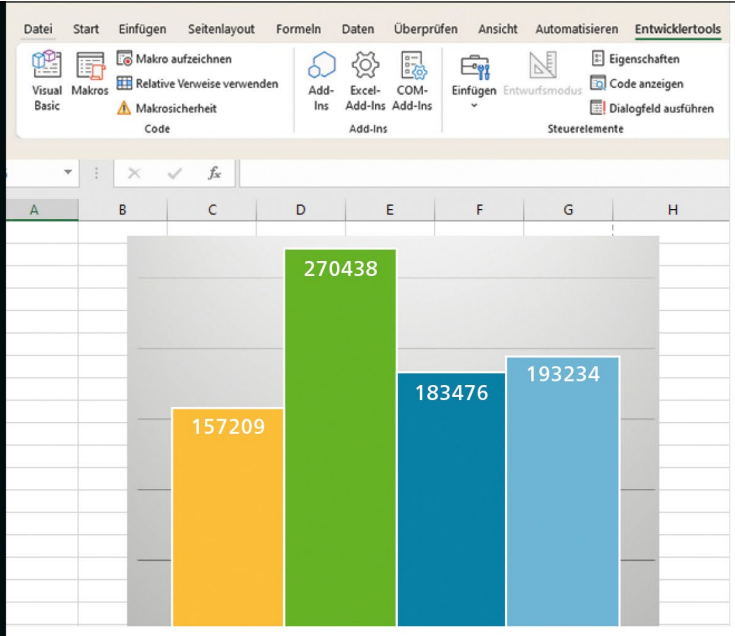


Excel 2016 bis 2024 und Microsoft 365

```
Sub DiagrammErstellen01()
Dim Bereich As Range
Dim s As String

  Set Bereich = Range(„A2:E6“)
  s = ActiveSheet.Name
  Charts.Add
  With ActiveChart
     .ChartType = xlColumnClustered
     .SetSourceData _
          Source:=Bereich, _
          PlotBy:=xlRows
     .HasTitle = True
     .ChartTitle.Text = s
  End With
End Sub
```

For Each ... Next
Worksheet_Chan
ListView, IsErr
xlValidateDate
Aut Show

Bernd Held

VBA mit Excel

Das umfassende Handbuch

- Das Standardwerk für Einsteiger und Fortgeschrittene
- Konzepte und Techniken der VBA-Programmierung
- Bewährte Lösungen für die tägliche Praxis

Über 650 geprüfte Makros für den sofortigen Einsatz zum Download

Rheinwerk Computing

Inhalt

6 Tabellen und Diagramme programmieren 353

7 Arbeitsmappen und Dateien programmieren 433

9 Ereignisse programmieren 555

10 Die VBE-Programmierung 591

11 Dialoge, Meldungen und UserForms programmieren 611

15 Fehlerbehandlung, Tuning und Schutz von VBA-Projekten sowie Support durch KI 895

16 Typische Verarbeitungsaufgaben aus der Praxis 915

Kapitel 4
Zellen und Bereiche programmieren

In diesem Kapitel dreht sich alles um Zellen, die durch das Objekt »Range« angesprochen werden. Das »Range«-Objekt kann aus einer einzigen Zelle oder aus mehreren Zellen bzw. aus einem Zellbereich oder mehreren Zellbereichen bestehen.

Das wichtigste Objekt in der Excel-Programmierung ist meiner Ansicht nach das Objekt `Range`. Wenn Sie mit diesem Objekt umgehen können, dann ist schon viel gewonnen.

In diesem Kapitel behandle ich die folgenden Themen:

- Zellen formatieren und konvertieren
- Zellen benennen und mit Namen arbeiten
- Formeln und Tabellenfunktionen einsetzen
- Gültigkeitsprüfungen durchführen
- Kommentare in Zellen verarbeiten

Die Beispiele aus diesem Kapitel zum Download

Sie finden alle Beispiele zu diesem Kapitel in der Datei *Zellen.xlsm*, die Sie auf der Verlagswebsite unter *www.rheinwerk-verlag.de/6028* herunterladen können.

4.1 Zahlenformat einstellen und/oder konvertieren

Beginnen wir, zunächst einige Zahlenformate für Zellen einzustellen.

In vielen Fällen reicht jedoch eine einfache Umformatierung von Daten leider nicht aus. Oft muss man Daten mithilfe von Konvertierungsfunktionen nachbereiten.

4.1.1 Zahlenformate einstellen (Datum und Zahl)

Die Zahlenformate finden Sie in der Oberfläche von Excel, wenn Sie beispielsweise einen Bereich markieren und die Tastenkombination Strg + 1 drücken. Daraufhin öffnet sich der Dialog ZELLEN FORMATIEREN, in dem Sie sämtliche Zahlenformate einstellen können.

Alle diese Formate können Sie aber auch automatisiert über Makros einstellen. Die dazu notwendige Eigenschaft heißt `NumberFormat`. Allerdings müssen Formate in der Entwicklungsumgebung so eingestellt werden, wie es im Land des Gründers von Microsoft üblich ist. Denken Sie also daran, dass das Tausendertrennzeichen in Amerika das Komma ist und das Dezimaltrennzeichen der Punkt. Auch bei den Datumsformatierungen wird mit Buchstabenkürzeln gearbeitet. So gelten die Kürzel aus Tabelle 4.1 bei den Datumsformaten, wenn wir von dem Datum 27.12.2021 ausgehen.

Englisch	Deutsch	Bedeutung	Ergebnis auf Deutsch
DD	TT	Tagesangabe zweistellig	27
DDD	TTT	Tagesangabe dreistellig	Mo
DDDD	DDDD	Tagesangabe vierstellig	Montag
MM	MM	Monatsangabe zweistellig	12
MMM	MMM	Monatsangabe dreistellig	Dez
MMMM	MMMM	Monatsangabe vierstellig	Dezember
YY	JJ	Jahresangabe zweistellig	21
YYYY	JJJJ	Jahresangabe vierstellig	2021

