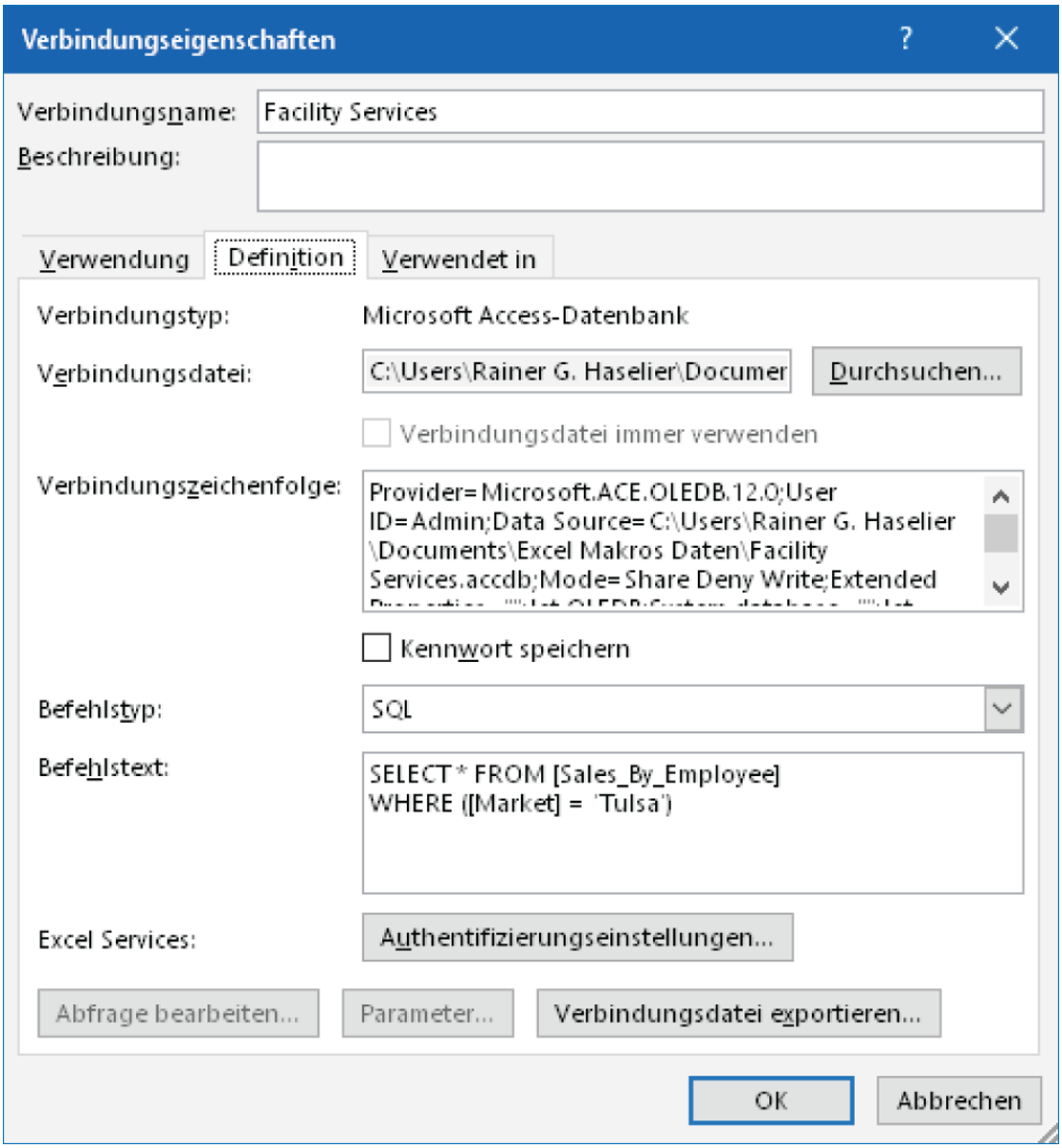


Abbildung 10.7: Legen Sie auf der Registerkarte DEFINITION den Befehlstyp auf »SQL« fest und geben Sie Ihre SQL-Anweisung ein.

5. **Klicken Sie auf OK, um die Änderungen zu bestätigen und das Dialogfeld Verbindungseigenschaften zu schließen.**

   Excel löst sofort eine Aktualisierung Ihrer externen Verbindung aus und ruft die neuen Daten ab.

Mit Makros dynamische Verbindungen erstellen

Vermutlich ist Ihnen aufgefallen, dass Sie in diesem Kapitel bis jetzt noch kein einziges Makro verwendet haben. Bisher haben Sie die Kriterien für Ihre Verbindung einfach hart-kodiert. In Abbildung 10.7 wird Tulsa beispielsweise direkt in der WHERE-Klausel der SQL-Anweisung angegeben. Dies führt natürlich dazu, dass die zurückgegebenen Daten immer Daten für Tulsa sind.

Wie aber gehen Sie vor, wenn Sie den Absatzmarkt auswählen und die SQL-Anweisung dynamisch ändern möchten, um auf Ihre Auswahl zu reagieren? Nun, Sie können ein wenig VBA verwenden, um die SQL-Anweisung interaktiv zu ändern. Führen Sie die folgenden Schritte aus:

1. **Legen Sie auf Ihrem Tabellenblatt eine Zelle fest, in der die dynamische Auswahl Ihrer Kriterien erfolgen soll.**

   In Abbildung 10.8 beispielweise kann der Benutzer den Absatzmarkt in Zelle C2 auswählen. In der Regel können Benutzer Kriterien entweder mit einer Combobox oder einer Liste zur Datenüberprüfung auswählen.

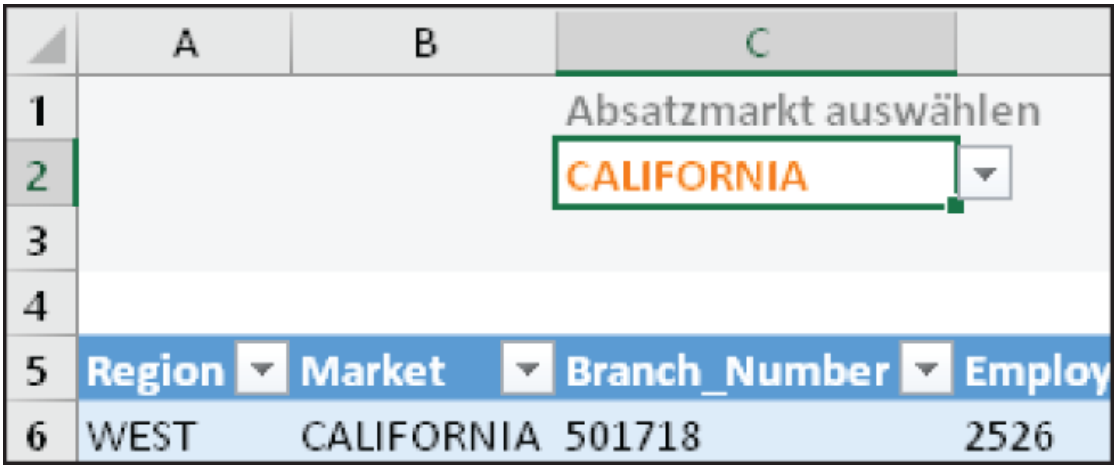


Abbildung 10.8: Legen Sie die Zelle fest, in der das Auswahlkriterium eingegeben werden soll.

2. **Öffnen Sie das Dialogfeld ARBEITSMAPPENVERBINDUNGEN, indem Sie auf der Registerkarte DATEN auf VERBINDUNGEN klicken. Oder öffnen Sie den Aufgabenbereich ABFRAGEN UND VERBINDUNGEN, indem Sie auf der Registerkarte DATEN auf ABFRAGEN UND VERBINDUNGEN klicken.**

   Merken Sie sich den Namen der Verbindung, die Sie dynamisch ändern wollen. In Abbildung 10.9 lautet der Verbindungsname *Facility Services*.

3. **Schließen Sie das Dialogfeld ARBEITSMAPPENVERBINDUNGEN bzw. den Aufgabenbereich ABFRAGEN UND VERBINDUNGEN und drücken Sie auf Ihrer Tastatur Alt + F11. Hierdurch wird der Visual Basic Editor geöffnet.**

4. **Wählen Sie im Menü des Visual-Basic-Editors den Befehl EINFüGEN | MODUL.**

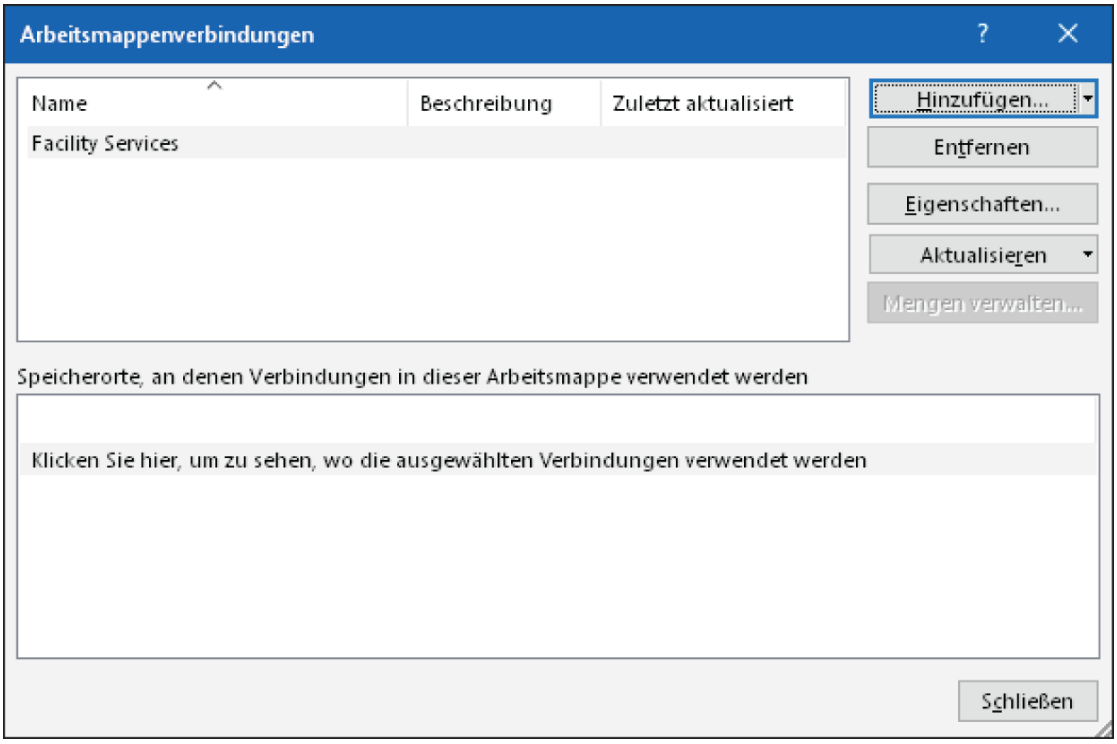


Abbildung 10.9: Merken Sie sich den Namen der Verbindung, in diesem Fall *Facility Services*.

5. **Geben Sie den folgenden Code in das neu erstellte Modul ein:**

```
Sub RefreshQuery()

ActiveWorkbook.Connections( _
 "Facility Services").OLEDBConnection.CommandText = _
 "SELECT * FROM [Sales_By_Employee] WHERE [Market] = '" &
_
 Range("C2").Value & "'"

ActiveWorkbook.Connections("Facility Services").Refresh

End Sub
```

   Dieser Code erstellt ein neues Makro mit dem Namen `RefreshQuery`. Dieses Makro verwendet die `Workbook.Connections`-Auflistung, um die Attribute der angegebenen Verbindung zu ändern. In diesem Fall möchten Sie die Eigenschaft `CommandText` der *Facility-Services*-Verbindung ändern.

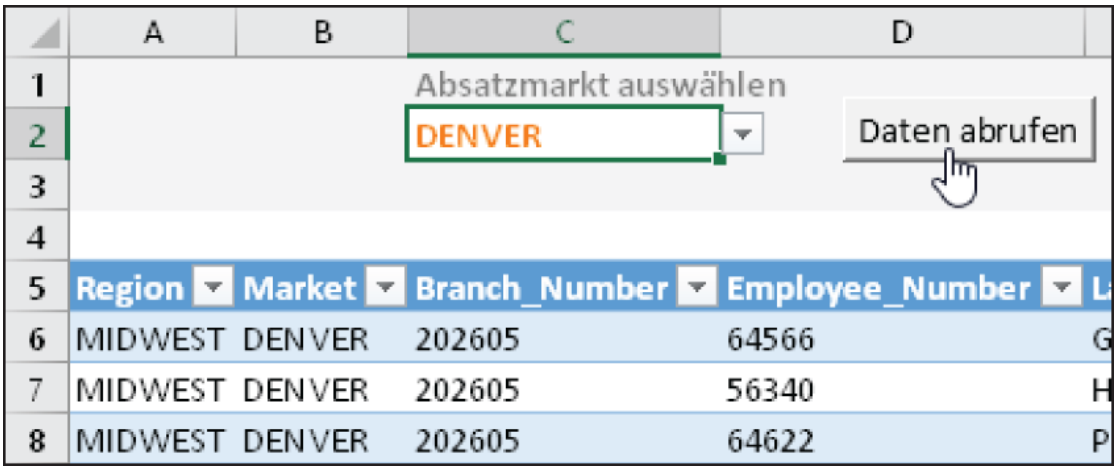
   Der Befehlstext ist im Grunde die SQL-Anweisung, die die Verbindung verwenden soll, wenn die Verbindung mit der Datenquelle hergestellt wird. In diesem Beispiel ruft der Befehlstext Zeilen aus der Tabelle `[Sales_By_Employee]` ab, wobei das Kriterium für das Feld `[Market]` auf den Wert in Zelle C2 festgelegt wird. Der Code aktualisiert dann die Verbindung zu *Facility Services*.

6. **Schließen Sie den Visual-Basic-Editor, und fügen Sie auf dem Tabellenblatt eine neue Befehlsschaltfläche ein. Klicken Sie dazu auf die Registerkarte ENTWICKLERTOOLS, öffnen Sie die Drop-down-Liste EINFüGEN, und fügen Sie ein Schaltflächen-Formularsteuerelement hinzu.**

7. **Weisen Sie der Befehlsschaltfläche das neu erstellte Makro `RefreshQuery` zu.**

   Wenn alles reibungslos verlaufen ist, verfügen Sie jetzt über einen schicken Mechanismus, der das dynamische Abrufen von Daten aus Ihrer externen Datenbank ermöglicht, und zwar

basierend auf von Ihnen angegebenen Kriterien (siehe Abbildung 10.10).



	A	B	C	D	
1			Absatzmarkt auswählen		
2			DENVER	Daten abrufen	
3					
4					
5	Region	Market	Branch_Number	Employee_Number	L
6	MIDWEST	DENVER	202605	64566	G
7	MIDWEST	DENVER	202605	56340	H
8	MIDWEST	DENVER	202605	64622	P

Abbildung 10.10: Sie haben nun ein einfach zu verwendendes Verfahren, um die externen Daten für einen bestimmten Absatzmarkt abzurufen.

Alle Verbindungen in einer Arbeitsmappe in einer Schleife bearbeiten

Sie können die `Workbook.Connections`-Auflistung dafür verwenden, um alle Verbindungsobjekte in einer Arbeitsmappe zu durchlaufen und deren Eigenschaften zu untersuchen oder zu ändern. Angenommen, Sie haben eine Arbeitsmappe mit mehreren externen Datenverbindungen. Sie können dann dieses Makro verwenden, um auf einem Tabellenblatt eine Liste aller Verbindungsobjekte in der aktuellen Arbeitsmappe mit den zugehörigen Verbindungszeichenfolgen und Befehlstexten zu erstellen:

```
Sub ListConnections()


'Schritt 1: Deklarieren Sie Ihre Variablen
Dim i As Long
Dim Cn As WorkbookConnection
```

```
'Schritt 2: Fügen Sie ein Tabellenblatt mit Überschriften für
'Verbindungsname, Verbindungszeichenfolge und Befehlstext ein
Worksheets.Add
With ActiveSheet.Range("A1:C1")
 .Value = Array("Verbindungsname", _
 "Verbindungszeichenfolge", "Befehlstext")
 .EntireColumn.AutoFit
End With

'Schritt 3: Durchlaufen Sie alle Verbindungen in einer Schleife
'und geben Sie die gewünschten Eigenschaften auf
'dem neu erstellten Tabellenblatt ein.
For Each Cn In ThisWorkbook.Connections
 i = i + 1

 Select Case Cn.Type
 Case Is = xlConnectionTypeODBC

 With ActiveSheet
 .Range("A1").Offset(i,0).Value=Cn.Name
 .Range("A1").Offset(i,1).Value=Cn.ODBCConnection.Connection
 .Range("A1").Offset(i,2).Value=Cn.ODBCConnection.CommandText
 End With

 Case Is = xlConnectionTypeOLEDB

 With ActiveSheet
 .Range("A1").Offset(i,0).Value=Cn.Name
 .Range("A1").Offset(i,1).Value=Cn.OLEDBConnection.Connection
 .Range("A1").Offset(i,2).Value=Cn.OLEDBConnection.CommandText
 End With

 End Select

Next Cn
End Sub
```

1. Schritt 1 deklariert zwei Variablen: eine Ganzzahlvariable, `i`, die sicherstellt, dass die Daten für jede Verbindungszeichenfolge in einer eigenen Zeile geschrieben werden, und ein `WorkbookConnection`-Objekt, `Cn`, das verwendet wird, um die gesuchten Eigenschaften verfügbar zu machen.

2. In Schritt 2 fügen Sie ein neues Tabellenblatt ein und ergänzen dort die Spaltenüberschriften für jede Verbindungseigenschaft, die Sie dokumentieren möchten.
3. Schritt 3 durchläuft alle Verbindungen in der Arbeitsmappe und schreibt die angegebenen Eigenschaften für jede Verbindung in das neue Tabellenblatt. Beachten Sie, dass Sie hierbei die Art der Verbindung überprüfen, die sich derzeit im Fokus befindet. Es gibt zwei Verbindungstypen: `xlConnectionTypeODBC` und `xlConnectionTypeOLEDB`. Da die Syntax zum Abrufen ihrer Eigenschaften etwas unterschiedlich ist, müssen Sie testen, mit welchem Typ Sie arbeiten.