Tabelle 4.1 Die wichtigsten Formatkürzel für das Datum

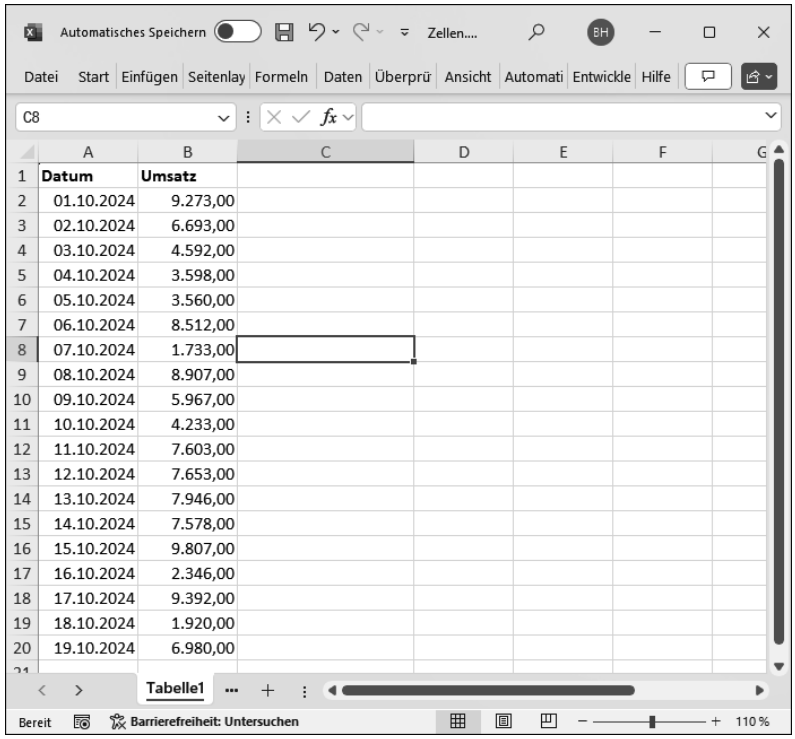
Die Konvertierung der in Englisch einzugebenden Formatcodes erfolgt automatisch. Sehen Sie sich dazu das Makro aus Listing 4.1 an. Dieses Makro versieht Spalte A von Tabelle1 mit dem Datumsformat `DD.MM.YYYY`.

```
Sub ZahlenFormatEinstellen()

 With Tabelle1

  'Datumsformat einstellen
  .Range("A:A").NumberFormat = "DD.MM.YYYY"

  'Zahlenformat: (Tausenderpunkt, Dezimalkomma mit zwei Nachkommastellen)
  .Range("B:B").NumberFormat = "#,###.00"

 End With

End Sub
```

Listing 4.1 Zahlenformat richtig einstellen

Mithilfe der Eigenschaft `NumberFormat` geben Sie das gewünschte Zahlenformat gleich für die komplette Spalte A an.

Für Spalte B wählen wir ein Zahlenformat, das den Tausenderpunkt und das Dezimalkomma enthält. Auch in diesem Fall müssen Sie die Formatierung so verwenden, wie sie in den USA üblich ist.



	A	B	C	D	E	F	G
1	**Datum**	**Umsatz**					
2	01.10.2024	9.273,00					
3	02.10.2024	6.693,00					
4	03.10.2024	4.592,00					
5	04.10.2024	3.598,00					
6	05.10.2024	3.560,00					
7	06.10.2024	8.512,00					
8	07.10.2024	1.733,00					
9	08.10.2024	8.907,00					
10	09.10.2024	5.967,00					
11	10.10.2024	4.233,00					
12	11.10.2024	7.603,00					
13	12.10.2024	7.653,00					
14	13.10.2024	7.946,00					
15	14.10.2024	7.578,00					
16	15.10.2024	9.807,00					
17	16.10.2024	2.346,00					
18	17.10.2024	9.392,00					
19	18.10.2024	1.920,00					
20	19.10.2024	6.980,00					

Abbildung 4.1 Die Formatierung wurde vorgenommen.

Wichtiger Hinweis

Verwenden Sie nicht die Eigenschaft `NumberFormatLocal`. Diese Eigenschaft erlaubt es Ihnen zwar, die landesüblichen Zahlenformate zu verwenden, aber wenn Sie dann die Arbeitsmappe über den Teich schicken, klappt das nicht mehr.

4.1.2 Zahlenformate einstellen (Text)

Etwas beschwerlich ist es auch, wenn Sie in Excel beispielsweise Telefonnummern mit führenden Nullen erfassen. Excel schluckt diese einfach. Sie wissen dann im Zweifelsfall nicht, wie viele Nullen Sie beim Telefonieren vorwählen müssen. Speziell in diesem Fall und in ähnlich gelagerten Fällen ist es sinnvoll, den entsprechenden

Bereich vorher über das Textformat zu formatieren. In diesem Fall bleiben die führenden Nullen erhalten.

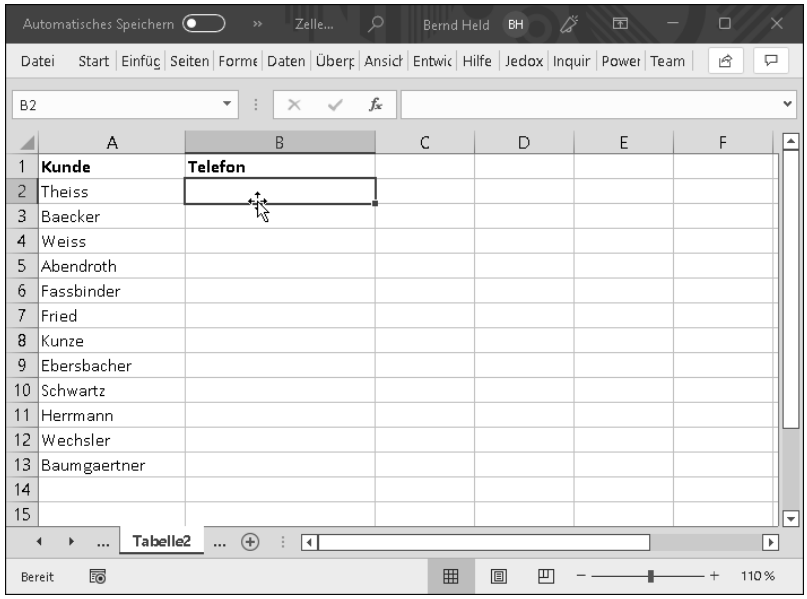


Abbildung 4.2 In Spalte B sollen Telefonnummern eingetragen werden.

Auch diese Einstellung wird über das Makro aus Listing 4.2 vorgenommen, bevor überhaupt Daten erfasst oder importiert werden. Denn sind die Nullen einmal weg, dann dauerhaft. Stellen Sie daher zuerst das Zahlenformat ein, und nehmen Sie danach die Befüllung vor.

```
Sub TextFormatEinstellen()
  Dim lngZeileMax As Long

  With Tabelle2
    lngZeileMax = .Cells(.Rows.Count, 1).End(xlUp).Row
    .Range("B2:B" & lngZeileMax).NumberFormat = "@"
  End With

End Sub
```

Listing 4.2 Einen verwendeten Bereich mit dem Zahlenformat »Text« formatieren

Das Formatkürzel `@` steht für das Zahlenformat `Text`. Dieses wird der Eigenschaft `NumberFormat` übergeben und dem vorher ermittelten Bereich zugewiesen. Alle Eingaben nach diesem Makrolauf werden als Text vorgenommen, d. h., die führenden Nullen werden von Excel nicht einkassiert.

4.1.3 Zahlenformate übertragen

Aufsetzend auf der gerade vorgestellten Aufgabe soll die Formatierung des verwendeten Bereichs in Spalte B der Tabelle auf den gleich großen Bereich von Spalte D übertragen werden. Das Makro zur Lösung dieser Aufgabenstellung sehen Sie in Listing 4.3.

```
Sub ZahlenformateÜbertragen()
  Dim rngQuelle As Range
  Dim rngZiel As Range
  Dim lngZeileMax As Long

  With Tabelle2

      lngZeileMax = .Cells(.Rows.Count, 1).End(xlUp).Row

      Set rngQuelle = .Range("B2:B" & lngZeileMax)
      Set rngZiel = .Range("D2:D" & lngZeileMax)

      rngZiel.NumberFormat = rngQuelle.NumberFormat

  End With

End Sub
```

Listing 4.3 Eingestelltes Zahlenformat aus einem Bereich auf einen anderen übertragen

Deklarieren Sie zu Beginn des Makros zwei Objektvariablen vom Typ `Range`, und geben Sie über die Anweisung `Set` bekannt, wo diese Bereiche in der Tabelle liegen. Danach übertragen Sie mithilfe der Eigenschaft `NumberFormat` die eingestellte Formatierung des Bereichs `rngQuelle` in den Bereich `rngZiel`.

4.1.4 Zellen mit Nullen auffüllen

Microsoft Excel wird oft als Schnittstelle zwischen mehreren Systemen verwendet. Dazu kann es einmal notwendig sein, Zellen mit führenden Nullen aufzufüllen, damit die Daten weiterverarbeitet werden können.

Bei der folgenden Aufgabenstellung liegen in Spalte A von Tabelle3 einige Zahlenwerte vor, die auf die Zeichenlänge 5 gebracht werden müssen. Sind zu wenige Ziffern erfasst, muss das Makro aus Listing 4.4 diese von links mit sichtbaren Nullen auffüllen.

```
Sub AuffüllenVonNullen()
 Dim rngZelle As Range

  With Tabelle3

   For Each rngZelle In .Range("A2:A10")

       rngZelle.Offset(0, 1).NumberFormat = "@"

       If Len(rngZelle.Value) < 5 Then
           rngZelle.Offset(0, 1).Value = _
           Right("00000" & rngZelle.Value, 5)
       Else
           rngZelle.Offset(0, 1).Value = rngZelle.Value
       End If

   Next rngZelle

   .Range("B:B").HorizontalAlignment = xlRight

  End With

End Sub
```

Listing 4.4 Eine Zahlenreihe mit führenden Nullen auffüllen

In einer Schleife des Typs `For Each ... Next` werden alle Zellen im angegebenen Bereich nacheinander verarbeitet. Innerhalb der Schleife wird für den um eine Spalte nach rechts versetzten Bereich das Textzahlenformat über die Eigenschaft `NumberFormat` festgelegt.

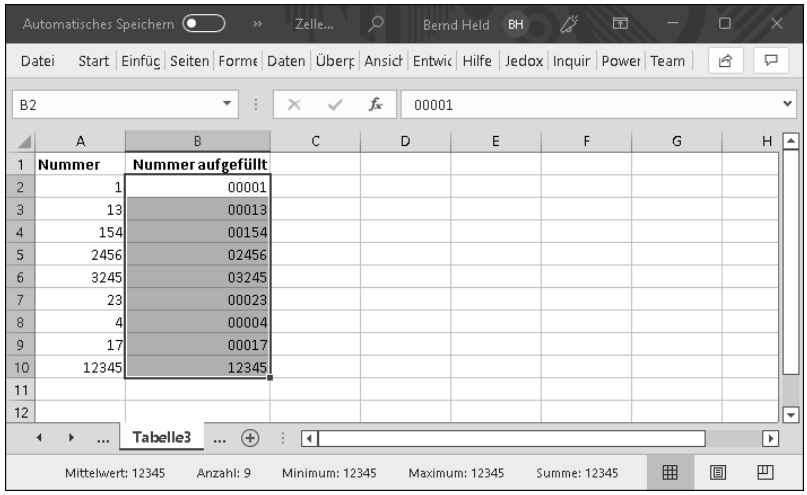
Die Verschiebung erfolgt über die Eigenschaft `Offset`. Diese Eigenschaft hat zwei Argumente:

- Im ersten Argument geben Sie die gewünschte Zeilenverschiebung an. Im vorliegenden Beispiel jedoch erfolgt hier keine Verschiebung, daher geben wir die Zeilenverschiebung mit `0` an.
- Im zweiten Argument wird die Spaltenverschiebung angegeben. Der Wert `1` bedeutet, dass die Spalte um eine Spalte weiter nach rechts verschoben wird.

In einer `If`-Bedingung fragen Sie über die Funktion `Len` die Anzahl der erfassten Zeichen ab. Ist die ermittelte Anzahl kleiner als 5 Zeichen, dann muss aufgefüllt werden.

Dazu setzen Sie die Funktion `Right` ein und verknüpfen den Inhalt mit der Funktion. Stimmt die Länge, dann übertragen Sie den Inhalt 1 : 1 in die Nebenzelle.

Über die Eigenschaft `HorizontalAlignment` richten Sie den Text rechtsbündig in der Zelle aus, indem Sie dieser Eigenschaft die Konstante `xlRight` zuweisen.