Eine funktionsfähige Version dieses Makros finden Sie in der Beispieldatei *10.1 DynamischeDatenverbindung.xlsm* für dieses Kapitel.

ADO und VBA verwenden, um externe Daten abzurufen

Eine weitere Technik für die Arbeit mit externen Daten ist die Verwendung von VBA mit ADO (ActiveX Data Objects). Mit der Kombination aus ADO und VBA können Sie externe Datensätzen im Speicher bearbeiten. Dies ist nützlich, wenn Sie komplexe, mehrschichtige Prozeduren und Überprüfungen externer Datensätze durchführen müssen, Sie aber weder Arbeitsmappenverbindungen erstellen noch diese externen Datensätze in die Arbeitsmappe einfügen möchten.

Wenn Sie mit komplexen Excel-Arbeitsmappen arbeiten, die Daten aus externen Quellen abrufen, finden Sie regelmäßig (von anderen geschriebenen) Code, der ADO verwendet. Es ist wichtig, dass Sie die Grundlagen von ADO erkennen und verstehen, damit Sie mit dieser Art von Code umgehen können. Die folgenden Abschnitte stellen einige der grundlegenden ADO-Konzepte vor und zeigen Ihnen, wie Sie Ihre eigenen ADO-Prozeduren erstellen, um Daten abzurufen. Denken Sie daran, dass ADO-Programmierung ein umfangreiches Thema ist, das hier beim Weitem nicht vollständig behandelt werden kann. Wenn Sie feststellen, dass Sie in Ihrer Excel-Anwendung ausgiebig mit ADO und externen Daten arbeiten müssen, sollten Sie in ein oder mehrere Bücher investieren, die sich ausführlich mit diesem Thema beschäftigen.

ADO-Syntax verstehen

Wenn Sie versuchen, die Grundlagen von ADO zu verstehen, hilft es, sich ADO als ein Tool vorzustellen, das vor allem zwei Aufgaben angeht: Eine Verbindung mit einer Datenquelle herzustellen und das Dataset anzugeben, mit dem Sie arbeiten möchten. Im folgenden Abschnitt wird die grundlegende Syntax untersucht, die Sie kennen müssen, um genau das zu tun.

Die Verbindungszeichenfolge

Als Erstes müssen Sie eine Verbindung mit einer Datenquelle herstellen. Dazu müssen Sie in VBA ein paar Informationen bereitstellen. Diese Informationen werden in Form einer *Verbindungszeichenfolge* an VBA übergeben. Hier ist ein Beispiel für eine Verbindungszeichenfolge, die auf eine Access-Datenbank verweist:

```
"Provider=Microsoft.ACE.OLEDB.12.0;" & _
 "Data Source= C:\MyDatabase.accdb;" & _
 "User ID=Administrator;" & _
 "Password=AdminPassword"
```

Lassen Sie sich bitte nicht von der Syntax einschüchtern. Eine Verbindungszeichenfolge ist im Grunde nichts anderes als eine Textzeichenfolge, die eine Reihe von Variablen (auch Argumente genannt) enthält, die VBA verwendet, um eine Verbindung zu einer Datenquelle zu definieren und zu öffnen. Obwohl Verbindungszeichenfolgen zu Access oder Excel mit einer Vielzahl von Argumenten und Optionen ziemlich umfangreich sein können, gibt es nur eine Handvoll häufig verwendeter Argumente, auf die sich Anfänger von ADO konzentrieren können und sollten: Provider, Data Source, Extended Properties, User-ID und Password.

- **Provider:** Das Argument `Provider` teilt VBA mit, mit welcher Datenquelle Sie arbeiten. Wenn Access oder Excel als Datenquelle verwendet wird, lautet die Syntax: `Provider=Microsoft.ACE.OLEDB.12`.
- **Data Source**: Das Argument `Data Source` teilt VBA mit, wo sich die Datenbank oder Arbeitsmappe befindet, die die benötigten Daten enthält. Mit dem Argument `Data Source` übergeben Sie den vollständigen Pfad der Datenbank oder Arbeitsmappe. Beispiel: `Data Source=C:\MeinOrdner\MeineDatenbank.accdb`.
- **Extended Properties:** Das Argument `Extended Properties` wird in der Regel beim Herstellen einer Verbindung mit einer Excel-Arbeitsmappe verwendet. Dieses Argument teilt VBA mit, dass die Datenquelle etwas anderes als eine Datenbank ist. Wenn Sie mit einer Excel-Arbeitsmappe arbeiten, lautet dieses Argument: `Extended Properties=Excel 12.0`.
- **User ID:** Das Argument `User ID` ist optional und wird nur verwendet, wenn zum Herstellen einer Verbindung mit der Datenquelle eine Benutzer-ID angegeben werden muss: `User ID =MyUserId`.
- **Password:** Auch das Argument `Password` ist optional und wird nur verwendet, wenn ein Kennwort zum Herstellen einer

Verbindung mit der Datenquelle erforderlich ist: `Password=MyPassword`.

Nehmen Sie sich jetzt einen Moment Zeit, um die zuvor gezeigte Syntax erneut zu untersuchen. Sie können nun die verschiedenen Argumente in der Verbindungszeichenfolge identifizieren:

```
"Provider=Microsoft.ACE.OLEDB.12.0;" & _
"Data Source= C:\MyDatabase.accdb;" & _
"User ID=Administrator;" & _
"Password=AdminPassword"
```

Ein Recordset deklarieren

Zusätzlich zum Erstellen einer Verbindung mit Ihrer Datenquelle müssen Sie das Dataset definieren, mit dem Sie arbeiten wollen. In ADO wird dieses Dataset als Recordset bezeichnet. Ein `Recordset`-Objekt ist im Wesentlichen ein Container für die Datensätze und Felder, die von der Datenquelle zurückgegeben werden. Der am häufigsten verwendete Weg zum Definieren eines Recordset besteht darin, mit den folgenden Argumenten eine vorhandene Tabelle oder Abfrage zu öffnen:

```
Recordset.Open Source, ConnectString, CursorType, LockType
```

- Das Argument **Source** gibt die abzurufenden Daten an. Dies ist in der Regel eine Tabelle, eine Abfrage oder eine SQL-Anweisung, die Datensätze abruft.
- Das Argument **ConnectString** gibt die Verbindungszeichenfolge an, die zum Herstellen einer Verbindung mit der ausgewählten Datenquelle verwendet wird.
- Das Argument **CursorType** definiert, wie Sie sich durch die abgerufenen Daten bewegen können. Für das Abrufen externer Daten in Excel verenden Sie für dieses Argument die Einstellung `adOpenForwardOnly`. Dieser Cursortyp ist der am meisten effiziente, da er vorschreibt, dass Sie sich nur in einer Richtung durch das Recordset bewegen dürfen, und zwar vom Anfang zum Ende. Dies ist für Berichte ideal, da hier die Daten lediglich abgerufen werden müssen.

- Mit dem Argument **LockType** geben Sie an, ob die vom Recordset zurückgegebenen Daten geändert werden können. Dieses Argument wird üblicherweise auf `adLockReadOnly` gesetzt (der Standardwert), um anzugeben, dass es nicht erforderlich ist, die zurückgegebenen Daten zu bearbeiten. Alternativ können Sie dieses Argument auf `adLockOptimistic` setzen, wodurch Sie die zurückgegebenen Daten auch verändern können, allerdings *nicht* die Daten in der Quelldatenbank.

Das folgende Beispiel zeigt die Syntax zur Deklaration eines Recordsets, das die Tabelle `Products` öffnet:

```
MyRecordset.Open "Products", _
MyConnection, adOpenForwardOnly, adLockReadOnly
```

ADO in einen Makro verwenden

Nachdem Sie nun diese einfachen ADO-Grundlagen kennengelernt haben, sind Sie bereit, Ihre eigene ADO-Prozedur zu erstellen. Bevor Sie jedoch etwas mit ADO tun, müssen Sie zuerst einen Verweis auf die ADO-Objektbibliothek festlegen. Genauso wie Excel über einen eigenen Satz von VBA-Objekten, -Eigenschaften und -Methoden verfügt, gilt dies auch für ADO. Da Excel das ADO-Objektmodell nicht von Natur aus kennt, müssen Sie Excel auf die ADO-Bibliothek verweisen.

1. **Erstellen Sie eine neue Arbeitsmappe und öffnen Sie den Visual-Basic-Editor (VBE).**
2. **Wenn der VBE angezeigt wird, wählen Sie im Menü den Befehl EXTRAS | VERWEISE. Hierdurch wird das Dialogfeld VERWEISE geöffnet, das Sie in Abbildung** 10.11 sehen.
3. **Führen Sie einen Bildlauf nach unten durch, bis Sie die neueste Version der MICROSOFT ACTIVEX DATA OBJECTS LIBRARY sehen.**

Es ist normal, dass im Dialogfeld VERWEISE mehrere Versionen derselben Bibliothek angezeigt werden. Im Allgemeinen ist es am besten, die aktuell verfügbare Version auszuwählen. Beachten Sie, dass Versionen nach 2.8 als MICROSOFT ACTIVEX DATA OBJECTS RECORDSET LIBRARY bezeichnet werden.

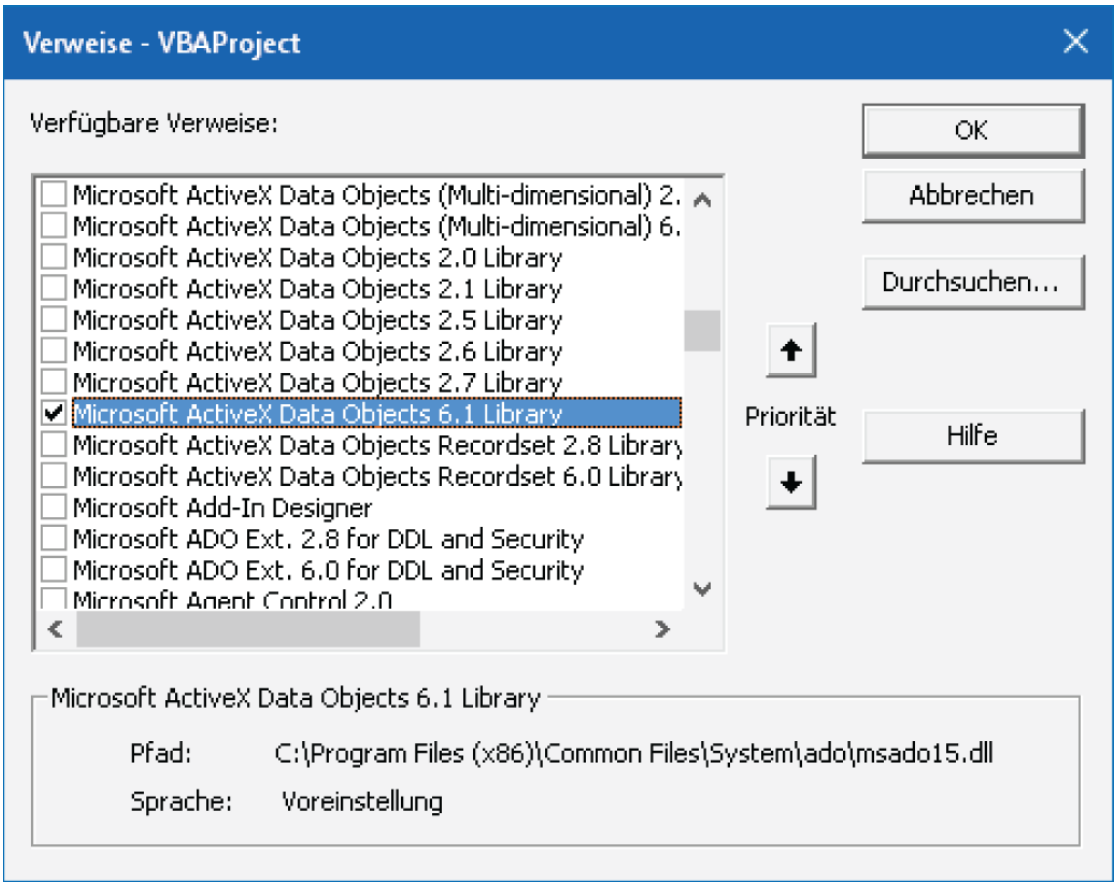


Abbildung 10.11: Wählen Sie die aktuelle Version der Microsoft ActiveX Data Objects Library aus.

4. **Klicken Sie auf OK, um Ihre Auswahl zu bestätigen.**

Sie können das Dialogfeld VERWEISE erneut öffnen, um sicherzustellen, dass der gewünschte Verweise festgelegt ist. Sie wissen, dass Ihre Auswahl wirksam wurde, wenn die MICROSOFT ACTIVEX DATA OBJECTS LIBRARY oben im Dialogfeld VERWEISE mitsamt aktiviertem Kontrollkästchen angezeigt wird.

Die Verweise, die Sie in einer bestimmten Arbeitsmappe oder Datenbank festlegen, werden nicht auf Anwendungsebene angewendet. Dies bedeutet, dass Sie diese Schritte für jeder neue Arbeitsmappe oder Datenbank wiederholen müssen, die Sie erstellen.

Jetzt können Sie das, was Sie soeben gelernt haben, in einem Makro anwenden. Im folgenden Beispiel wird ADO verwendet, um eine Verbindung mit einer Access-Datenbank herzustellen und die Tabelle *Products* abzurufen:

```
Sub GetAccessData()

'Schritt 1: Deklarieren Sie Ihre Variablen
 Dim MyConnect As String
 Dim MyRecordset As ADODB.Recordset


'Schritt 2: Definieren Sie die Verbindungszeichenfolge und
'öffnen Sie das Recordset
 MyConnect = "Provider=Microsoft.ACE.OLEDB.12.0;" & _
 "Data Source= C:\MeinOrdner\MeineDatenbank.accdb"

 Set MyRecordset = New ADODB.Recordset

 MyRecordset.Open "Products", _
 MyConnect, adOpenStatic, adLockReadOnly

'Schritt 3: Daten aus dem Recordset in das Tabellenblatt kopieren
 Sheets("MeinBlatt").Range("A2").CopyFromRecordset _
 MyRecordset


'Schritt 4: Datenbeschriftungen hinzufügen
 With ActiveSheet.Range("A1:C1")
 .Value = Array("Produkt", "Beschreibung", "Kategorie")
 .EntireColumn.AutoFit
 End With

End Sub
```

1. Schritt 1 deklariert zwei Variablen: eine Zeichenfolgenvariable, die die Verbindungszeichenfolge enthält, und ein `Recordset`-

Objekt, um die Ergebnisse des Datenabrufs zu speichern. In diesem Beispiel enthält die Variable `MyConnect` die Verbindungszeichenfolge, die die Datenquelle identifiziert. Die Variable mit dem Namen `MyRecordset` wird die von der Prozedur zurückgegebenen Daten enthalten.

2. In Schritt 2 definieren Sie die Verbindungszeichenfolge für die ADO-Prozedur. In diesem Szenario stellen Sie eine Verbindung mit der Datei *MeineDatenbank.accdb* her, die sich im Verzeichnis *C:\MeinOrdner* befindet. Nachdem Sie Ihre Datenquelle definiert haben, können Sie Ihr Recordset öffnen und `MyConnect` verwenden, um die statischen, schreibgeschützten Daten abzurufen.
3. Schritt 3 verwendet die Excel-Methode `CopyFromRecordset`, um die Daten aus dem Recordset in das Tabellenblatt zu übertragen. Diese Methode benötigt zwei Informationen: den Ort, an dem die Daten ausgegeben werden sollen, sowie das `Recordset`-Objekt, das die Daten enthält. In diesem Beispiel kopieren Sie die Daten aus dem `MyRecordset`-Objekt auf das Blatt `MeinBlatt` (beginnend bei Zelle A2).
4. Es ist wichtig zu wissen, dass die Methode `CopyFromRecordset` keine Spaltenüberschriften beziehungsweise Feldnamen zurückgibt. Dadurch wird eine letzte Aktion (Schritt 4) erforderlich, bei der Sie Spaltenüberschriften hinzufügen, indem Sie diese einfach in einem Array definieren und in das aktive Blatt schreiben.

Mit ADO und VBA können Sie alle erforderlichen Bestandteil in einen aufgeräumten Makro zusammenführen und sie danach einfach vergessen. Solange sich die in Ihrem Code definierten Variablen (die Datenquelle, das Recordset und der Ausgabeort) nicht ändern, benötigen Ihre auf ADO basierten Prozeduren so gut wie keine Wartung.

Mit Textdateien arbeiten

VBA enthält eine Anzahl von Anweisungen, die es Ihnen erlauben, direkt mit Dateien zu arbeiten. Mit diesen Ein-/Ausgabeanweisungen haben Sie viel mehr Kontrolle über Dateien als mit den normalen Import- und Exportanweisungen, die Excel zur Verfügung stellt.

Eine Textdatei öffnen

Die VBA-Anweisung `Open` (nicht mit der Methode `Open` des `Workbooks`-Objekts zu verwechseln) öffnet eine Datei zum Lesen oder Schreiben. Bevor Sie aus einer Datei lesen oder in eine Datei schreiben können, müssen Sie sie natürlich erst einmal öffnen.

Die `Open`-Anweisung von VBA verfügt über einen eigenen Satz von Argumenten (im Folgenden in Fettdruck dargestellt). Hier ist die erforderliche Syntax für die `Open`-Anweisung:

```
Open [Pfadname] For [Modus] As [#]Dateinummer
```

- **Pfadname:** Das Argument `Pfadname` der `Open`-Anweisung ist recht klar. Es enthält einfach den Namen und den optionalen Pfad der Datei, die geöffnet werden soll.
- **Modus:** Im Teil `Modus` der `Open`-Anweisung legen Sie fest, ob und wie die Datei bearbeitet oder in die Datei geschrieben werden kann. Dieses Argument ist ein erforderliches Feld, das eines der folgenden Schlüsselwörter enthalten kann:

  Append: Mit diesem Zugriffsmodus kann die Datei gelesen und es können Daten an das Ende der Datei angehängt werden.

  Input: Dieser Modus ermöglicht das Lesen aus, aber nicht das Schreiben in die Datei.

  Output: Dieser Modus erstellt immer eine neue Datei, aus der gelesen und in die geschrieben werden kann.

  Binary: Ein wahlfreier Zugriffsmodus, der es erlaubt, byteweise aus der Datei zu lesen und in sie zu schreiben.

Random: Der wahlfreie Zugriff wird verwendet, um Daten in Einheiten bestimmter Größe zu lesen oder zu schreiben. Meist handelt es sich bei den Blöcken um Datensätze. Die Größe der Datensätze wird durch das optionale Argument `Reclength` der `Open`-Anweisung bestimmt.

- **Dateinummer:** Dieses erforderliche Argument ist eine Dateinummer im Bereich von 1 bis 511. Sie können die Funktion `FreeFile` verwenden, um die nächste verfügbare Dateinummer zu erhalten.

Im folgenden Beispiel verwenden Sie die `Open`-Anweisung, um eine Textdatei namens *MeineDatei* als schreibgeschützte Datei zu öffnen, die die Dateinummer #1 besitzt:

```
Open "MeineDatei.txt" For Input As #1
```

Aus der geöffneten Textdatei lesen und in die Datei schreiben

Nachdem eine Textdatei geöffnet wurde, wollen Sie in der Regel die Datei lesen oder in die Datei schreiben.

Es gibt drei Möglichkeiten, mit denen Sie VBA anweisen können, eine Textdatei zu lesen. Diese Anweisungen werden zum Lesen von Daten aus einer sequenziellen Textdatei verwendet:

- **Input:** Liest eine angegebene Anzahl von Zeichen aus einer Datei.
- **Input #:** Liest Daten als eine Reihe von Variablen, wobei die Variablen durch ein Komma getrennt sind.
- **Line Input #:** Liest eine komplette Datenzeile (die mit einem Wagenrücklaufzeichen, einem Zeilenvorschubzeichen oder beidem abgeschlossen ist).

Zwei Anweisungen werden zum Schreiben von Daten in die geöffnete Textdatei verwendet:

- ✔ **Write #:** Schreibt eine Reihe von Werten, wobei jeder Wert in Anführungszeichen eingeschlossen ist und die einzelnen Werte durch ein Komma voneinander getrennt sind.
- ✔ **Print #:** Schreibt eine Reihe von Werten, wobei die einzelnen Werte durch ein Tabulatorzeichen voneinander getrennt sind.

Praktisches Beispiel: Excel-Nutzung in einer Textdatei protokollieren

Wenn Sie jetzt völlig verwirrt sind, ist dies kein Grund zur Sorge. Die Dinge sollten mit einem praktischen Beispiel klarer werden. Angenommen, Sie möchten protokollieren, wann Excel gestartet wird. Sie können hierzu ein Makro erstellen, dass jedes Mal, wenn Excel geöffnet oder geschlossen wird, Datum und Uhrzeit in eine Textdatei schreibt.

Damit dieses Beispiel zuverlässig funktioniert, muss sich das Makro in einer Arbeitsmappe befinden, die bei jedem Start von Excel geöffnet wird. Das Speichern des Makros in Ihrer persönlichen Makroarbeitsmappe ist eine ausgezeichnete Wahl.

Die persönliche Makroarbeitsmappe wird bei jedem Start von Excel automatisch geladen. Im Projekt-Explorer des Visual-Basic-Editors erkennen Sie diese Mappe am Dateinamen *personal.xlsb*.

1. **Aktivieren Sie den Visual-Basic-Editor, indem Sie auf Ihrer Tastatur** Alt **+** F11 **drücken.**
2. **Suchen Sie im Projekt-Explorer nach PERSONAL.XLSB. Klicken Sie auf das Pluszeichen neben dem Namen, damit alle Tabellenblätter im Projekt-Explorer angezeigt werden.**
3. **Klicken Sie auf DIESEARBEITSMAPPE.**
4. **Wählen Sie in der Drop-down-Liste mit den Ereignissen das Ereignis OPEN aus.**
5. **Tippen Sie den folgenden Code ein oder kopieren Sie ihn dorthin:**

```
Private Sub Workbook_Open()
 Open Application.DefaultFilePath & "\excelnutzung.txt" _
 For Append As #1
 Print #1, "Gestartet " & Now
 Close #1
End Sub
```

Eine funktionsfähige Version dieses Makros finden Sie in den Begleitdateien zum Buch in der Datei *Excel Nutzung Protokoll.xlsm*.

Dieses Makro öffnet zunächst eine Textdatei mit dem Namen *excelnutzung.txt* in Ihrem Windows-Verzeichnis *Dokumente* (`Application.DefaultFilePath`). Wenn die Textdatei nicht vorhanden ist, erstellt Excel sie. Sobald die Textdatei geöffnet ist, fügt dieses Makro eine neue Zeile an, die das aktuelle Datum und die aktuelle Uhrzeit enthält und ungefähr so aussieht:

```
Gestartet 29.10.2020 09:51:00
```

Praktisches Beispiel: Textdatei in einen Bereich importieren

Das Beispiel in diesem Abschnitt liest die Datei *TextDatei.csv* (die Sie in den Begleitdateien zum Buch finden) ein und fügt dann die Werte ab der aktiven Zelle in das aktive Arbeitsblatt ein. Der Code liest die einzelnen Zeichen ein und parst dabei die Datenzeilen; Anführungszeichen werden ignoriert und es wird nach Kommas gesucht, um die Spalten voneinander zu trennen.

Eine funktionsfähige Version dieses Makros finden Sie in den Begleitdateien zum Buch in der Datei *Importiere Textdatei.xlsm*.

```
Sub ImportRange()
 Dim ImpRng As Range
 Dim Filename As String
 Dim r As Long, c As Integer
 Dim txt As String, Char As String * 1
```

```
 Dim Data
 Dim i As Integer

 Set ImpRng = ActiveCell
 On Error Resume Next
 Filename = ThisWorkbook.Path & "\textdatei.csv"
 Open Filename For Input As #1
 If Err &lt;&gt; 0 Then
 MsgBox "Datei nicht gefunden: " & Filename, _
 vbCritical, "FEHLER"
 Exit Sub
 End If
 r = 0
 c = 0
 txt = ""
 Application.ScreenUpdating = False
 Do Until EOF(1)
 Line Input #1, Data
 For i = 1 To Len(Data)
 Char = Mid(Data, i, 1)
 If Char = "," Then 'Komma
 ActiveCell.Offset(r, c) = txt
 c = c + 1
 txt = ""
 ElseIf i = Len(Data) Then 'Zeilenende
 If Char &lt;&gt; Chr(34) Then txt = txt & Char
 ActiveCell.Offset(r, c) = txt
 txt = ""
 ElseIf Char &lt;&gt; Chr(34) Then
 txt = txt & Char
 End If
 Next i
 c = 0
 r = r + 1
 Loop
 Close #1
 Application.ScreenUpdating = True
End Sub
```

Das obige Beispiel funktioniert mit den meisten Daten, jedoch hat es einen Schönheitsfehler: Es kann nicht mit Daten umgehen, die ein Komma oder ein Anführungszeichen enthalten. Kommas, die Ergebnisse der Formatierung sind, werden richtig behandelt (sie werden ignoriert). Darüber

hinaus werden importierte Datumswerte von Kreuzzeichen eingeschlossen, zum Beispiel `#2020-05-12#`.

Teil V

Der Top-Ten-Teil

Unter www.fuer-dummies.de finden Sie noch mehr Bücher für Dummies!

IN DIESEM TEIL …

- ✔ Sehen Sie sich ein paar Tricks an, mit denen Sie den Visual-Basic-Editor effizienter verwenden können
- ✔ Entdecken Sie ein paar Debugging-Tipps, um Fehler in Ihrem VBA-Code zu vermeiden
- ✔ Suchen Sie im Excel-Hilfesystem effizient nach Ratschlägen zu VBA-Themen
- ✔ Lernen Sie einige Online-Ressourcen kennen, mit denen Sie Ihre Makrofähigkeiten noch weiter verbessern können

Kapitel 11

Zehn nützliche Tipps für den Visual-Basic-Editor

IN DIESEM KAPITEL

- Kommentarblöcke verwenden
- Mehrere Codezeilen auf einen Rutsch kopieren
- Zwischen Modulen und Prozeduren wechseln
- Beamen Sie sich zu Ihren Funktionen
- In der richtigen Prozedur bleiben
- Den Code schrittweise (bis zu einer bestimmten Zelle) ausführen
- Die Codeausführung an einem bestimmten Punkt anhalten
- Den Anfang und das Ende des Variableninhalts ansehen
- Automatische Syntaxüberprüfung ausschalten

Da Sie ja nun einige Zeit mit der Erstellung von Makros im Visual-Basic-Editor (VBE) verbringen, sollten Sie die Vorteile einiger eingebauter Werkzeuge nutzen, mit denen Sie sich das Leben erleichtern können. Gleichgültig, ob Sie ein Datenanalyst sind, der neu in die Makro-Programmierung einsteigt, oder ob Sie bereits Erfahrungen mit Excel-Makros haben: Diese Tipps können Ihnen das Leben als Makro-Programmierer erheblich vereinfachen.

Kommentarblöcke verwenden

Wenn Sie am Anfang einer Codezeile ein Hochkomma eingeben, erstellen Sie eine Kommentarzeile und weisen Excel an, diese

Zeile bei der Codeausführung zu ignorieren. Diese Technik wird *Code auskommentieren* genannt. Die meisten Programmierer verwenden das Hochkomma, um im Code Kommentare oder Notizen einzugeben, wie es Abbildung 11.1 zeigt.

```
'Schritt 1:  Deklarieren Sie Ihre Variablen
    Dim ws As Worksheet

'Schritt 2: Fehlermeldung unterdrücken, falls keine Formeln gefunden wurden
    On Error Resume Next

'Schritt 3:  Die einzelnen Tabellenblätter in einer Schleife durchlaufen
    For Each ws In ActiveWorkbook.Worksheets

'Schritt 4:  Zellen mit Formeln markieren und hervorheben
    With ws.Cells.SpecialCells(xlCellTypeFormulas)
    .Interior.ColorIndex = 36
    End With

'Schritt 5: Nächstes Tabellenblatt holen
    Next ws
```

Abbildung 11.1: Ein Hochkomma am Anfang einer Zeile wandelt diese Zeile in einen Kommentar um.

Manchmal ist es nützlich, in einem Rutsch mehrere Codezeilen auszukommentieren. Sie können so bestimmte Codezeilen testen und Excel gleichzeitig anweisen, die auskommentierten Zeilen zu ignorieren.

Anstatt jede Zeile einzeln mit einem Hochkomma zu versehen, können Sie auch die Symbolleiste BEARBEITEN verwenden, um ganze Codeblöcke auszukommentieren.

Sie können die Symbolleiste BEARBEITEN einblenden, indem Sie im Menü des VBE den Befehl ANSICHT | SYMBOLLEISTEN | BEARBEITEN wählen. Markieren Sie die Zeilen, die Sie auskommentieren wollen, und klicken Sie dann auf der Symbolleiste BEARBEITEN auf BLOCK AUSKOMMENTIEREN (siehe Abbildung 11.2).

Damit die Symbolleiste BEARBEITEN immer sichtbar ist, können Sie sie Richtung VBE-Menü ziehen. Die Symbolleiste

verankert sich dann an der von Ihnen gewählten Stelle.

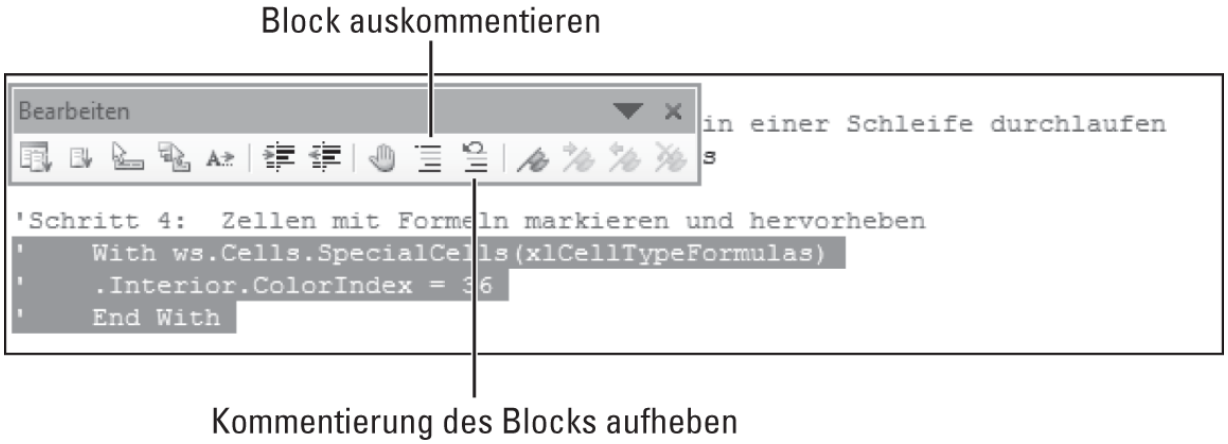


Abbildung 11.2: Verwenden Sie die Symbolleiste BEARBEITEN, wenn Sie einen Block auskommentieren wollen, in dem allen Zeilen der Markierung ein Hochkomma vorangestellt wird.

Mehrere Codezeilen auf einen Rutsch kopieren

Sie können ganze Codeblöcke kopieren, indem Sie die gewünschten Zeilen markieren und die Strg -Taste auf Ihrer Tastatur drücken, während Sie den Block ziehen. Dies ist ein alter Windows-Trick, der auch dann funktioniert, wenn Sie den markierten Code in ein anderes Modul ziehen.

Wenn sich neben dem Mauszeiger ein Plussymbol befindet, können Sie erkennen, dass durch das Ziehen eine Kopie erstellt wird (siehe Abbildung 11.3).

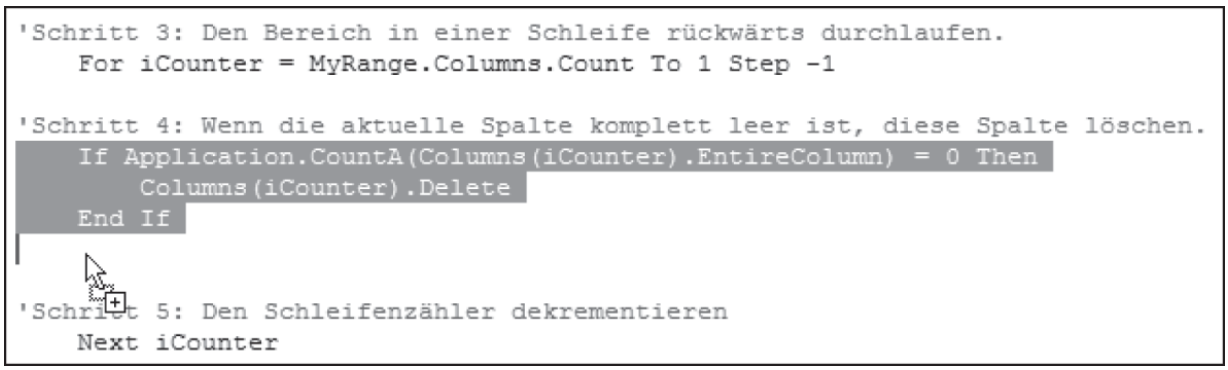


Abbildung 11.3: Wenn Sie beim Ziehen die Strg -Taste gedrückt halten, erstellen Sie eine Kopie an der Stelle, auf die der Mauszeiger zeigt.

Zwischen Modulen und Prozeduren springen

Wenn der Umfang Ihrer Codesammlung immer größer wird, ist es etwas umständlich, schnell zwischen Modulen und Prozeduren zu wechseln. Ein paar Tastenkombinationen machen Ihnen das Leben leichter:

- ✔ Drücken Sie Strg+⇥, um schnell zwischen Modulen zu wechseln.
- ✔ Drücken Sie Strg+Bild ⇧ und Strg+Bild ⇩, um zwischen den Prozeduren eines Moduls zu wechseln.

Beamen Sie sich zu Ihren Funktionen

Während Sie an einem Makro arbeiten, treffen Sie manchmal auf den Namen einer Variablen oder einer Funktion, die auf ein anderes Stück Code zeigt. Anstatt alle Module zu durchsuchen, um die Stelle zu finden, an der die Funktion oder Variable definiert ist, können Sie auch einfach den Mauszeiger auf den Namen der Funktion/Variablen setzen und dann ⇧ + F2 drücken.

Wie Abbildung 11.4 zeigt, werden Sie automatisch an den Ursprung der Funktion beziehungsweise Variablen gebeamt.

```
Error GoTo ServerFail
ets("ReferenceTables").Visible = tue
 = "SELECT Version FROM RefVersionCheck"
ordset.Open MySql, MyConnection, adOpenstatic, adlock



st MyConnection As String = "Provider=seqoledb; " & _
 "Data Source=prdss2kic004,1433; " & _
 "database=sops;" & _
 "User ID=sopsuser;" & _
 "Password=myPassword;"
```

Abbildung 11.4: Drücken Sie ⇧+F2, wenn der Cursor auf dem Namen einer Variablen oder Funktion steht, um zu der Stelle zu gelangen, an der sie definiert ist.

Drücken Sie ⇧+Strg+F2, um zur Ausgangsstelle zurückzukehren.

In der richtigen Prozedur bleiben

Wenn Ihre Module zahlreiche Prozeduren enthalten, kann es schwierig werden, durch eine bestimmte Prozedur zu scrollen, ohne dabei versehentlich in den Code einer anderen Prozedur zu geraten. Deswegen werden Sie vermutlich oft hektisch nach oben oder nach unten scrollen, während Sie versuchen, das richtige Stück Code zu finden.