	A	B	C	D	E	F	G	H
1	Nummer	Nummer aufgefüllt						
2	1	00001						
3	13	00013						
4	154	00154						
5	2456	02456						
6	3245	03245						
7	23	00023						
8	4	00004						
9	17	00017						
10	12345	12345						
11								
12								

Abbildung 4.3 Die zu kurzen Werte wurden um vorangestellte Nullen komplettiert.

4.1.5 Einheitliches Datumsformat einstellen

Es ist wie verhext – alle machen, was sie wollen. In Abbildung 4.4 sehen Sie, dass Tabelle4 Datumsangaben in den unterschiedlichsten Formen enthält. Mehrere Datumseingaben werden gar nicht erst von Excel erkannt. Wer soll so eine Liste noch auswerten können? Hilft nichts, da muss ein Makro drüber und die Datumsformate vereinheitlichen. Dabei sollen die korrekten Datumsformate in Spalte C geschrieben werden.

Starten Sie das Makro aus Listing 4.5, um die Datumsformate zu vereinheitlichen.

```
Sub EinheitlichesDatumsformatEinstellen()
  Dim lngZeile As Long
  Dim lngZeileMax As Long

  With Tabelle4
    .Range("C1").Value = "Datum korrigiert"
    .Columns(3).NumberFormat = "DD.MM.YYYY"

    lngZeileMax = .Cells(.Rows.Count, 1).End(xlUp).Row
```

```
    For lngZeile = 2 To lngZeileMax

     If IsDate(.Cells(lngZeile, 1).Value) = True Then
        .Cells(lngZeile, 3).Value = CDate(.Cells(lngZeile, 1).Value)
     Else
        .Cells(lngZeile, 1).Interior.ColorIndex = 3
     End If

    Next lngZeile

  End With

 End Sub
```

Listing 4.5 Datumsformate vereinheitlichen und in von Excel interpretierbare Datumsangaben wandeln

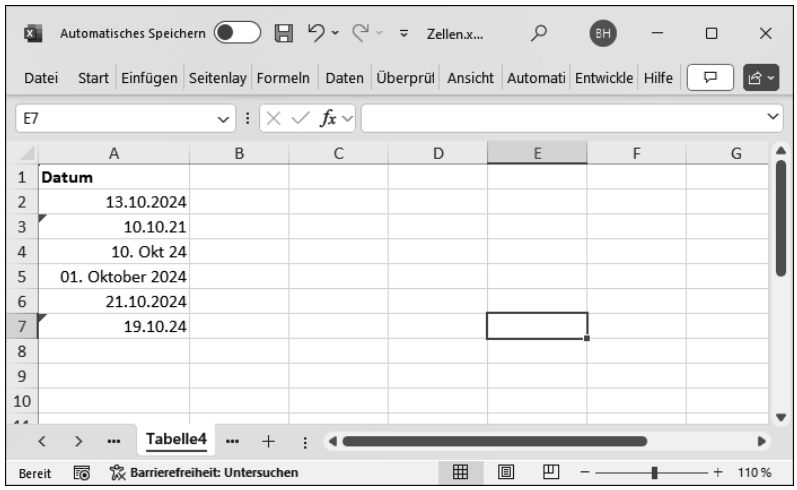
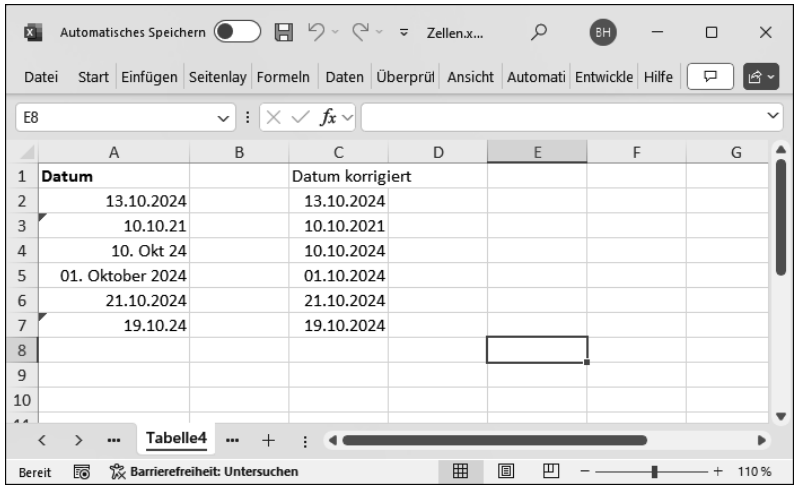


Abbildung 4.4 Die Datumsangaben müssen vereinheitlicht und vor allem von Excel erkannt werden.

Die eigentliche Umwandlung und Vereinheitlichung findet direkt in der `For ... Next`-Schleife statt. Dort prüfen Sie sicherheitshalber über die Funktion `IsDate`, ob Excel überhaupt eine Chance hat, ein datumsähnliches Format zu erkennen. Es kann nur dann die Umwandlungsfunktion `CDate` zum Einsatz kommen, wenn sichergestellt ist, dass es sich um ein brauchbares Datum handelt. Sollte in einer Zelle ein ungültiges Datum (z. B. »30.02.15«) stehen, dann wird der `Else`-Zweig der Bedingung durchlaufen. In diesem Fall wird die entsprechende Zelle rot angemalt.



	A	B	C	D	E	F	G
1	**Datum**		Datum korrigiert				
2	13.10.2024		13.10.2024				
3	10.10.21		10.10.2021				
4	10. Okt 24		10.10.2024				
5	01. Oktober 2024		01.10.2024				
6	21.10.2024		21.10.2024				
7	19.10.24		19.10.2024				
8							
9							
10							

Abbildung 4.5 Alle Datumsangaben wurden konvertiert und einheitlich formatiert.

4.1.6 Unerwünschte führende und nachgestellte Leerzeichen entfernen

Etwas lästig – um nicht zu sagen, etwas hinterlistig – sind auch führende oder nachgestellte Leerzeichen in Zellen. Diese unnützen Leerzeichen manuell zu entfernen ist eine Bestrafung, erst recht dann, wenn Sie diese Arbeit mit mehreren tausend Zeilen durchführen dürfen. Auch hier wird ein Makro eingesetzt, um alle Geschwindigkeitsrekorde zu brechen. Sehen Sie sich aber zuerst die Ausgangssituation aus Abbildung 4.6 an.