Damit dies nicht passiert, können Sie unten links im Editorfenster die Schaltfläche PROZEDURANSICHT verwenden, wie es Abbildung 11.5 zeigt. Nach einem Klick auf diese kleine Schaltfläche wird der Bildlauf auf die aktuelle Prozedur eingeschränkt.

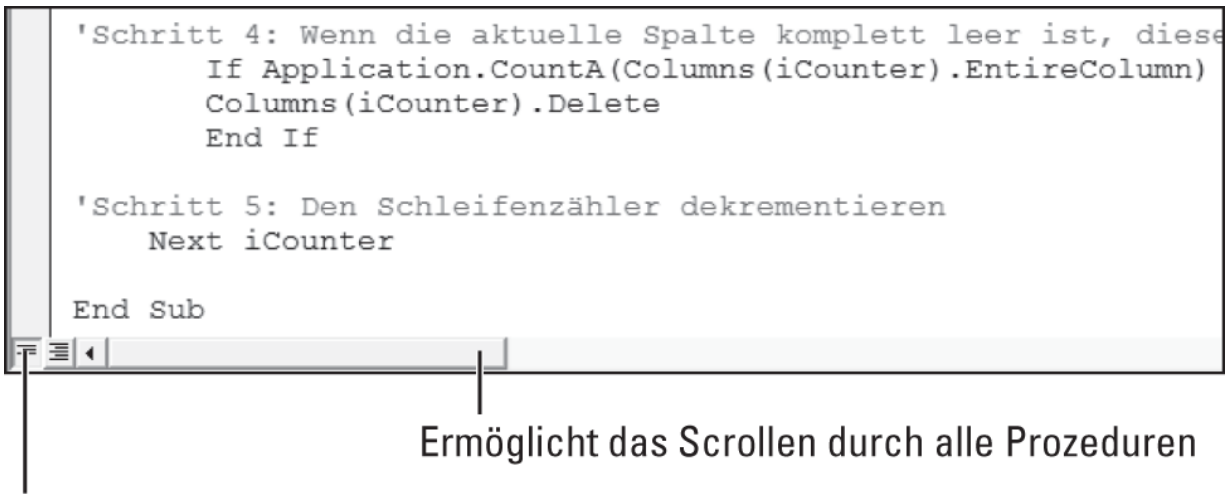


Abbildung 11.5: Verwenden Sie die Schaltfläche »Prozeduransicht«, um den Bildlauf auf die aktive Prozedur zu beschränken.

Den Code schrittweise ausführen

Der Visual-Basic-Editor enthält einige Werkzeuge, mit denen Sie Ihren Code debuggen können. Bei der Programmierung bedeutet *Debugging* das Finden und Beseitigen von Fehlern (»Bugs«) in Ihrem Code.

Eines der nützlichsten Werkzeuge beim Debugging ist die Möglichkeit, den Code zeilenweise ausführen zu lassen. Bei der zeilenweisen Ausführung können Sie genau sehen, was passiert, wenn eine bestimmte Codezeile ausgeführt wird.

Um den Code schrittweise ausführen zu lassen, setzen Sie die Einfügemarke an eine beliebige Stelle im Makro und drücken dann auf Ihrer Tastatur F8. Hierdurch wird der Debugmodus für das Makro aktiviert.

Die erste Codezeile des Makros wird hervorgehoben, und in der Markierungsleiste an der linken Seite des Editorfensters wird ein gelber Pfeil angezeigt (siehe Abbildung 11.6). Drücken Sie erneut F8, um die hervorgehobene Codezeile auszuführen; danach

wird die nächste Codezeile markiert. Drücken Sie so oft F8, bis Sie an der letzten Codezeile des Makros angekommen sind.

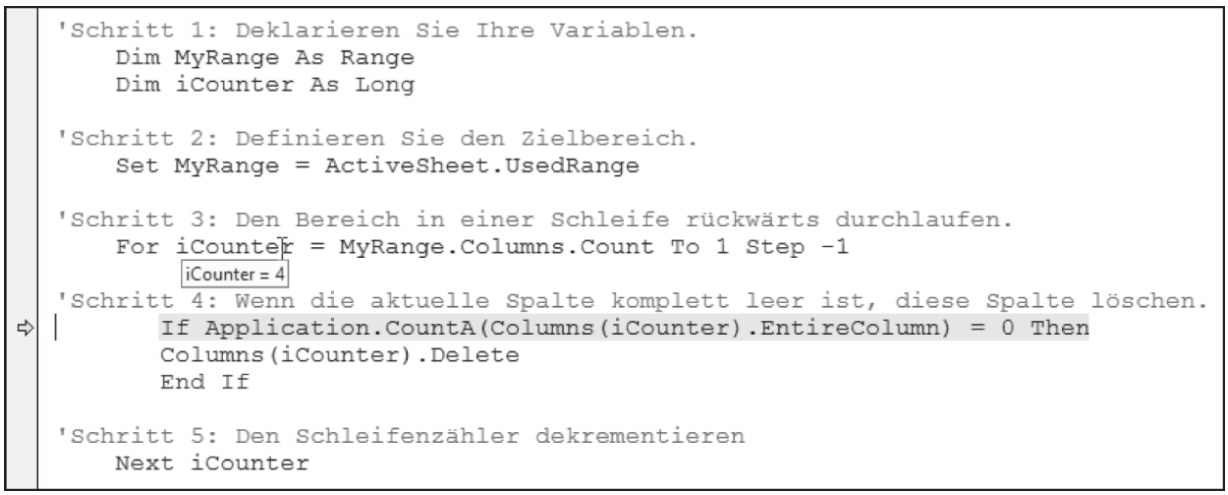


Abbildung 11.6: Drücken Sie auf Ihrer Tastatur F8, um das Makro schrittweise ausführen zu lassen.

Während Sie den Code schrittweise ausführen lassen, können Sie den Mauszeiger auf den Namen einer String- oder Integer-Variablen bewegen, damit der aktuelle Wert dieser Variablen angezeigt wird.

Um den Debugmodus zu beenden, wählen Sie im Menü von VBE DEBUGGEN | PROZEDUR ABSCHLIESSEN.

Zu einer bestimmten Zeile in Ihrem Code springen

Im vorigen Beispiel haben Sie gesehen, wie Sie Ihren Code schrittweise ausführen lassen können: Setzen Sie die Einfügemarke an eine beliebige Stelle im Code und drücken Sie F8. Der Debugmodus wird aktiviert, die erste Codezeile wird hervorgehoben und an der linken Seite des Codefensters erscheint ein kleiner gelber Pfeil.

Das ist prima, was aber, wenn die schrittweise Codeausführung an einer anderen Stelle beginnen soll? Auch das ist ganz einfach: Ziehen Sie dazu den Pfeil einfach vor die gewünschte Codezeile.

Wenn im Debugmodus eine Codezeile hervorgehoben wird, können Sie den Pfeil in der Randspalte nach oben oder unten ziehen und so festlegen, welche Zeile als nächste ausgeführt werden soll (siehe Abbildung 11.7).

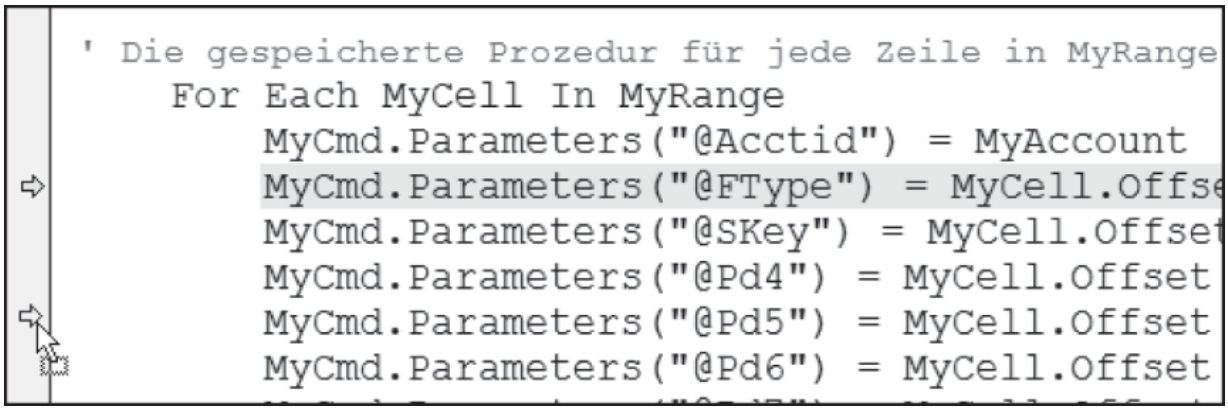


Abbildung 11.7: Ziehen Sie den Pfeil, während Ihr Code schrittweise ausgeführt wird.

Die Ausführung Ihres Codes an einer bestimmten Stelle unterbrechen

Ein weiteres nützliches Debuggingwerkzeug ist die Möglichkeit, in Ihrem Code Haltepunkte festzulegen. Wenn Sie einen Haltepunkt definieren, wird Ihr Code so lange normal ausgeführt, bis die Codeausführung an der Zeile ankommt, die Sie als Haltepunkt definiert haben.

Diese Debuggingtechnik ist nützlich, wenn Sie nur kleinere Codefragmente testen und untersuchen wollen. Angenommen, Sie haben ein Makro, bei dem Sie vermuten, dass sich an irgendeiner Stelle ein Fehler eingeschlichen hat, der größte Teil des Makros dagegen fehlerfrei abläuft. Sie können dann an der Codestelle, an der Sie den Fehler vermuten, einen Haltepunkt setzen und das Makro starten. Wenn die Makroausführung den Haltepunkt erreicht, wird die Ausführung unterbrochen und Sie können das Makro durch Drücken von F8 ab dieser Stelle schrittweise ausführen lassen.

Um einen Haltepunkt festzulegen, setzen Sie die Einfügemarke in die gewünschte Codezeile und drücken dann F9. Wie Abbildung 11.8 zeigt, wird in der Randspalte des Editorfensters ein roter Punkt angezeigt und die Codezeile selbst dunkelrot unterlegt.

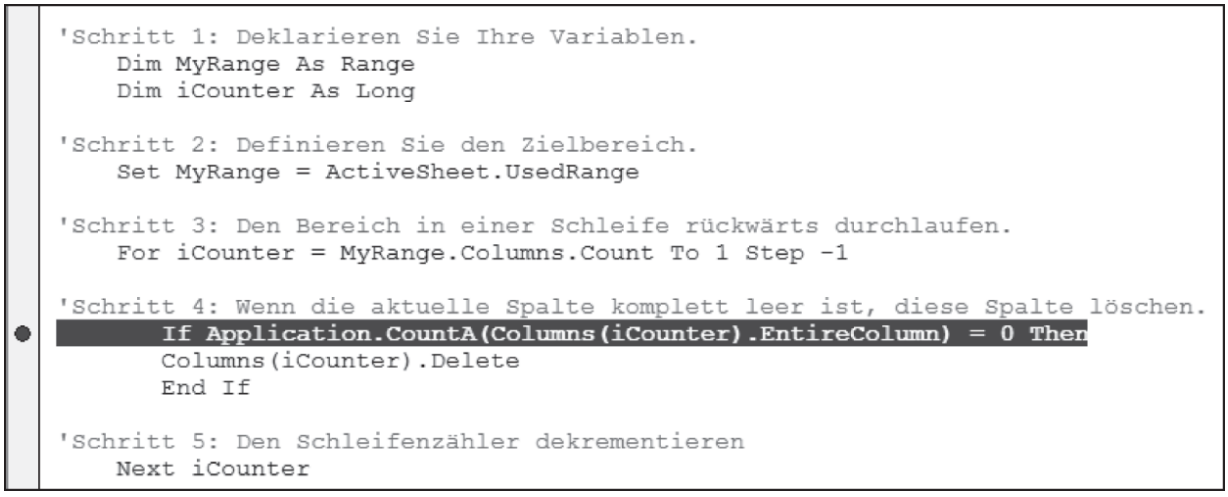


Abbildung 11.8: Ein Haltepunkt wird durch einen roten Punkt in der Markierungsleiste an der linken Seite des Editorfensters markiert. Außerdem wird diese Codezeile dunkelrot unterlegt.

Wenn die Makroausführung auf einen Haltepunkt trifft, wird automatisch der Debugmodus aktiviert. Um den Debugmodus zu beenden, wählen Sie im Menü von VBE DEBUGGEN | PROZEDUR ABSCHLIESSEN.

Den Anfang und das Ende von Variablenwerten anzeigen

Wenn Sie im Visual-Basic-Editor auf den Namen einer Variablen des Typs `String` oder `Integer` zeigen, während der Debugmodus aktiv ist, wird der Wert der Variablen in einer QuickInfo angezeigt. Hierdurch können Sie sich Änderungen am Wert einer Variablen ansehen und so Ihren Makro-Code einfacher debuggen.

Jedoch ist die Länge des Texts in einer QuickInfo auf 77 Zeichen begrenzt; dies schließt den angezeigten Namen der Variablen ein. Falls der Wert der Variablen mehr Platz braucht ist, wird er abgeschnitten.

Falls Sie mehr als die ersten 77 Zeichen sehen wollen, drücken Sie die Strg-Taste, während Sie auf den Variablennamen zeigen.

Abbildung 11.9 zeigt, wie die QuickInfo aussieht, während Sie bei aktivem Debugmodus auf den Namen einer Variablen zeigen.

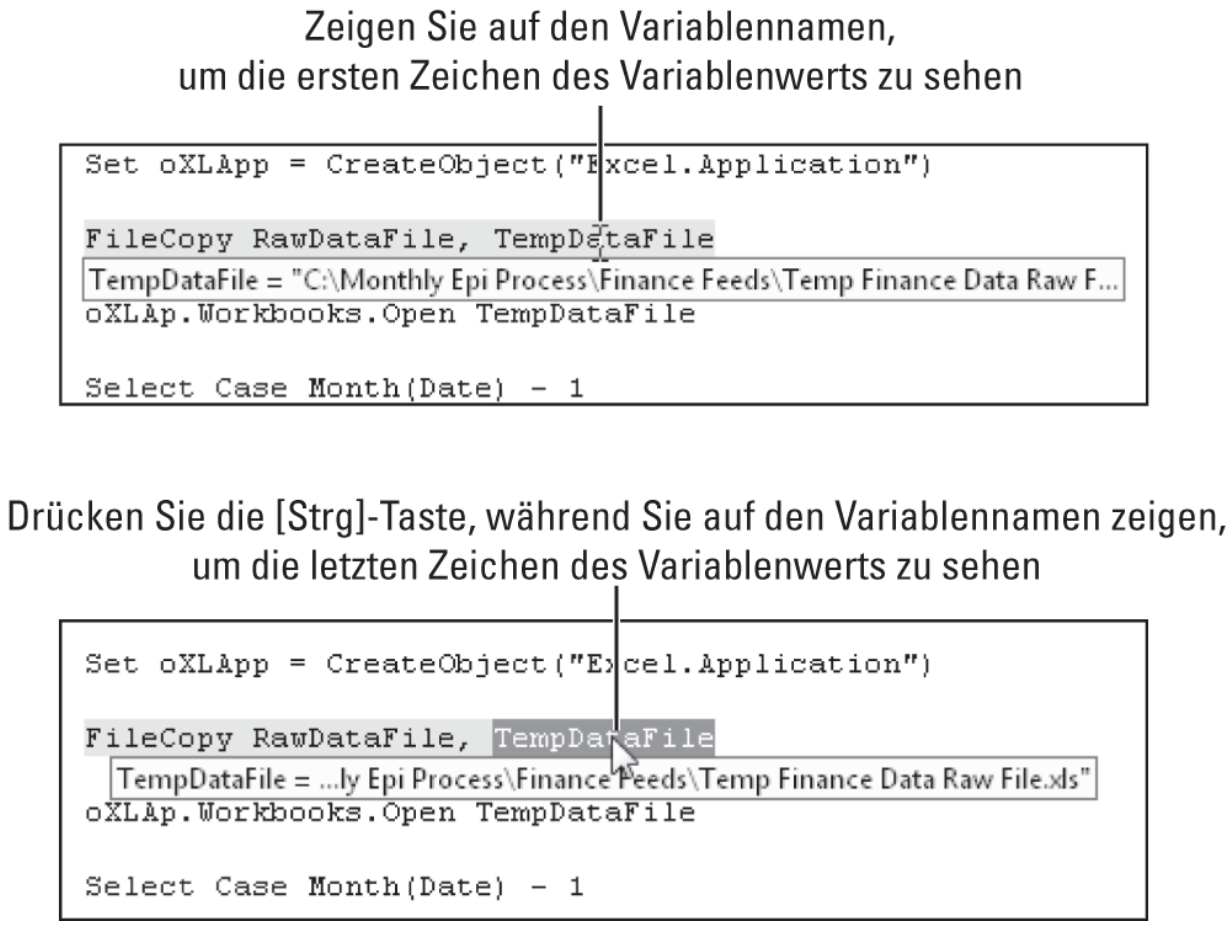


Abbildung 11.9: Anzeige der ersten und der letzten Zeichen einer String-Variablen: oben abgeschnitten, unten dank der Strg -Taste komplett

Die automatische Syntaxüberprüfung ausschalten

Während Sie an Ihrem Code arbeiten, kommt es immer wieder vor, dass Sie zu einer anderen Zeile gehen müssen, um dort irgendetwas zu kopieren. Die aktuelle Codezeile ist noch nicht fertig; Sie müssen sie für einen Moment einfach so lassen, wie sie gerade ist. Der Visual-Basic-Editor hält Sie jedoch von Ihrem Vorgehen ab und zeigt eine Warnung an ähnlich wie die in

Abbildung 11.10; dabei wissen Sie doch, dass die Zeile noch nicht komplett ist.

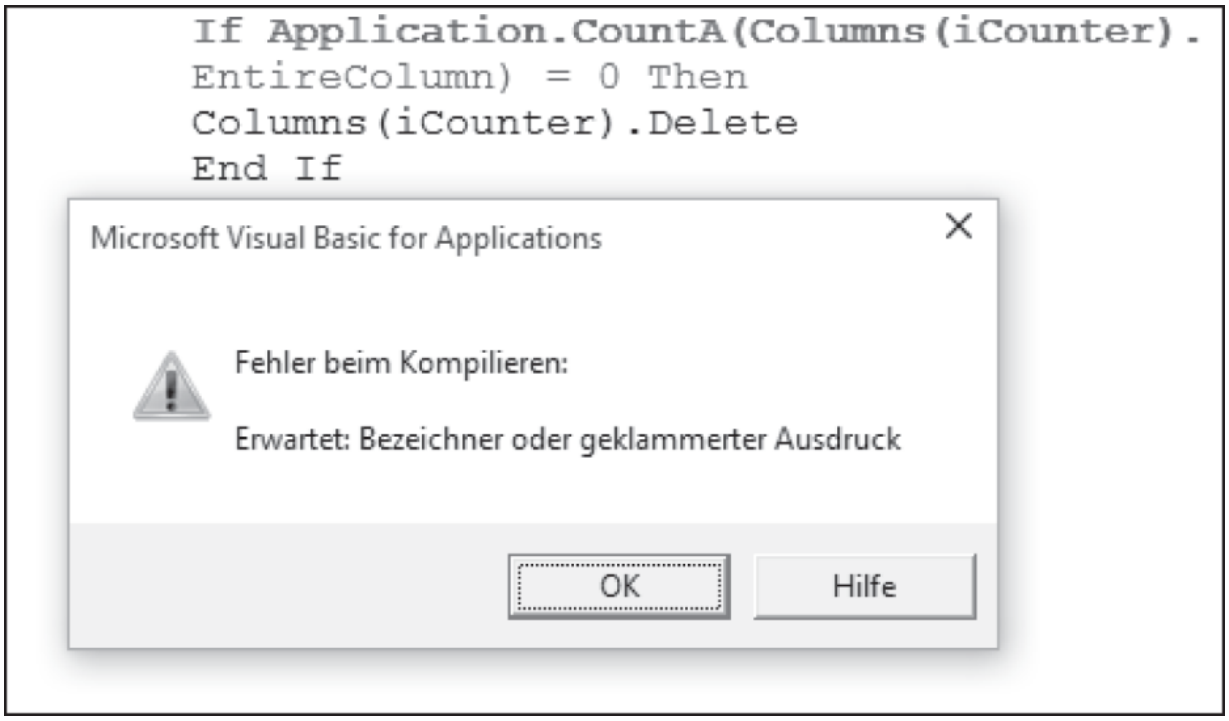


Abbildung 11.10: Eine unvollendete Codezeile führt zu diesem irritierenden Warnhinweis.

Dieser Warnhinweis führt zu einer Zwangspause, da Sie erst dann weitermachen können, wenn Sie das Dialogfeld mit OK bestätigt haben. Wenn man diese Warnhinweise auch nur einen halben Tag lang ertragen muss, ist man so genervt, dass man den Computer am liebsten gegen die nächste Wand werfen möchte.

Sie können Ihren Computer und auch Ihre Nerven schonen, indem Sie diesen Warnhinweis abschalten. Wählen Sie im Menü von VBE den Befehl EXTRAS | OPTIONEN.

Das Dialogfeld OPTIONEN wird angezeigt; die Registerkarte EDITOR ist bereits geöffnet (siehe Abbildung 11.11). Schalten Sie das Kontrollkästchen AUTOMATISCHE SYNTAXüBERPRüFUNG aus und die nervenden Warnhinweise verschwinden.

Sie brauchen sich keine Sorgen zu machen, dass hierdurch ein echter Fehler unerkannt bleibt. Ihr Code wird auch weiterhin rot eingefärbt, falls der Editor irgendwo ein Problem erkennt.

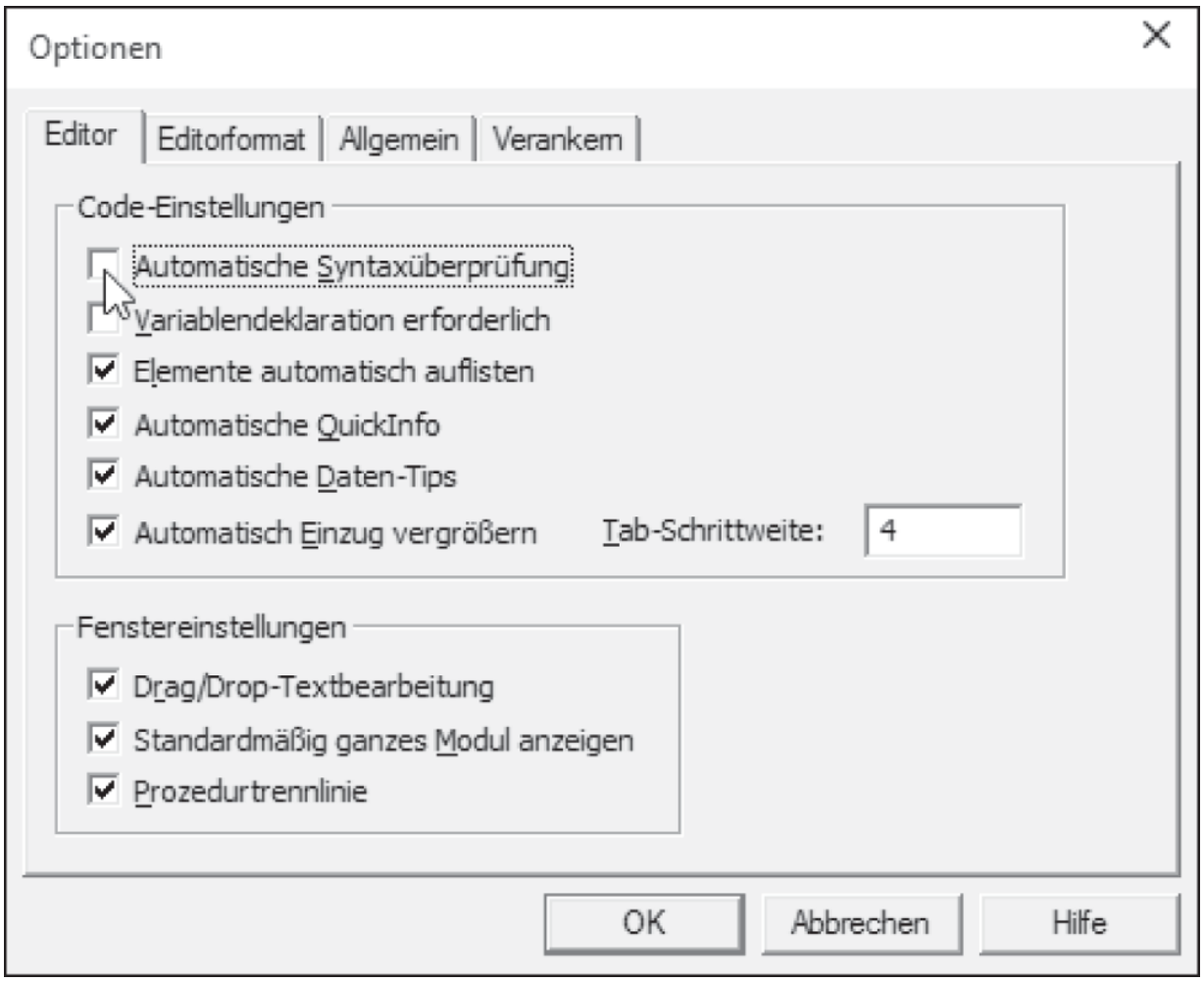


Abbildung 11.11: So unterdrücken Sie während der Programmierung die Syntax-Warnhinweise.

Kapitel 12

Zehn Orte, wo Sie Hilfe bei der Makro-Programmierung erhalten

IN DIESEM KAPITEL

- Wo Sie selbst auf die schnelle Hilfe finden
- Wo und wie Ihnen andere Leute helfen können
- Wo sich unerwartete Hilfe verstecken könnte

Niemand wird über Nacht zum Makro-Experten. VBA zu erlernen, braucht Zeit und Übung. Die gute Nachricht ist, dass es zahlreiche Ressourcen gibt, die Sie auf diesem Wege unterstützen. In diesem Kapitel stelle ich zehn nützliche Orte vor, an die Sie sich wenden können, wenn Sie einen kleinen Schubs in die richtige Richtung benötigen.

Lassen Sie Excel die Makros für Sie schreiben

Eine der besten Stellen, um Hilfe zu Makros zu erhalten, ist der Makrorekorder von Excel. Wenn Sie in Excel ein Makro aufzeichnen, übersetzt der Makrorekorder die einzelnen Schritte in den zugrundeliegenden VBA-Code. Nach der Aufzeichnung können Sie sich den erzeugten Code ansehen und ihn so bearbeiten, dass er genau zu Ihren Anforderungen passt.

Angenommen, Sie benötigen ein Makro, das alle Pivot-Tabellen Ihrer Arbeitsmappe aktualisiert und in jeder Pivot-Tabelle alle Filter zurücksetzt. Dieses Makro komplett neu zu schreiben, ist vielleicht eine Aufgabe, vor der Sie zurückschrecken. Stattdessen können Sie jedoch den Makrorekorder starten und die Aktualisierung aller PivotTable-Berichte und das Zurücksetzen aller Filter aufzeichnen. Danach beenden Sie die Makroaufzeichnung, schauen sich das so entstandene Makro an und nehmen dann die Änderungen vor, die Sie für notwendig erachten.

Verwenden Sie die VBA-Hilfedateien

Das Hilfesystem von VBA wirkt auf Excel-Anfänger wie ein klobiges Add-on, das eine verwirrende Liste mit Themen liefert, die nichts mit dem Problem zu tun hat, nach dem gesucht wurde. Nachdem Sie jedoch gelernt haben, das Excel-Hilfesystem optimal zu nutzen, ist es oft der einfachste und schnellste Weg, um zusätzliche Hilfe zu einem bestimmten Thema zu erhalten.

Wichtig ist, sich die zwei Eckpfeiler des Excel-Hilfesystems vor Augen zu halten:

- **Es ist wichtig, wo Sie sich befinden, wenn Sie Hilfe anfordern.** Excel stellt genau genommen zwei Hilfesysteme zur Verfügung: Das eine Hilfesystem liefert Hilfe zu den Excel-Features und das andere enthält Hilfe zu VBA-Programmierthemen. Nachdem Sie Ihre Suchbegriffe eingegeben haben, schickt Excel eine Suchanfrage an das Hilfesystem des Programms, in dem Sie sich gerade befinden. Falls Sie sich also in Excel selbst befinden, erhalten Sie Hilfe zu den Excel-Features. Um Hilfe zu Makros und zur VBA-Programmierung zu erhalten, verwenden Sie das Hilfe-Menü (symbolisiert durch das Fragezeichen) im Visual-Basic-Editor. Oder Sie drücken im Code-Editor des Visual-Basic-Editors auf

F1, um Hilfe zu dem VBA-Schlüsselwort zu erhalten, in dem sich gerade die Einfügemarke befindet.

- **Online-Hilfe ist besser als Offline-Hilfe.** Wenn Sie im Hilfesystem nach einem Thema suchen, prüft Excel, ob Sie mit dem Internet verbunden sind. Kann Excel eine Internetverbindung erkennen, erhalten Sie eine Liste mit Hilfethemen, die auf den Online-Inhalten der Microsoft-Website basiert. Wenn keine Internetverbindung besteht, verwendet Excel die Hilfedateien, die auf Ihrem Computer gespeichert sind und die mit Microsoft Office installiert wurden. Die Online-Hilfe ist in den meisten Fällen besser als die Offline-Hilfe, da die Inhalte der Online-Hilfe meistens mehr Detailinformationen liefern, weil die Informationen aktueller sind und weil Sie dort Links zu anderen, weiterführenden Ressourcen erhalten, die offline nicht zur Verfügung stehen.

Stibitzen Sie Code im Internet

Das schmutzige Geheimnis bei der Programmierung im Internetzeitalter ist, dass es kaum noch originalen Code gibt. Jegliche Makrosyntax, die irgendwann einmal von irgendjemandem benötigt wurde, ist irgendwo im Internet zu finden. Dies führt dazu, dass es beim Programmieren weniger um den Code geht, den jemand komplett neu erstellt, sondern vielmehr darum, bereits vorhandenen Code zu nehmen und diesen kreativ an die eigenen, spezifischen Anforderungen anzupassen.

Wenn Sie beim Erstellen eines Makros für eine bestimmte Aufgabe nicht mehr weiterkommen, starten Sie Ihre bevorzugte Suchmaschine und beschreiben Sie die Aufgabe, die Sie lösen wollen. Um gute Suchergebnisse zu erhalten, geben Sie vor der Aufgabenbeschreibung zusätzlich die Suchbegriffe *Excel VBA* ein.

Angenommen, Sie wollen ein Makro erstellen, das die leeren Zeilen eines Tabellenblatts löscht. Suchen Sie dann nach *Excel*

VBA leere Zeilen löschen. Sie können zwei Monatsgehälter darauf verwetten, dass sich irgendjemand im Netz dieser Aufgabe bereits gestellt hat. In neun von zehn Fällen werden Sie Beispielcode finden, der genau die Informationsbrocken enthält, mit denen Sie Ihr eigenes Makro erstellen können.

Userforen optimal nutzen

Sollten Sie mal in der Klemme stecken und nicht mehr weiterwissen, können Sie Ihre Fragen in einem Forum posten und so maßgeschneiderte Beratung erhalten.

Userforen sind Online-Communities, die sich einem bestimmten Thema widmen. In diesen Foren können Sie Ihre Fragen loswerden und dazu Tipps von Experten erhalten. Die Leute, die Ihre Fragen beantworten, machen dies meist ehrenamtlich und verbringen einen Teil ihrer Zeit damit, andere Forumsuser zu unterstützen und Probleme aus dem echten (Programmierer-)Leben zu lösen.

Es gibt zahlreiche Foren, die sich mit allen Fragen rund um Excel befassen. Geben Sie in Ihre bevorzugte Suchmaschine *Excel Forum* ein, um Foren zu finden.

Hier ein paar Tipps, wie Sie das meiste aus den Excel-Foren herausholen:

- **Lesen und befolgen Sie immer die Forumsregeln, bevor Sie loslegen.** Die Forumsregeln enthalten häufig Ratschläge zum Posten von Fragen und Richtlinien zu den Umgangsformen im Forum.
- **Verwenden Sie für Ihre Fragen einen präzisen und akkuraten Betreff.** Erstellen Sie keine Forumsfragen mit vagen Titeln wie *Brauche Tipp* oder *Hilfe!!!*
- **Grenzen Sie Ihre Frage so eng wie möglich ein.** Stellen Sie keine Fragen wie »Wie erstelle ich mit Excel ein Rechnungs-Makro?«.

- **Haben Sie Geduld.** Denken Sie daran, dass die Leute, die Ihre Fragen beantworten, dies ehrenamtlich tun und in der Regel einem normalen Job nachgehen. Geben Sie der Community mit der Beantwortung Ihrer Frage ein wenig Zeit.
- **Schauen Sie oft nach.** Nachdem Sie Ihre Frage gestellt haben, kann es sein, dass Sie Nachfragen erhalten und gebeten werden, weitere Details zu liefern oder das Einsatzszenario genauer zu beschreiben. Tun Sie sich und anderen den Gefallen und schauen Sie regelmäßig nach, ob es Antworten zu Ihrer Frage gibt oder ob es Folgefragen gibt.
- **Bedanken Sie sich bei dem Experten, der Ihre Frage beantwortet hat.** Wenn Sie eine Antwort erhalten haben, die Ihnen weiterhilft, nehmen Sie sich einen Moment Zeit und posten Sie ein Dankeschön für den Experten, der Ihnen weitergeholfen hat.

Besuchen Sie Experten-Blogs

Es gibt einige Excel-Gurus und -Experten, die ihr Wissen in Blogs mit anderen teilen. Oft sind diese Blogs wahre Schatztruhen voller Tipps und Tricks, die Ihnen helfen, Ihr Können zu verbessern. Das Beste ist: Sie sind auch noch kostenlos.

Auch wenn die Blogs vielleicht nicht immer direkt zu Ihrem konkreten Problem passen, so stoßen Sie dort auf zahlreiche Artikel, mit denen Sie Ihr Wissen über Excel erweitern können. Außerdem finden Sie dort zahlreiche Tipps dazu, wie sich Excel optimal in Unternehmen einsetzen lässt.

Hier folgt eine Auswahl der besten Excel-Blogs, die Sie derzeit im Internet finden können.

- **ExcelGuru:** Ken Puls ist ein Microsoft Excel MVP (Most Valuable Professional), der sein Wissen auf seinem Blog mit anderen teilt (www.excelguru.ca/blog). Neben den Artikeln in seinem Blog finden Sie dort weitere Ressourcen für das E-

Learning, wie beispielsweise Videotrainings, mit denen Sie Ihr Excel-Wissen weiter vertiefen können.

- Chandoo.org: Purna »Chandoo« Duggirala ist ebenfalls ein Microsoft Excel MVP aus Indien, dessen Blog seit 2007 aktiv ist. Sie finden in diesem innovativen Blog (www.chandoo.org) zahlreiche Vorlagen und Artikel, die generell das Ziel haben, mit Excel großartige Dinge zu machen.
- **Contextures:** Debra Dalgleish ist ein weiterer Microsoft Excel MVP und Inhaber der beliebten Excel-Site www.contextures.com. Sie finden dort eine alphabetische Liste mit mehr als 350 Excel-Themen, bei denen bestimmt auch für Sie etwas Interessantes dabei ist.
- **DailyDose:** Dick Kusleika ist der Inhaber des am längsten bestehenden Excel-Blogs (dailydoseofexcel.com). Er ist so etwas wie der König unter den Excel-VBA-Bloggern. Auf seiner Site finden Sie seine Sammlung von Artikeln und Beispielen, die er in den zehn Jahren seit der Gründung des Blogs geschrieben hat.
- **MrExcel:** Bill Jelen ist ein legendärer Excel-Botschafter. Er ist schon lange ein Microsoft Excel MVP und stellt auf seiner Website (mrexcel.com) tausende kostenloser Videos und eine riesige Bibliothek mit weiterem Schulungsmaterial bereit.
- **ThehosBlog:** Andreas Thehos ist IT-Consultant und betreibt seinen deutschsprachigen Office-Blog seit dem Jahr 2013. Excel ist eines seiner Schwerpunktthemen; Sie finden in seinem Blog (thehosblog.com) mehr als 880 Videos, in denen neben Anwenderthemen zu Excel auch Fragen zur VBA-Programmierung beantwortet werden.