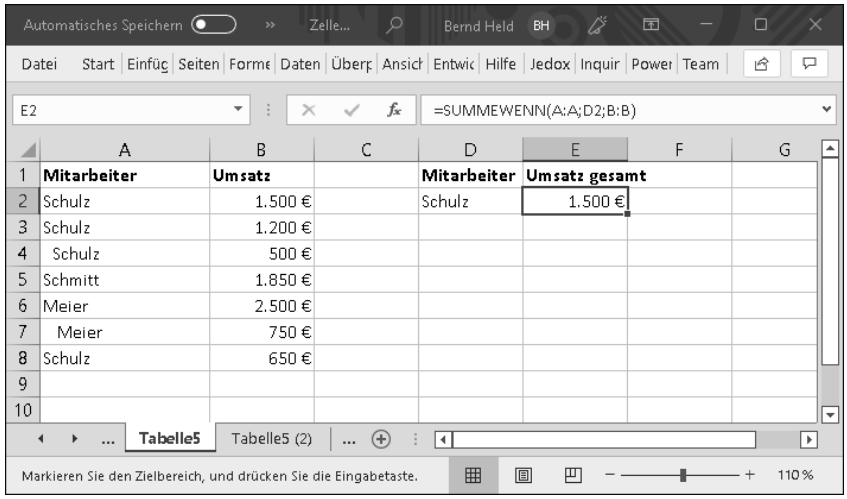
Mitarbeiter Schulz müsste eigentlich einen Gesamtumsatz von 3.850 € haben. Stattdessen liefert uns die Tabellenfunktion SUMMEWENN ein Ergebnis von 1.500 €. Das Problem hierbei sind vorangestellte und nachgestellte Leerzeichen im Datenbestand. Dafür kann die Tabellenfunktion nichts! Für die Funktion handelt es sich um verschiedene Mitarbeiter.

Sie können jetzt aber auch nicht hergehen und die Zellen einzeln kontrollieren, indem Sie die Zelle markieren, die Taste F2 drücken und oben in der Bearbeitungsleiste schauen, ob da vorangestellte oder nachgestellte leere Zeichen beim Namen stehen. Das ist ein Ding der Unmöglichkeit. Auch wenn dieser Effekt nur in ein paar Zellen von Tausenden auftritt, es bleibt doch ein mulmiges Gefühl, weil das Ergebnis daraus einfach nicht passt. Daher können Sie zur Sicherheit das Makro aus Listing 4.6 über den Datenbestand laufen lassen. Danach können Sie sicher sein, dass alle »Störenfriede« entfernt wurden.

```
Sub LeerzeichenEntfernen()
  'unerwünschte Leerzeichen am linken/rechten Rand entfernen
  'Trim entfernt Leerzeichen am rechten und linken Rand
```

```
  'LTrim entfernt Leerzeichen am linken Rand
  'RTrim entfernt Leerzeichen am rechten Rand
  Dim lngZeile As Long
  Dim lngZeileMax As Long

  With Tabelle5
    lngZeileMax = .UsedRange.Rows.Count

    For lngZeile = 2 To lngZeileMax

      .Range("A" & lngZeile).Value = Trim(.Range("A" & lngZeile).Value)

    Next lngZeile

  End With

End Sub
```

Listing 4.6 Unerwünschte Leerzeichen aus Zellen automatisch entfernen



	A	B	C	D	E	F	G
1	**Mitarbeiter**	**Umsatz**		**Mitarbeiter**	**Umsatz gesamt**		
2	Schulz	1.500 €		Schulz	1.500 €		
3	Schulz	1.200 €					
4	Schulz	500 €					
5	Schmitt	1.850 €					
6	Meier	2.500 €					
7	Meier	750 €					
8	Schulz	650 €					
9							
10							

Abbildung 4.6 Dass hier etwas nicht stimmt, ist offensichtlich.

Mithilfe der Funktion `Trim` entfernen Sie führende und nachgestellte Leerzeichen aus Zellen. Diese Funktion wird in der Schleife eingesetzt. Dabei werden die alten Zelleninhalte mit den »getrimmten« Zellenwerten überschrieben.

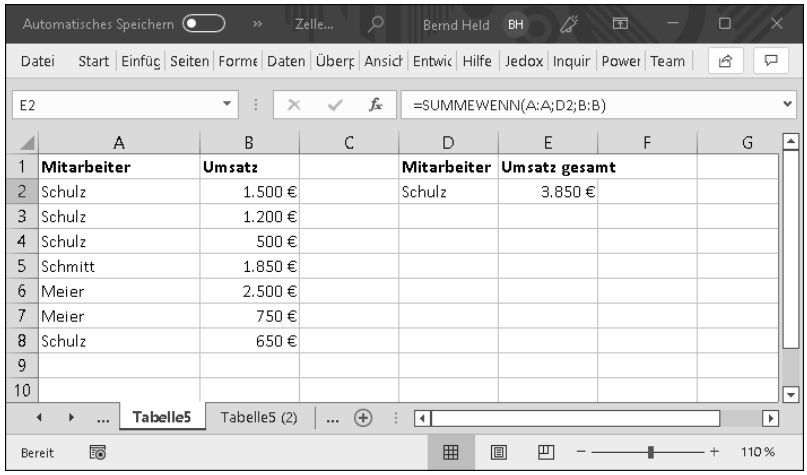


Abbildung 4.7 Alle Leerzeichen sind raus – Excel rechnet richtig.

4.1.7 Korrektur nach fehlerhaftem Import von Daten

Vielleicht kennen Sie dieses nicht gerade selten auftretende Phänomen? Sie importieren Daten aus einem Fremdsystem nach Excel und stellen nachher mehr oder weniger zufällig fest, dass Excel manche Zahlen wohl oder übel als Text interpretiert. Dieses »Fehlverhalten« von Excel ist nicht unbedingt gleich auf den ersten Blick erkennbar. Erst wenn Sie testweise einige Zellen markieren und in der Statusleiste die Summe mit den markierten Zahlen vergleichen, sehen Sie, dass etwas nicht stimmt. Schauen Sie sich dazu einmal Abbildung 4.8 an.