Suchen Sie auf YouTube nach Schulungsvideos

Einige von uns lernen besser, wenn sie die zur Lösung einer bestimmten Aufgabe erforderlichen Schritte sehen. Falls auch Sie

Schulungsvideos besser aufnehmen können als Online-Artikel, können Sie YouTube zurate ziehen. Sie werden erstaunt sein, wie viele qualitativ hochwertige und dazu noch kostenlose Video-Tutorials Sie dort finden können, die von tollen Leuten produziert wurden, die ihr Wissen mit Ihnen teilen.

Gehen Sie zu www.YouTube.com und geben Sie als Suchbegriff **Excel VBA** ein.

Nehmen Sie an Online-Kursen teil

Online-Kurse stellen ebenfalls eine hervorragende Möglichkeit dar, sich neues Wissen anzueignen. Im englischsprachigen Raum gibt es auch Kurse, bei denen Sie direkten Kontakt mit einem Trainer haben und bei denen Sie im Kontakt und Austausch mit anderen »Schülern« neue Ideen und Tipps erhalten können.

Hier sind ein paar Links zu englisch- und deutschsprachigen Websites, auf denen Sie Online-Kurse zu Excel und zu VBA finden:

- **Excel Hero Academy:** http://academy.excelhero.com/excel-hero-academy-tuition
- Chandoo.org: http://chandoo.org/wp/vba-classes
- **ExcelJet:** https://exceljet.net
- **Video2Brain:** Video2Brain hat sich auf Videotrainings zu allen möglichen Themen spezialisiert. Besuchen Sie die Website unter www.video2brain.com und suchen Sie dort nach »Excel VBA«, um die Trainings zu finden, die sich mit der VBA-Programmierung in Excel beschäftigen.
- **Excel-Training:** Sie finden auf dieser Website (www.excel-training.de) Lektionen zu Anwenderthemen für Excel-Benutzer sowie auch einen VBA-Kurs.

Vom Office Dev Center lernen

Das *Microsoft Office Dev(elopment) Center* hat sich zum Ziel gesetzt, neue Entwickler beim Einstieg in die Programmierung der Office-Produkte zu unterstützen. Sie finden den Excel-Bereich dieser Website unter der folgenden Adresse: https://msdn.microsoft.com/de-de/library/office/fp179694.aspx.

Auch wenn die Navigation in der Website ein wenig umständlich ist, lohnt sich ein Besuch. Sie finden dort viel kostenloses Material, einschließlich Beispielcode, Werkzeugen und Schritt-für-Schritt-Anleitungen.

Analysieren Sie andere Excel-Dateien in Ihrem Unternehmen

Auch die Excel-Dateien in Ihrem eigenen Unternehmen beziehungsweise in Ihrer Organisation können Schätze enthalten, die sich zu bergen lohnen. Öffnen Sie die Excel-Dateien, die Makros enthalten, und analysieren Sie, welche Aufgaben von Ihren Kollegen mit Makros auf welche Art und Weise erledigt wurden. Schauen Sie sich die Makros genau an und prüfen Sie, ob Sie dort neue Tipps und Anregungen erhalten, die auch für Sie relevant sind.

Sie können so auf neue Techniken stoßen, an die Sie bisher möglicherweise noch nicht gedacht haben. Falls Ihre Suche optimal verläuft, finden Sie sogar komplette Codefragmente, die Sie kopieren und dann in Ihren eigenen Arbeitsmappen implementieren können.

Fragen Sie Ihre Excel-Experten vor Ort

Gibt es in Ihrer Firma, Ihrer Abteilung oder Ihrer Organisation ein Excel-Genie? Freunden Sie sich noch heute mit dieser Person an! Ihnen steht dann Ihr ganz persönliches Excel-Forum zur Verfügung.

Viele Excel-Experten lieben es, ihr Wissen mit anderen zu teilen. Haben Sie daher keine Scheu, Ihren Excel-Experten vor Ort zu fragen oder ihn um Tipps zu bitten, wie Sie ihre superinteressanten Makro-Probleme lösen können.

Kapitel 13

Zehn Methoden zur Beschleunigung von Makros

IN DIESEM KAPITEL

- Excel-Features deaktivieren, die sich nachteilig auf die Performance auswirken
- Unnötige Makroaktionen vermeiden
- Ihren Makro-Code für bessere Performance optimieren

Sobald Ihre Makros robuster und komplexer werden, kann es passieren, dass sie an Leistung verlieren. Bei Makros bezeichnet das Wort Leistung normalerweise die Ausführungsgeschwindigkeit der Makros. Hiermit ist gemeint, wie lange Ihre VBA-Prozeduren für die Bearbeitung einer bestimmten Aufgabe benötigen.

Sie können verschiedene Schritte durchführen, um die Leistung Ihrer Makros zu verbessern. In diesem Kapitel lernen Sie zehn Methoden kennen, mit denen Ihre Excel-Makros so schnell wie möglich laufen werdaen.

Automatische Neuberechnung abschalten

Wussten Sie, dass Excel immer, wenn Sie eine Zelle ändern, die Auswirkungen auf eine Formel in Ihrem Tabellenblatt hat, das gesamte Tabellenblatt neu berechnet?

Wenn sich in Ihren Tabellenblättern viele Formeln befinden, kann dieses Verhalten Ihre Makros drastisch langsamer machen.

In Tabellen mit vielen Formeln wollen Sie nicht, dass Excel immer dann eine Neuberechnung durchführt, wenn Ihr Makro nur eine Zelle ändert. Sie können die Eigenschaft `Application.Calculation` verwenden, um die manuelle Neuberechnung zu aktivieren. Befindet sich eine Arbeitsmappe im Modus manuelle Neuberechnung, wird die Arbeitsmappe erst dann neu berechnet, wenn Sie die Taste F9 drücken.

Das Abschalten der automatischen Neuberechnung kann die Geschwindigkeit Ihrer Makros erheblich verbessern. Schalten Sie die automatische Neuberechnung ab, führen Sie Ihren Code aus und schalten Sie die automatische Neuberechnung danach wieder ein:

```
Sub Makro1()
 Application.Calculation = xlCalculationManual


' Fügen Sie hier Ihren Makro-Code ein.

 Application.Calculation = xlCalculationAutomatic

End Sub
```

Wenn Sie den Berechnungsmodus auf `xlCalculationAutomatic` zurücksetzen, führt Excel automatisch eine Neuberechnung durch; es ist daher nicht erforderlich, nach der Ausführung des Makros F9 zu drücken.

Je nach Aufgabenstellung kann es sein, dass Sie während der Ausführung Ihres Makros tatsächlich Berechnungen durchführen müssen. In diesem Fall möchten Sie nicht den manuellen Berechnungsmodus aufrufen. Denken Sie unbedingt über Ihr spezielles Szenario nach und stellen Sie fest, was passiert, wenn die Berechnungen während der Ausführung Ihres Makros ausgeschaltet werden.

Die Bildschirmaktualisierung deaktivieren

Sie werden feststellen, dass Ihr Bildschirm ziemlich flackert, während ein Makro ausgeführt wird. Das Flackern resultiert daher, dass Excel versucht, den Bildschirm neu zu zeichnen, um den jeweils aktuellen Zustand des Tabellenblatts wiederzugeben. Das Neuzeichnen des Bildschirms erfordert jedoch Rechenzeit und verbraucht Speicherressourcen. In den meisten Fällen ist es nicht erforderlich, dass Excel während der Laufzeit Ihres Makros permanent den Bildschirminhalt aktualisiert.

Sie können nicht nur den Berechnungsmodus auf manuell setzen, sondern auch mit der Eigenschaft `Application.ScreenUpdating` die Bildschirmaktualisierung abschalten, während Ihr Makro läuft. Dies spart Rechenzeit und Ressourcen und Ihr Makro läuft ein wenig schneller. Nachdem Ihr Makro-Code seine Aufgaben erledig hat, können Sie die Bildschirmaktualisierung wieder einschalten.

```
Sub Makro1()
 Application.Calculation = xlCalculationManual
 Application.ScreenUpdating = False

' Fügen Sie hier Ihren Makro-Code ein.

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

Nachdem Sie die Eigenschaft `ScreenUpdating` wieder auf `True` gesetzt haben, nimmt Excel automatisch eine Aktualisierung des Bildschirminhalts vor.

Aktualisierung der Statusleiste abschalten

Am unteren Rand des Excel-Fensters befindet sich die Excel-Statusleiste. In der Statusleiste wird normalerweise der Fortschritt bestimmter Aktionen in Excel angezeigt. Wenn Sie beispielsweise einen Bereich kopieren und einfügen, zeigt Excel den Fortschritt dieser Aktion in der Statusleiste an. Oft wird die Aktion jedoch so schnell ausgeführt, dass Sie die Meldung in der Statusleiste überhaupt nicht sehen. Falls Ihr Makro jedoch viele Datenoperationen durchführt, wird durch die nicht einmal sichtbare Aktualisierung der Statusleiste Rechenzeit vergeudet.

Bitte beachten Sie, dass sich das Abschalten der Bildschirmaktualisierung nicht auf die Aktualisierung der Statusleiste auswirkt. Anders ausgedrückt: Auch wenn Sie die Bildschirmaktualisierung abgeschaltet haben, wird die Statusleiste weiterhin aktualisiert. Sie können jedoch die Eigenschaft `Application.DisplayStatusBar` verwenden, um die Aktualisierung der Statusleiste temporär abzuschalten, um so die Ausführungsgeschwindigkeit Ihrer Makros zu verbessern:

```
Sub Makro1()

 Application.Calculation = xlCalculationManual
 Application.ScreenUpdating = False
 Application.DisplayStatusBar = False

' Fügen Sie hier Ihren Makro-Code ein.

 Application.Calculation = xlCalculationAutomatic
 Application.ScreenUpdating = True
 Application.DisplayStatusBar = True
```

```
End Sub
```

Excel anweisen, Ereignisse zu ignorieren

Wie in Kapitel 3 beschrieben, können Sie Makros als Ereignisprozeduren implementieren. Hierdurch wird bestimmter Code ausgeführt, wenn sich eine Arbeitsmappe beziehungsweise ein Tabellenblatt ändert.

Manchmal nehmen Standardmakros Änderungen vor, die eine Ereignisprozedur auslösen. Angenommen, Sie haben für *Tabelle1* Ihrer Arbeitsmappe das Ereignis `Worksheet_Change` implementiert. Immer wenn eine Zelle oder ein Bereich geändert wird, wird die Ereignisprozedur `Worksheet_Change` ausgeführt.

Falls Ihr Standardmakro mehrere Zellen auf *Tabelle1* ändert, wird das Makro nach jeder Änderung einer Zelle auf diesem Tabellenblatt angehalten, damit die Ereignisprozedur `Worksheet_Change` ausgeführt werden kann. Sie können sich leicht ausmalen, wie dieses Verhalten Ihre Makros verlangsamt.

Sie können Ihre Makros weiter beschleunigen, indem Sie Excel anweisen, während der Ausführung Ihrer Makros Ereignisse zu ignorieren.

Setzen Sie dazu die Eigenschaft `EnableEvents` auf `False`, bevor Sie das Makro ausführen. Wenn die Ausführung Ihres Codes beendet ist, setzen Sie die Eigenschaft `EnableEvents` zurück auf `True`.

```
Sub Makro1()

 Application.Calculation = xlCalculationManual
 Application.ScreenUpdating = False
 Application.DisplayStatusBar = False
 Application.EnableEvents = False

' Fügen Sie hier Ihren Makro-Code ein.
```

```
    Application.Calculation = xlCalculationAutomatic
    Application.ScreenUpdating = True
    Application.DisplayStatusBar = True
    Application.EnableEvents =True

End Sub
```

Obwohl das Abschalten der Ereignisse die Ausführungsgeschwindigkeit Ihrer Makros verbessert, kann es sein, dass während der Ausführung Ihrer Makros bestimmte Ereignisse ausgelöst werden *müssen*. Analysieren Sie daher Ihr spezifisches Szenario und überlegen Sie, wie sich das Ignorieren von Arbeitsmappen- und Tabellenblattereignissen auf Ihren Makro-Code auswirkt.

Seitenwechsel ausblenden

Eine weitere Möglichkeit für die Optimierung der Ausführungsgeschwindigkeit bieten Seitenwechsel. Immer dann, wenn ein Makro die Anzahl der Zeilen beziehungsweise die Anzahl der Spalten ändert oder Änderungen an der Seiteneinrichtung eines Tabellenblatts vornimmt, ist Excel gezwungen, die Seitenwechsel des Tabellenblatts neu zu berechnen, was Zeit benötigt.

Sie können dies vermeiden, indem Sie vor dem Start Ihres Makros alle Seitenwechsel ausblenden.

Um die Seitenwechsel auszublenden, setzen Sie die Sheet-Eigenschaft `DisplayPageBreaks` auf `False`. Wenn Sie die Seitenwechsel nach der Ausführung Ihres Makros wieder anzeigen wollen, setzen Sie die Eigenschaft `DisplayPageBreaks` auf `True` zurück.

```
Sub Makro1()

    Application.Calculation = xlCalculationManual
    Application.ScreenUpdating = False
    Application.DisplayStatusBar = False
```

```
 Application.EnableEvents = False
 ActiveSheet.DisplayPageBreaks = False

' Fügen Sie hier Ihren Makro-Code ein.

 Application.Calculation = xlCalculationAutomatic
 Application.ScreenUpdating = True
 Application.DisplayStatusBar = True
 Application.EnableEvents =True
 ActiveSheet.DisplayPageBreaks = True

End Sub
```

Die Aktualisierung von Pivot-Tabellen unterdrücken

Wenn Ihr Makro Pivot-Tabellen ändert, die auf umfangreichen Datenquellen basieren, kann es sein, dass beim Einfügen oder Verschieben von Pivot-Feldern die Ausführungsgeschwindigkeit nachlässt. Bei jeder Änderung, die Sie an der Struktur der Pivot-Tabelle vornehmen, muss Excel die Werte der Pivot-Tabelle für jedes Feld neu berechnen, das von Ihrem Makro angefasst wird.

Sie können die Ausführungsgeschwindigkeit Ihres Makros verbessern, indem Sie die Neuberechnung der Pivot-Tabelle so lange unterdrücken, bis alle Änderungen an den Pivot-Feldern vorgenommen wurden. Hierzu setzen Sie die Eigenschaft `PivotTable.ManualUpdate` auf `False`, führen Ihren Makro-Code aus und setzen dann `PivotTable.ManualUpdate` wieder zurück auf `True`, damit die Neuberechnung vorgenommen wird.

```
Sub Makro1()

 ActiveSheet.PivotTables("PivotTable1").ManualUpdate=True

 ' Fügen Sie hier Ihren Makro-Code ein.

 ActiveSheet.PivotTables("PivotTable1").ManualUpdate=False

End Sub
```

Kopieren und Einfügen vermeiden

Es ist wichtig, daran zu denken, dass der Makrorekorder zwar Zeit beim Erstellen von VBA-Code einsparen kann, er aber nicht immer sehr effizienten Code erstellt. Ein typisches Beispiel hierfür ist der Code, den der Makrorekorder für das Kopieren und Einfügen erzeugt.

Angenommen, Sie wollen den Inhalt von Zelle A1 in Zelle B1 kopieren und zeichnen diese Schritte auf, so erzeugt der Makrorekorder den folgenden Code:

```
Range("A1").Select
Selection.Copy
Range("B1").Select
ActiveSheet.Paste
```

Der Code macht zwar, was er soll, allerdings wird Excel hiermit angewiesen, die Zwischenablage als Mittelsmann zu verwenden, was aber in diesem Fall gar nicht erforderlich ist.

Sie können die Geschwindigkeit Ihrer Makros leicht verbessern, indem Sie den Mittelsmann ausschalten und direkt von einer Zelle in eine Zielzelle kopieren. Der alternative Code verwendet das Argument `Destination`, um die Zwischenablage zu vermeiden und um die Inhalte der Zelle A1 direkt in Zelle B1 zu kopieren.

```
Range("A1").Copy Destination:=Range("B1")
```

Falls Sie lediglich Werte und keine Formeln oder Formatierung kopieren wollen, können Sie die Leistung weiter verbessern, indem Sie komplett auf die Methode `Copy` verzichten. Setzen Sie den Wert der Zielzelle einfach auf den Wert, der sich in der Quellzelle befindet. Dieses Verfahren ist ungefähr 25-mal schneller als die Verwendung der Methode `Copy`:

```
Range("B1").Value = Range("A1").Value
```

Falls Sie nur Formeln aus einer Zelle in eine andere kopieren wollen, aber keine Werte und keine Formatierung, können Sie die

Formel der Zielzelle auf die gleiche Formel setzen, die sich in der Ausgangszelle befindet:

```
Range("B1").Formula = Range("A1").Formula
```

Die With-Anweisung verwenden

Beim Aufzeichnen von Makros nehmen Sie oft mehrere Änderungen am gleichen Objekt vor. Angenommen, Sie wollen die Formatierung in Zelle A1 auf unterstrichen, kursiv und fett festlegen. Wenn Sie ein Makro aufzeichnen, das diese Formatierungen an Zelle A1 vornimmt, erhalten Sie folgendes Makro:

```
Range("A1").Select
Selection.Font.Bold = True
Selection.Font.Italic = True
Selection.Font.Underline = xlUnderlineStyleSingle
```

Unglücklicherweise ist dieser Code nicht so effizient, wie er sein könnte, da Excel gezwungen wird, jede Eigenschaft getrennt auszuwählen und zu ändern.

Sie können Rechenzeit sparen und die Leistung verbessern, indem Sie die `With`-Anweisung verwenden, um mehrere Aktionen an einem Objekt in einem Rutsch auszuführen.

Die `With`-Anweisung im folgenden Codefragment weist Excel an, alle Formatierungen aus obigem Code auf einmal anzuwenden:

```
With Range("A1").Font
 .Bold = True
 .Italic = True
 .Underline = xlUnderlineStyleSingle
End With
```

Wenn Sie sich angewöhnen, für solche Aktionen die `With`-Anweisung zu verwenden, werden Ihre Makros nicht nur schneller ablaufen, sondern Ihr Makro-Code wird so auch besser lesbar.

Die Methode Select vermeiden

Wenn Sie ein Makro aufzeichnen, das auf mehreren Tabellenblättern jeweils in Zelle A1 den Wert 1000 eingibt, erhalten Sie einen Code, der etwa so aussieht wie folgendes Fragment:

```
Sheets("Tabelle1").Select
Range("A1").Select
ActiveCell.FormulaR1C1 = "1000"

Sheets("Tabelle2").Select
Range("A1").Select
ActiveCell.FormulaR1C1 = "1000"

Sheets("Tabelle3").Select
Range("A1").Select
ActiveCell.FormulaR1C1 = "1000"
```

Wie Sie sehen, mag der Makrorekorder die `Select`-Anweisung, um die Objekte explizit auszuwählen, an denen anschließend Änderungen vorgenommen werden. Auch wenn dieser Code gut funktioniert, so ist er nicht besonders effizient, da er Excel dazu zwingt, mit der Auswahl des Objekts, das anschließend bearbeitet wird, Zeit zu verbringen.

Es ist im Allgemeinen nicht erforderlich, ein Objekt auszuwählen, bevor Sie es bearbeiten. Sie können sogar die Ausführungsgeschwindigkeit Ihrer Makros enorm verbessern, indem Sie ganz auf den Einsatz der Methode `Select` verzichten.

Nachdem Sie ein Makro aufgezeichnet haben, sollten Sie sich den generierten Code anschauen und die Verwendung von `Select` entfernen. Der optimierte Code für unser Beispiel sieht so aus:

```
Sheets("Tabelle1").Range("A1").FormulaR1C1 = "1000"
Sheets("Tabelle2").Range("A1").FormulaR1C1 = "1000"
Sheets("Tabelle3").Range("A1").FormulaR1C1 = "1000"
```

Beachten Sie, dass hier nichts ausgewählt wird. Der Code verwendet einfach die Objekthierarchie, um die erforderlichen Aktionen durchzuführen.

Zugriffe auf das Tabellenblatt optimieren

Ein weiterer Weg zur Optimierung der Ausführungsgeschwindigkeit Ihrer Makros besteht darin, die Anzahl der Zugriffe auf Daten des Tabellenblatts zu reduzieren. Es ist immer weniger effizient, die Daten aus dem Tabellenblatt zu holen, statt sie direkt aus dem Speicher zu lesen. Ihre Makros werden schneller ablaufen, wenn sie nicht wiederholt mit dem Tabellenblatt interagieren müssen.

In dem folgenden Codefragment muss Excel in der `For`-Schleife wiederholt den Wert der Zelle A1 ermitteln, da er für den Vergleich in der `If`-Anweisung benötigt wird.

```
For ReportMonth = 1 To 12
 If Range("A1").Value = ReportMonth Then
 MsgBox 1000000 / ReportMonth
 End If
Next ReportMonth
```

Es ist viel effizienter, den Wert in Zelle A1 einmal aus dem Tabellenblatt zu lesen und ihn in einer Variablen abzuspeichern. Der Code in der `If`-Anweisung greift nun auf die Variable `MyMonth` zu und nicht mehr wiederholt auf das Tabellenblatt:

```
Dim MyMonth as Integer
MyMonth = Range("A1").Value

For ReportMonth = 1 To 12
 If MyMonth = ReportMonth Then
 MsgBox 1000000 / ReportMonth
 End If
Next ReportMonth
```

Stichwortverzeichnis

Symbole

Strg+Bild ab 275
Strg+Tab 275
| (Pipe-Symbol) 117
& (Kaufmanns-Und) 176
* (Sternchen) 117
\" (Anführungszeichen) 117
> (Größerzeichen) 117
< (Kleinerzeichen) 117
/ (Schrägstrich) 117
_ (Unterstrich) 51

A

Absolute Verweise, Makroaufzeichnung 30–35
Access
    Daten in Excel importieren 252
ActiveCell.PivotTable.Name, Eigenschaft 200
ActiveChart.PrintOut, Methode 231
ActiveSheet.PrintOut, Methode 220
ActiveSheet.UsedRange 178
ActiveWorkbook, Objekt 194
Add, Methode76, 102
adLockOptimistic, ADO-Locktype 264

adLockReadOnly, ADO-Locktype 264
adOpenForwardOnly, ADO-Cursortyp 264
ADO-Syntax 262
    Verbindungszeichenfolge 262
Adressen, an Kontaktliste mailen 242–244
Aktivieren
    Visual-Basic-Editor 19, 28, 45
Aktualisieren
    externe Daten 254
Aktualisierung
    der Statusleiste abschalten 291
    des Bildschirms deaktivieren 290–291
    von Pivot-Tabellen in Arbeitsmappen 194–195
    von Pivot-Tabellen unterdrücken 293
Allgemein, Registerkarte (VBA) 57
Alphabetisch sortieren, Datenfelder 207–209
Anhänge
    alle in Ordner speichern 245
    Arbeitsmappe versenden 233
    bestimmte in Ordner speichern 247
    bestimmtes Tabellenblatt senden 234
    Zellbereich mailen 236
Anpassen der Titel von Datenfeldern einer Pivot-Tabelle 199

Anweisungen
Dim 63
On Error GoTo 0 71
On Error GoTo IrgendeineSprungmarke 69
On Error Resume Next 70
Open, Syntax 267
Select Case 80
With 294–295
With-End-With 150
ANZAHL2, Funktion 33
Anzeigen
Spalten, nach denen gefiltert wird 186
Application.Calculation, Eigenschaft 289
Application.DisplayStatusBar, Eigenschaft 291
Application-Objekt 60
Application.ScreenUpdating, Eigenschaft 290

Arbeitsmappe
aktive als E-Mail-Anhang versenden 233–234
alle geöffneten schließen 93–94
alle Pivot-Tabellen aktualisieren 194–195
beim Öffnen bestimmtes Tabellenblatt anzeigen 85–86
erst schließen, wenn bestimmte Zelle Inhalte enthält 96–98
Formeln auswählen 149
gefilterte Zeilen in neue kopieren 103–105
Inhaltsverzeichnis erstellen 123
Mail mit Link zu Arbeitsmappe senden 240–242
neue erstellen 75–77
neue für jedes Element des Berichtsfilters 222–224
prüfen, ob geöffnet 88–91
prüfen, ob vorhanden 91–93
Sicherung erstellen 98–99
von Benutzer ausgewählte öffnen 90
vor Schließen speichern 80–82
Zusammenfassung aller Pivot-Tabellen erstellen 195
Arbeitsmappenereignisse 67–68
Arbeitsmappenverbindungen
bearbeiten 256
ASCII-Code 201
Atmt, Variable 246, 249
Aufzeichnung
Makros 26–28
mit relativen Verweisen 33

Ausblenden
alle Tabellenblätter bis auf das aktive 105–107
Seitenwechsel 292–293
Statusleiste 291
Ausblenden
AutoFilter-Drop-down-Listen 183–185
AutoFilter-Funktion 183
AutoFilterMode, Eigenschaft 185–186, 188
Automatisch Einzug vergrößern, Registerkarte Editor 55
Automatische Daten-Tipps, Registerkarte Editor 55
Automatische QuickInfo, Registerkarte Editor 55
Automatische Syntaxüberprüfung
ausschalten 280–281
Automatische Syntaxüberprüfung, Registerkarte Editor 54
Automatisierung
Definition 17
AutoSort, Methode 209

B

Backup von Arbeitsmappen mit aktuellem Datum 98–99
BCC, Eigenschaft 244
Bearbeiten von Makros 29
Bearbeiten, Symbolleiste 274
BeforeClose, Ereignis 65, 80, 82, 190
Benutzerdefinierte Sortierung 209–210

Bereiche
alle Zellen bearbeiten 139
als E-Mail-Anhang 236
auswählen 134–136
benannte erstellen 136
Diagramme platzieren 226–228
Duplikate in Daten hervorheben 178
einfügen 156
formatieren 135
Formeln in Werte konvertieren 157–159
kopieren 156
leere Spalten löschen 145
leere Zeilen einfügen 140
leere Zeilen löschen 143
Scrollbereich einschränken 147

Berichte
alle Diagramme eines Tabellenblatts drucken 230–231
Datenelemente in Pivot-Tabellen als Zahlen formatieren 204
Datenelemente in Pivot-Tabellen benutzerdefiniert sortieren 209
Datenfelder in Pivot-Tabellen alphabetisch sortieren 207
Drilldown-Tabellenblätter automatisch entfernen 215
Einschränkungen für Pivot-Tabellen festlegen 211
Einschränkungen für PivotTable-Felder festlegen 213
Größe aller Diagramme ändern 225
Liste mit Pivot-Tabellen erstellen 195
neue Arbeitsmappe für jedes Element des Berichtsfilters 222
Pivot-Tabellen für jedes Element des Berichtsfilters drucken 220
PivotTables-Datenfelder anpassen 199
Summe für Zusammenfassung in Pivot-Tabellen verwenden 201
Verknüpfung zwischen Diagramm und Daten lösen 229
Berichtsfilter
Pivot-Tabellen für jeden drucken 219–221
Pivot-Tabellen für jeden erstellen 222–224
Beschreibung, Dialogfeld Makro aufzeichnen 27
Bildlauf
einschränken 147–148
Bildschirmaktualisierung deaktivieren 290–291
Binärer Dateizugriff 267
Blattschutz
beim Öffnen aufheben 84

Blogs 285–286
Boolean, Datentyp 63
BreakLink, Methode 230

C

Caption, Eigenschaft 201
CDbl, Funktion 164
Cells, Sammlung 150
Chandoo.org, Blog 286–287
ChartObjects, Sammlung 226, 231
ChartObjects.ShapeRange.Group 230
Chr, Funktion 178, 201
Code
    Ausführung an bestimmter Stelle unterbrechen 278–279
    im Internet finden 284–285
    mehrere Codezeilen kopieren 275
    schrittweise ausführen 276–277
    zu einer bestimmten Codezeile springen 277
Code-Farben, Registerkarte Editorformat 56
Codefenster (VBE) 47
    anordnen 50
    maximieren 50
    schließen 51
    wiederherstellen 50
Collections 61
Columns, Sammlung 147
Contextures, Blog 286

Copy, Methode 156, 294
Count, Eigenschaft 226
COUNTA, Funktion 33
CountIf, Funktion 179
CSV-Dateien
    in Bereich importieren 269
CurrentSheetIndex, Variable 111
Cursortypen 264

D

Daily Dose of Excel, Blog 286
Dalgleish, Debra (Blogger) 286
Data Source, ADO 263
Date, Funktion 98
Dateierweiterungen, für Makros 35–41
Dateinamen, nicht erlaubte Zeichen 117

Datenbearbeitung
alle Zeilen bis auf Zeilen mit Duplikaten ausblenden 180
AutoFilter-Drop-down-Listen ausblenden 183
Duplikate hervorheben 178
Formeln in Werte konvertieren 157
führende Nullen einfügen 169
führende und nachstehende Leerzeichen entfernen 165
gefilterte Zeilen in neue Mappe kopieren 185
leere Zellen durch Wert ersetzen 172
nachstehende Minuszeichen entfernen 162
nicht druckbare Zeichen entfernen 177
Spalten anzeigen, nach denen gefiltert wird 186
Text in Spalten 159
Text kürzen 167
vor oder nach Zellinhalt Text einfügen 174
Zellbereich kopieren 156–157
Datenfelder
alphabetisch sortieren 207–209
Datum
Arbeitsmappe mit aktuellem Datum sichern 98
Deaktivieren
Aktualisierung von Pivot-Tabellen 293
automatische Neuberechnung 289–290
Bildschirmaktualisierung 290–291
Ereignisse 291–292
Statusleiste 291

Debugging
Ausführung an bestimmter Stelle unterbrechen 278
schrittweise ausführen 276
Variablenwert anzeigen lassen 279
zu einer bestimmten Codezeile springen 277–278
Debugmodus
aktivieren 277
beenden 277, 279
Diagramme
alle auf einem Tabellenblatt drucken 230–231
an Zellbereich ausrichten 226–228
Größe ändern 225–226
gruppieren 230
Verknüpfung zu Daten aufheben 228–230
Dim, Anweisung 63
Direktbereich (VBE) 47
Direktfenster (VBE) 47
Dir, Funktion 92, 95
Display, Methode 235, 237, 240, 244
DisplayAlerts, Eigenschaft 77, 103
DisplayPageBreaks, Eigenschaft 292
Doppelklick
zoomen 126–128
Double, Datentyp 63
Drag/Drop-Textbearbeitung, Registerkarte Editor 55
DragToColumn, Eigenschaft 213
DragToData, Eigenschaft 214
DragToHide, Eigenschaft 214

DragToPage, Eigenschaft 213
DragToRow, Eigenschaft 213
Drilldown-Tabellenblätter entfernen 215
Drucken
alle Arbeitsmappen eines Ordners 94
alle Diagramme eines Tabellenblatts 230–231
bestimmte Tabellenblätter 118
Duggirala, Purna »Chandoo« (Blogger) 286
Duplikate
alle Zeilen bis auf Zeilen mit Duplikaten ausblenden 180–183
in einem Datenbereich hervorheben 178–180

E

Editor, Registerkarte (VBA) 53
Editorformat, Registerkarte (VBA) 56

Eigenschaften 61
ActiveCell.PivotTable.Name 200
Application.Calculation 289
Application.DisplayStatusBar 291
Application.ScreenUpdating 290
AutoFilterMode 185–186, 188
BCC 244
Caption 201
Count 181
DisplayAlerts 77, 103
DisplayPageBreaks 292
DragToColumn 213
DragToData 214
DragToHide 214
DragToPage 213
DragToRow 213
EnableDrilldown 211, 217
EnableEvents 292
EnableFieldDialog 211
EnableFieldList 211
EnableItemSelection 214
EnableWizard 211
End 154
Function 202–203
HasFormula 158
Height 227
HTMLBody 242
Left 227

Name (ActiveSheet) 102
NumberFormat 171
Offset 34, 153–154, 197
PivotCache.EnableRefresh 212
PivotTable.ManualUpdate 293
Position 210
SourceData 204
SourceName 201
StatusBar 189
Top 227
UsedRange 143–145
Width 227
Einblenden
alle Zeilen und Spalten 142
Dialogfeld 107
Symbolleiste Bearbeiten 274
Tabellenblatt 107
Einfügen
leere Zeilen in einen Bereich 140–142
Tabellenblatt 101–103
Text vor oder nach Zellinhalt 174
VBA-Modul 48–50
Zellen mit führenden Nullen versehen 169
Einschränkungen
für Pivot-Tabellen festlegen 210–213
für PivotTable-Felder festlegen 213–215
Elemente automatisch auflisten, Registerkarte Editor 54
E-Mail-Anhang. siehe Anhänge

E-Mails
aktive Arbeitsmappe senden 233–235
alle Anhänge in einem Ordner speichern 245
an alle Adressen in Liste mit Kontakten senden 242
bestimmte Anhänge in einem Ordner speichern 247
bestimmten Zellbereich senden 236
bestimmtes Tabellenblatt senden 238
Link zu Arbeitsmappe senden 240
EnableDrilldown, Eigenschaft 211, 217
EnableEvents, Eigenschaft 292
EnableFieldDialog, Eigenschaft 211
EnableFieldList, Eigenschaft 211
EnableItemSelection, Eigenschaft 214
EnableWizard, Eigenschaft 211
End, Eigenschaft 154
Entwicklertools, Registerkarte 45
anzeigen 26
Befehl Makros 35
relative Verweise verwenden 33

Ereignisse
Arbeitsmappe 67
BeforeClose 65, 80, 82, 190
Calculate 189
Definition 64
ignorieren 291–292
Open 84
Tabellenblatt 65
Workbook_BeforeClose 68, 190, 216–217, 219
Workbook_BeforeSave 68
Workbook_Open 68, 84
Workbook_SheetChange 68
Worksheet_Activate 67, 190
WorkSheet_BeforeDoubleClick 67, 216, 218
Worksheet_BeforeRightClick 67
Worksheet_Calculate 67, 189–190
Worksheet_Change 66, 292
Worksheet_Deactivate 67, 190
Worksheet_SelectionChange 66
Erstellen
Arbeitsmappe für jedes Element des Berichtsfilters 222–224
benannte Bereiche 136
Navigationsschaltflächen 41
Zusammenfassung aller Pivot-Tabellen 195
Excel-Experten 285
ExcelGuru, Blog 286
Excel Hero Academy, Website 287
ExcelJet, Website 287

Excel-Objektmodell 60
Eigenschaften 61–62
Methoden 62
Objekte 60
Sammlungen 61
Excel-Training, Website 287
Extended Properties, ADO 263
Externe Daten
abrufen 252
Externe Datenverbindungen
bearbeiten 255
manuell erstellen 252
verwenden 251

F

Farben, Tabellenblätter gruppieren 113–115
Fehler, Makros verwenden zur Fehlervermeidung 26
Fehlerbehandlung
On Error GoTo 0 71
On Error GoTo IrgendeineSprungmarke 69
On Error Resume Next 70
FileName, Variable 246, 249
Fließkommazahlen 63
For-Each-Anweisung 139, 200, 203, 209, 215
Formatierung
alle Formeln 151
Zellbereich 135

Formeln
in Werte konvertieren 157–159
suchen 150
Function, Eigenschaft 202–203
Function-Prozedur 49
Funktionen
ANZAHL 233
CDbl 164
Chr 178, 201
COUNTA 33
CountIf 179
Dir 92, 95
InStr 248
IsEmpty 161–162, 166, 168, 171, 174–175, 181–182
IsNumeric 163–164
JETZT 189
Left 167
Len 172
MkDir 246, 248
Right 172
Trim 165
UCase 112
ZÄHLENWENN 180

G

GetOpenFilename, Methode 88
Groß- und Kleinschreibung bei Kennwörtern 83

Größe, Registerkarte Editorformat 57

H

Haltepunkte setzen 278
HasFormula, Eigenschaft 158
Height, Eigenschaft 227
Hervorheben
    Duplikate 178–180
HTMLBody, Eigenschaft 242

I

Icons 19
If-Anweisung 79
Inhaltsverzeichnis einer Arbeitsmappe erstellen 123–124
Input #, Textdatei lesen 268
Input, Textdatei lesen 268
InStr, Funktion 248
Integer, Datentyp 63
Integer-Variablen 279
Intersect, Methode 78
IsEmpty, Funktion 161–162, 166, 168, 171, 174–175, 181–182
IsNumeric, Funktion 163–164

J

Jelen, Bill (Blogger) 286
JETZT, Funktion 189

K

Kaufmanns-Und (&) 176
Kennwörter, Groß- und Kleinschreibung 84
Kennzeichenleiste, Registerkarte Editorformat 57
Kommentarblöcke 273–274
Kommentare 29
Kontakte 242
Konvertieren
    Formeln in Werte 157–158
    in Datentyp Double 164
    nachstehende Minuszeichen 162
Kopieren
    mehrere Codezeilen 275
    gefilterte Zeilen in neue Arbeitsmappe 185
    Zellbereich 156
Kusleika, Dick (Blogger) 286

L

LastColumn, Variable 154
LastRow, Variable 153
Leerzeichen, geschützt 177–178
Left, Eigenschaft 227
Left, Funktion 167
Len, Funktion 172
Line Input #, Textdatei lesen 268
Link zu Arbeitsmappe versenden 240
Long, Datentyp 63

Löschen
alle Tabellenblätter bis auf das aktive 103
Drilldown-Tabellenblätter 215
führende und nachstehende Leerzeichen entfernen 165
leere Spalten 145
leere Zeilen 143
nicht druckbare Zeichen 177
VBA-Modul 50

M

Makro aufzeichnen, Dialogfeld 94, 143, 145, 147, 151
Makro speichern in, Dialogfeld Makro aufzeichnen 27
Makroname, Dialogfeld Makro aufzeichnen 27
Makrorekorder 283

Makros
aufzeichnen 26
bearbeiten 29
Beispiele für den Einsatz von 41
beschleunigen 289
Dateierweiterungen 35
definition 17
Ereignisprozedur 64
Excel-Objektmodell 60
Fehlerbehandlung 69
mit relativen Verweisen aufzeichnen 33
in persönlicher Makroarbeitsmappe speichern 37
testen 29
Variablen 62
zuweisen an Formularsteuerelement 38
zuweisen in Symbolleiste für den Schnellzugriff 40
Makrosicherheit 36
MAPI-Namensraum 249
MAPI-Sitzung 235
Maximieren des Codefensters (VBE) 50
Menüband anpassen 26
Menüleiste (VBE) 46

Methoden 62
ActiveChart.PrintOut 231
ActiveSheet.PrintOut 220
Add 76, 102
AutoSort 209
BreakLink 230
Copy 156, 294
Display 235, 237, 240, 244
Excel-Objektmodell 59
GetOpenFilename 88
Intersect 78
Move 109, 112
Paste 76
PasteSpecial 156
PrintOut 119
RefreshAll 195
RefreshTable 195
Replace 177–178
Save 81
SaveAs 77
SaveCopyAs 99
Select vermeiden 295
SendMail 234
ShapeRange.Group 229
ShowDetail 217
SpecialCells 150
Microsoft Office Dev Center, Website 287
Microsoft Outlook Object Library 234, 236, 238, 241, 243, 245, 248

MkDir, Funktion 246, 248
Module
springen zwischen Prozeduren und Modulen 275
Move, Methode 109, 112

N

Name
Tabellenblätter nach Namen sortieren 110
Name (Workbook), Eigenschaft 98
Name, Eigenschaft 102
Namens-Manager 137
Navigation, Makros verwenden für 26
Navigationsschaltflächen 41
Neuberechnung abschalten 289
Neuer Name, Dialogfeld 137
Nicht druckbare Zeichen entfernen 177
Nicht vergessen, Icon 19
Nothing, Konstante 78
ns, Variable 246, 249
NumberFormat, Eigenschaft 171

O

Object, Datentyp 63

Objekte
ActiveWorkbook 194
PageField 220
PivotItems 210, 220, 223
ThisWorkbook 194
Offline-Hilfe 284
Offnen
beim Öffnen Blattschutz aufheben 84
bestimmtes Tabellenblatt anzeigen 85
vom Benutzer ausgewählte Arbeitsmappe 87
Offset, Eigenschaft 34, 153–154, 197
OLApp, Variable 235, 237, 239, 242, 244
OLApp.Session.Logon 235, 237, 239, 242, 244
OLMail, Variable 235, 237, 239, 242, 244
On Error GoTo 0 71
On Error GoTo IrgendeineSprungmarke 69
On Error Resume Next 200, 203, 206, 209, 221, 224, 247, 249
Online-Hilfe 284
Online-Kurse 287
Open, Anweisung 267
Open, Ereignis 84
Option Explicit 54
Ordner
alle Arbeitsmappen eines Ordners drucken 94
prüfen, ob Arbeitsmappe vorhanden ist 91
Ordner, E-Mail-Anhänge speichern 245, 247

P

PageField 225
PageField, Objekt 220
Password, ADO 263
Password, Parameter 120
Paste, Methode 76
PasteSpecial, Methode 156
Path (Workbook), Eigenschaft 98
personal.xlsb, persönliche Makroarbeitsmappe 38
Persönliche Makroarbeitsmappe 38
Pivot-Cache 204
PivotCache.EnableRefresh, Eigenschaft 212
PivotItems, Objekt 210, 220, 223
Pivot-Tabellen
    Aktualisierung unterdrücken 293
    alle aktualisieren 194
    Daten neu anordnen 42
    Datenelemente als Zahlen formatieren 204
    Datenelemente benutzerdefiniert sortieren 209
    Datenfelder alphabetisch sortieren 207
    Datenfelder anpassen 199
    Drilldown-Tabellenblätter automatisch entfernen 215
    Einschränkungen festlegen 211
    Einschränkungen für PivotTable-Felder festlegen 213
    für jedes Element des Berichtsfilters drucken 220
    Liste erstellen 195
    neue Arbeitsmappe für jedes Element des Berichtsfilters 219
    Summe für Zusammenfassung verwenden 201
PivotTable.ManualUpdate, Eigenschaft 293

Position, Eigenschaft 210
PrevSheetIndex, Variable 111
Print #, Textdatei schreiben 268
PrintOut, Methode 119
Projekt-Explorer
    VBA-Modul einfügen 48
    VBA-Modul löschen 50
Projekt-Explorer (VBE) 47
Provider, ADO 263
Prozeduren
    springen zwischen Prozeduren und Modulen 275
Prozedurtrennlinie, Registerkarte Editor 56
Puls, Ken (Blogger) 286

R

Recordset-Objekt 263
Redundante Schritte, durch Makro erledigen lassen 25
RefreshAll, Methode 195
RefreshTable, Methode 195
Registerfarbe, Tabellenblätter gruppieren 113
relative Verweise, Makroaufzeichnung 33
Replace, Methode 177–178
Replacement 178
Right, Funktion 172
Rows, Sammlung 144

S

Sammlungen 61
Save, Methode 81
SaveAs, Methode 77
SaveCopyAs, Methode 99
Schließen
    alle Arbeitsmappen auf einmal 93
    vorher automatisch speichern 80
Schriftart, Registerkarte Editorformat 56
Schulungsvideos 286
Schützen
    Tabellenblatt 120
Schützen von Tabellenblättern 84
ScrollArea, Eigenschaft 147
Seitenwechsel ausblenden 292
Select Case, Anweisung 80
Select vermeiden 295
SelectionChange, Ereignis 66
SendMail, Methode 234
ShapeRange.Group, Methode 229
ShowDetail, Methode 217
Sich wiederholende Aufgaben, Makros verwenden für 25
Sicherungskopien von Arbeitsmappen mit aktuellem Datum 98
SkipIt, Sprungmarke 249
Sortieren
    Datenelemente benutzerdefiniert sortieren 209
    Datenfelder in Pivot-Tabellen alphabetisch sortieren 207
    Tabellenblätter 110
SourceData, Eigenschaft 204

SourceName, Eigenschaft 201
Spalten
alle einblenden 142
erste leere finden 152
leere löschen 145
SpecialCells, Methode 150
Speichern
E-Mail-Anhänge in Ordnern 247
vor dem Schließen 80
wenn Zellen geändert wurden 77
Speicherorte, vertrauenswürdige 36
Springen zwischen Modulen und Prozeduren 275
Sprungmarke 249
SQL
Befehlstyp 257
WHERE-Klausel 258
SrcRange, Variable 206
Standardmäßig ganzes Modul anzeigen, Registerkarte Editor 55
Standardsymbolleiste (VBE) 46
StatusBar 189
Statusleiste
Aktualisierung abschalten 291
Spalten anzeigen, nach denen gefiltert wird 186
strFormat, Variable 206
Strg+Bild ab 275
Strg+Bild auf 275
Strg+Tab 275
String, Datentyp 62

strLabel, Variable 206
Sub-Prozedur 49
Suchen
    Formeln 150
    geschützte Leerzeichen 178
    Zeichen für Wagenrücklauf 178
Suchen und Ersetzen 177
Symbolleiste (VBE) 46
Symbolleiste für den Schnellzugriff
    Makro einfügen 40

T

Tabellenblatt
    alle bis auf das aktive löschen 103
    als E-Mail-Anhang versenden 233
    ausblenden 105
    einblenden 107
    mehrere drucken 118
    nach Farben gruppieren 113
    per Doppelklick zoomen 126
    schützen 120
    Seitenwechsel ausblenden 292
    sortieren 110
    verschieben 109, 112
    Zugriff auf Zellen optimieren 296
    Tabellenblattereignisse 65
    Tab-Taste, Einzüge im Editor 55

Tastenkombination, Dialogfeld Makro aufzeichnen 27
Text in Spalten, Befehl 159
Textdateien
in Bereich importieren 269
binärer Zugriff 267
Daten lesen 268
Daten schreiben 268
öffnen 267
Textkonvertierungs-Assistent, Dialogfeld 159
Thehos, Andreas (Blogger) 286
ThisWorkbook, Objekt 194
Tipp, Icon 19
Top, Eigenschaft 227
Trim, Funktion 165
Trust Center, Dialogfeld 36

U

UCase, Funktion 112
Unterbrechen bei Fehlern, Registerkarte Allgemein 57
Unterstrich 51
UsedRange, Eigenschaft 143–145
User ID, ADO 263

V

Variablen
Atmt 246, 249
Boolean 63
deklarieren 63
Double 63
FileName 246, 249
Integer 63
LastColumn 154
LastRow 153
Long 63
ns 246, 249
Object 63
OLApp 235, 237, 239, 242, 244
OLMail 235, 237, 239, 242, 244
SrcRange 206
strFormat 206
String 62
strLabel 206
Variant 63
wbLinks 230
Variablendeklaration erforderlich, Registerkarte Editor 54
Variant, Datentyp 63
VBA 17
VBA-Hilfedateien 284
VBA-Modul
einfügen 48
entfernen 50

VBE. siehe Visual-Basic-Editor
VB-Editor. siehe Visual-Basic-Editor
Verankern, Registerkarte (VBA) 58
Verbindungszeichenfolge 262
verschieben
Tabellenblatt 109, 112
Vertrauenswürdige Speicherorte 36
Video2Brain, Kurse 287
VisibleDropDown 184
Visual Basic for Applications 17
Visual-Basic-Editor
aktivieren 19, 28
anpassen 53
Codefenster 47
Direktfenster 47
Hilfe zu 284
Kommentarblöcke verwenden 273
mehrere Codezeilen kopieren 275
Projekt-Explorer 47
springen zwischen Modulen und Prozeduren 275
Symbolleiste Bearbeiten einblenden 274
VBA-Modul einfügen 48
VBA-Modul löschen 50

W

Wagenrücklauf 177–178
Wahrheitswerte 63

Warnung, Icon 19
wbLinks, Variable 230
Websites
dailydoseofexcel.com 286
Excel-Funktionsnamen Deutsch und Englisch 33
Microsoft Office Dev Center 287
mrexcel.com 286
thehosblog.com 286
www.chandoo.org 286
www.contextures.com 286
YouTube 286
Werte
Formeln ersetzen durch Werte 157
leere Zellen durch Werte ersetzen 172
von Variablen 279
WHERE-Klausel 258
Width, Eigenschaft 227
With-Anweisung 294
With-End-With-Anweisung 150
Workbook_BeforeClose, Ereignis 68, 190
Workbook_BeforeSave, Ereignis 68
Workbook.Connections 260
Workbook_Open, Ereignis 68, 84
Workbook_SheetChange, Ereignis 68
Worksheet_Activate, Ereignis 67, 190
Worksheet_BeforeDoubleClick 216, 218
Worksheet_BeforeDoubleClick, Ereignis 67
Worksheet_BeforeRightClick, Ereignis 67

Worksheet_Calculate, Ereignis 67, 189–190
Worksheet_Change 292
Worksheet_Change, Ereignis 66
Worksheet_Deactivate, Ereignis 67, 190
Worksheet_SelectionChange, Ereignis 66
Write #, Textdatei schreiben 268

X

xlAverage 202
xlCalculationAutomatic 290
xlCalculationManual 290
xlCellTypeFormulas 150
xlConnectionTypeODBC 261
xlConnectionTypeOLEDB 261
xlCount 202
xlLandspace 231
xlMax 202
xlMin 202
xlPasteFormats 156
xlPasteValues 156
xlPortrait 231
xlSheetHidden 106
xlSheetVeryHidden 106
xlSheetVisible 108
xlsm, Dateierweiterung 35
xlSum 202

Y

YouTube, Schulungsvideos 286

Z

Zahlenformatierung, Datenelemente 204
ZÄHLENWENN, Funktion 180
Zeichen, Dateinamen 117
Zeilen
    alle einblenden 142
    erste leere finden 152
    leere löschen 143
Zeilenfortsetzungszeichen 51
Zeilenschaltung 177
Zellbereiche
    aus CSV-Datei importieren 269
Zelle
    Datei erst dann schließen, wenn Zelle Inhalt hat 96
    Datei nach Änderung von Zelle automatisch speichern 77
    führende und nachstehende Leerzeichen entfernen 165
    leere durch Wert ersetzen 172
    mit führenden Nullen versehen 169

Dummies Junior – die frechen »... für Dummies« für interessierte Kids und Jugendliche

- » Projekte zum Ausprobieren, Programmieren und Experimentieren
- » Mit pädagogischem Konzept
- » Viele Abbildungen
- » Verständliche Texte
- » In Workshops erprobt

C. Ermel, N. Rosenfeld

Spaß mit Elektronik für Dummies Junior

1. Auflage 2020 **ISBN:** 978-3-527-71705-7

198 Seiten

Format: 176 mm x 240 mm

Ladenpreis: 15,– €*

In diesem Buch lernst du, Schaltungen für coole Gadgets aufzubauen: eine Glückwunschkarte, die leuchtet, eine blinkende Weihnachtsbaumkugel, einen klingenden Draht und anderes mehr.

M. Ponce Kärgel

Hörspiel und Podcast selber machen für Dummies Junior

1. Auflage 2020 **ISBN:** 978-3-527-71704-0

192 Seiten

Format: 176 mm x 240 mm

Ladenpreis: 16,– €*

Sein eigener Produzent sein, wer will das nicht? Steig ein in die Welt der Hörspiele und Podcasts. Dieses Buch zeigt dir Schritt für Schritt, wie du deine eigenen Audiobeiträge produzieren und veröffentlichen kannst.

U. Schmid, K. Weitz, M. Siebers

Künstliche Intelligenz selber programmieren für Dummies Junior

1. Auflage 2019 **ISBN:** 978-3-527-71573-2

134 Seiten

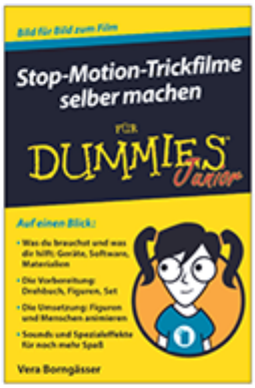
Format: 140 mm x 216 mm

Ladenpreis: 12,99 €*

Mit diesem Buch verstehst du, was Künstliche Intelligenz ist. Du findest heraus, ob es Künstliche Intelligenz bereits gibt. Und das Beste: Mit Hilfe kleiner Programme erkennst du, wie durch Computerprogramme Künstliche Intelligenz entsteht

V. Borngässer

Stop-Motion-Trickfilme selber machen für Dummies Junior



1. Auflage 2018 **ISBN:** 978-3-527-71484-1

144 Seiten

Format: 140 mm x 216 mm

Ladenpreis: 11,99 €*

Schritt für Schritt zum eigenen Stop-Motion-Video mit der richtigen Beleuchtung, krassen Geräuschen und Spezialeffekten. Hier erfährst du, wie es geht.

N. Bergner, Th. Leonhardt

Eigene Apps programmieren für Dummies Junior

2. Auflage 2019 **ISBN:** 978-3-527-71596-1

136 Seiten

Format: 140 mm x 216 mm

Ladenpreis: 11,99 €*

Hast du Lust, deine eigene App zu programmieren, die dann auch wirklich auf einem Android-Smartphone läuft? Mit dem kostenlosen App Inventor ist das gar nicht schwer.

* Der €-Preis gilt nur für Deutschland. Preisänderungen und Irrtümer vorbehalten.

WILEY END USER LICENSE AGREEMENT

Besuchen Sie www.wiley.com/go/eula, um Wiley's E-Book-EULA einzusehen.