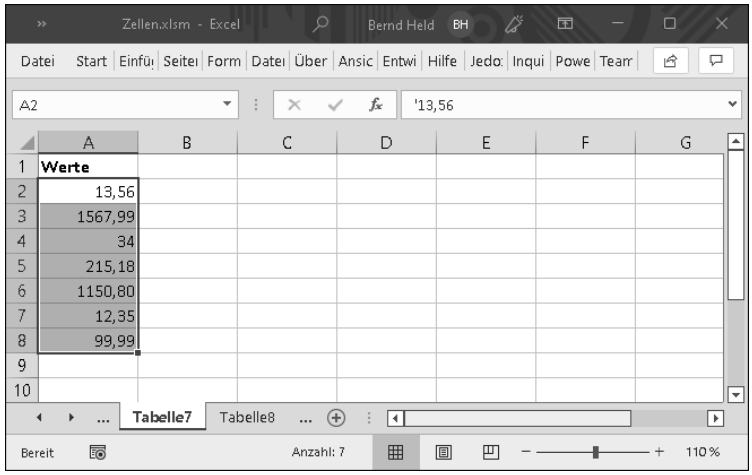


Abbildung 4.8 Nanu? Da wird ja gar nichts mehr berechnet!

Ja, genau. Excel erkennt diese importierten Daten nicht. Was nun? Schreiben Sie ein Makro wie das in Listing 4.7, das Excel zwingt, die Zahlen richtig zu erkennen.

```
Sub ZahlenwerteRichtigErkennen()
  Dim lngZeile As Long
  Dim lngZeileMax As Long

  With Tabelle7
    .Range("C1").Value = "korr. Werte"
    lngZeileMax = .Range("A" & .Rows.Count).End(xlUp).Row

    For lngZeile = 2 To lngZeileMax

      If IsNumeric(.Range("A" & lngZeile).Value) = True Then
         .Range("C" & lngZeile).Value = _
         .Range("A" & lngZeile).Value * 1
      Else
         .Range("C" & lngZeile).Value = 0
      End If

    Next lngZeile

  End With

End Sub
```

Listing 4.7 Zahlenwerte in Excel nachberechnen

Deklarieren Sie zwei Variablen vom Typ `Long`. Danach legen Sie über die Anweisung `With` fest, auf welcher Tabelle die Verarbeitung erfolgen soll. Nun ermitteln Sie, wie viele Zeilen in Spalte A der Tabelle belegt sind. Dazu arbeiten Sie mit der Eigenschaft `End`, über die Sie ausgehend von der letzten möglichen Zelle einer Tabelle zur letzten belegten Zelle nach oben gehen und über die Eigenschaft `Row` die dazugehörige Zeilennummer auslesen. Sie haben jetzt die Zeile ermittelt, bis zu der die Verarbeitung durchgeführt werden soll. In einer Schleife arbeiten Sie sich also beginnend von der zweiten Zeile bis zur letzten belegten Zeile zeilenweise durch die Tabelle. Innerhalb der Schleife schreiben Sie das Ergebnis der Konvertierung in Spalte C. Dabei prüfen Sie zunächst über die Funktion `IsNumeric`, ob der jeweilige Zelleninhalt aus Spalte A numerisch ist. Wenn nicht, dann schreiben Sie in Spalte H den Wert 0. Im anderen Fall zwingen Sie Excel sicherheitshalber, die ermittelte Zahl wirklich als Zahl zu erkennen, indem Sie sie mit dem Wert 1 multiplizieren. Dadurch führt Excel eine Neuberechnung der Zelle durch, und der Wert wird richtig erkannt.

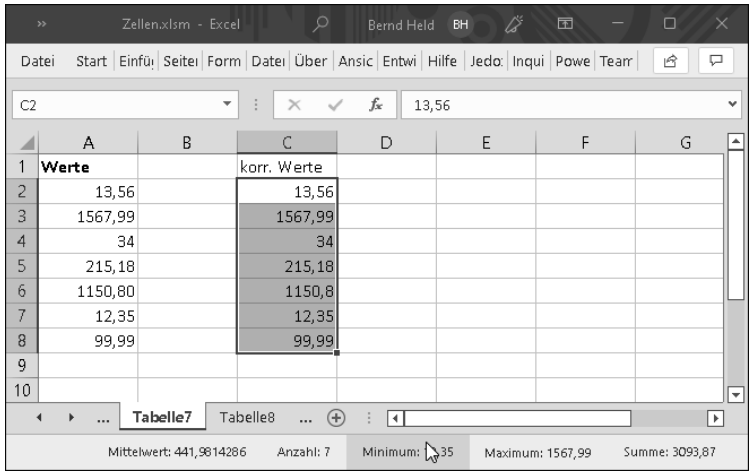


Abbildung 4.9 Jetzt erkennt Excel die Zahlenwerte.

4.1.8 Die Position des Minuszeichens umstellen

Manche Warenwirtschaftssysteme haben die Angewohnheit, negative Zahlen mit einem Minuszeichen am rechten Rand auszustatten. Microsoft Excel erwartet die Minuszeichen jedoch auf der linken Seite und kann mit rechtslastigen Minuszeichen leider nichts anfangen. Haben Sie einmal versucht, solche Zellen manuell zu bereinigen? Diese Nackenschmerzen auslösende Arbeit muss nun wirklich nicht sein – wie schnell verrutschen Sie einmal um eine Stelle oder löschen aus Versehen ein oder auch mehrere Zeichen!

Sehen Sie sich zunächst einmal die Ausgangssituation in Abbildung 4.10 etwas genauer an.

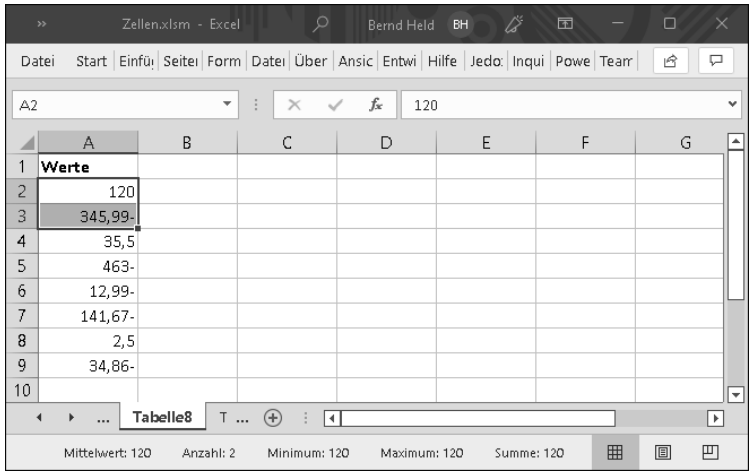


Abbildung 4.10 Das Ergebnis in der Statusleiste stimmt ja in 100 Jahren nicht!

Was genau soll denn das gewünschte Makro tun? Wie drücken Sie in der VBA-Sprache aus, was Sie tun möchten? Es reicht nicht zu sagen: »Bitte entferne auf der rechten Seite das Minuszeichen, und hänge es stattdessen am linken Rand wieder an.« Einen solchen Befehl gibt es in VBA nicht. Sie müssen versuchen, diese Aufgabe schrittweise anzugehen und zu lösen:

1. Prüfe, ob am rechten Rand einer Zelle ein Minuszeichen steht.
2. Wenn ja, dann übertrage vom linken Rand aus gesehen alle Zeichen bis auf das letzte (Minuszeichen).
3. Versorge die Zelle noch mit einem führenden Minuszeichen.

Arbeiten Sie diese einzelnen Schritte nacheinander ab, wie in Listing 4.8 gezeigt.

```
Sub MinuszeichenUmstellen()
  Dim lngZeile As Long
  Dim lngZeileMax As Long

  With Tabelle8
    .Range("C1").Value = "korr. Werte"
    lngZeileMax = .Range("A" & .Rows.Count).End(xlUp).Row

    For lngZeile = 2 To lngZeileMax

      If Right(.Range("A" & lngZeile).Value, 1) = "-" Then
       .Range("C" & lngZeile).Value = _
       Replace(.Range("A" & lngZeile).Value, "-", "") * -1
      Else
       .Range("C" & lngZeile).Value = _
       .Range("A" & lngZeile).Value
      End If

    Next lngZeile

  End With

End Sub
```

Listing 4.8 Die Position des Minuszeichens tauschen

Mit der Funktion `Right` prüfen Sie eine bestimmte Anzahl von Zeichen einer Zelle von rechts gesehen. Indem Sie dieser Funktion den Wert `1` im zweiten Argument übergeben, werfen Sie einen Blick auf das eine Zeichen ganz rechts in der jeweiligen Zelle. Wenn dieses Zeichen dem Minuszeichen entspricht, dann können Sie die Funktion `Replace` auch hier einsetzen, um das Zeichen am rechten Rand zu eliminieren. Danach

multiplizieren Sie das Ergebnis mit dem Wert -1. Damit wird das Minuszeichen an der gewünschten Stelle automatisch gesetzt.

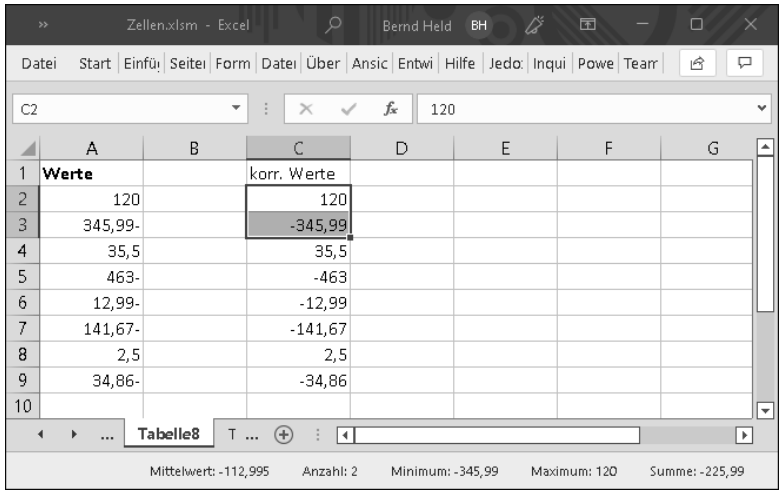


Abbildung 4.11 Excel kann jetzt alle Zellen in Spalte C berechnen.

4.1.9 Daten umschlüsseln

Um Daten weiterverarbeiten zu können, müssen sie hin und wieder auch umgeschlüsselt werden. Sehen Sie sich dazu die Ausgangssituation in Abbildung 4.12 an.

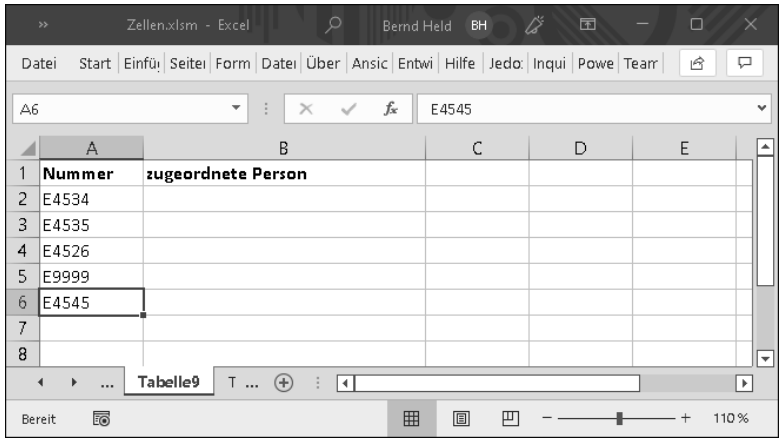


Abbildung 4.12 Welche Nummern werden von welcher Person verarbeitet?

Beantworten Sie diese Frage, indem Sie das Makro aus Listing 4.9 starten.

```
Sub UmschlüsselungVornehmen()
  Dim lngZeile As Long
  Dim lngZeileMax As Long
```

```
With Tabelle9

    lngZeileMax = .Range("A" & .Rows.Count).End(xlUp).Row

    For lngZeile = 2 To lngZeileMax

        Select Case Trim(UCase(.Cells(lngZeile, 1).Value))

        Case "E4534", "E9999"
            .Cells(lngZeile, 2).Value = "Christian"
        Case "E4535"
            .Cells(lngZeile, 2).Value = "Irina"
        Case "E4536"
            .Cells(lngZeile, 2).Value = "Erik"
        Case Else
            .Cells(lngZeile, 2).Value = "N.N"
        End Select

    Next lngZeile

End With

End Sub
```

Listing 4.9 Der jeweiligen Nummer den passenden Mitarbeiter zuordnen

Im Makro aus Listing 4.9 sorgen Sie im Innern der `For ... Next`-Schleife dafür, dass gegebenenfalls vorgestellte und nachgestellte Leerzeichen vorab berücksichtigt werden.

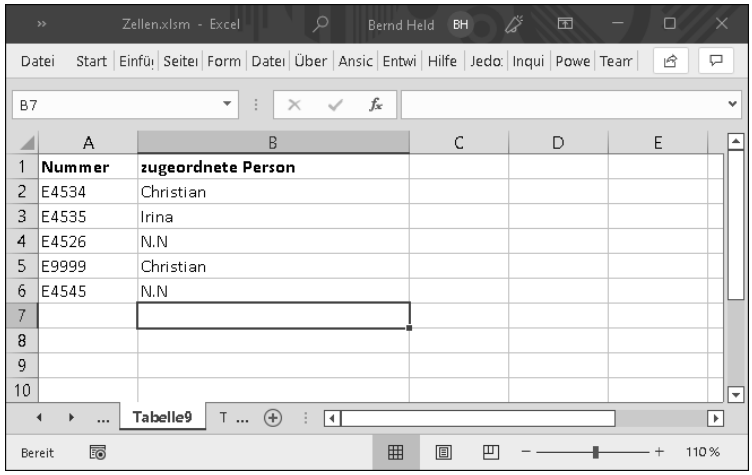


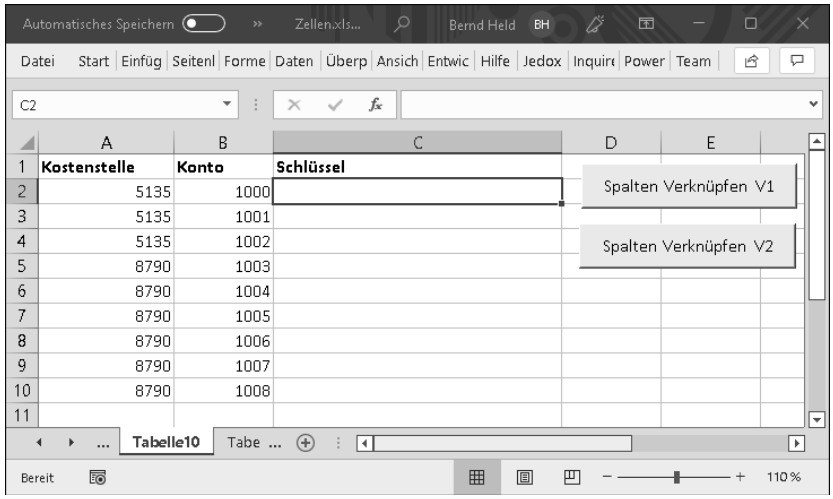
Abbildung 4.13 Die Zuordnung passt und kann weiter angepasst werden.

Dazu verwenden Sie die Funktion `Trim`, die Sie bereits kennengelernt haben. Des Weiteren sorgen Sie mit der Funktion `UCase` dafür, dass Excel nicht zwischen Groß- und Kleinschreibung unterscheidet.

In einer `Select Case`-Anweisung ordnen Sie die einzelnen Nummern den dazugehörigen Mitarbeitern zu.

4.1.10 Einen eindeutigen Schlüssel aus mehreren Spalten basteln

Wenn Sie beispielsweise gern mit der Tabellenfunktion SVERWEIS arbeiten, dann brauchen Sie einen eindeutigen Schlüssel, um die gewünschten Daten zuverlässig zu finden. Gibt es zu einem Schlüssel mehrere Datensätze, dann findet SVERWEIS immer nur den ersten Satz. Daher bilden Sie einen Schlüssel über mehrere Spalten. Schauen Sie sich zunächst einmal den Sachverhalt in Abbildung 4.14 an.



	A	B	C	D	E
1	**Kostenstelle**	**Konto**	**Schlüssel**		
2	5135	1000			
3	5135	1001			
4	5135	1002			
5	8790	1003			
6	8790	1004			
7	8790	1005			
8	8790	1006			
9	8790	1007			
10	8790	1008			
11					

Abbildung 4.14 Die beiden Spalten A und B sollen verknüpft und in Spalte C ausgegeben werden.

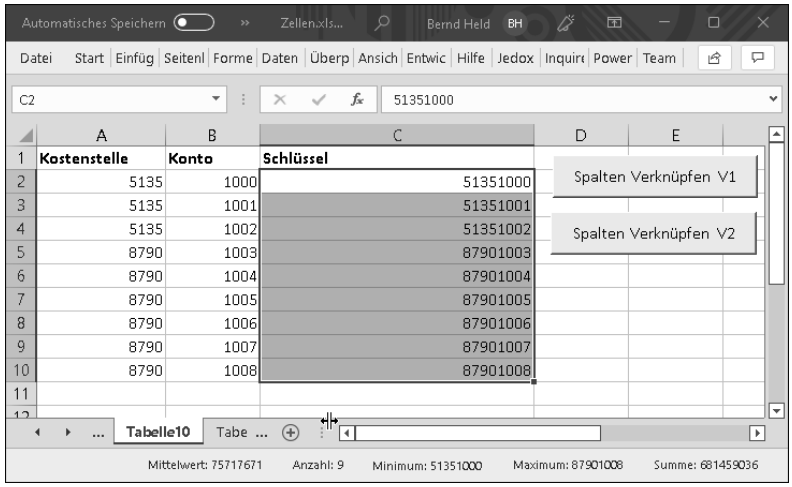
Für diese Aufgabe soll aber keine Schleife verwendet werden. Gerade bei großen Datenmengen würde eine Schleife unter Umständen zu viel Zeit benötigen. Es soll aber auch keine Formel in Spalte C stehen, über die die beiden Spalten A und B verknüpft werden, da Formeln die Berechnung von Excel etwas lähmen. Was nun? Schauen Sie sich dazu das Makro aus Listing 4.10 an.

```
Sub SpaltenVerknuepfenFuerEindeutigenSchluesselV1()

  With Tabelle10

    With .Range("C2:C" & .UsedRange.Rows.Count)
```

```
            .Value = Evaluate(.Offset(0, -2).Address & "&" _
                              & .Offset(0, -1).Address)
        End With

    End With

End Sub
```

Listing 4.10 Pfeilschnell zwei Spalten miteinander verknüpfen – Variante 1

Mithilfe der Methode `Evaluate` simulieren Sie quasi eine Formel, die aber eben nicht als Formel in eine Zelle geschrieben, sondern als Festwert in der Zelle abgelegt wird. Da Sie vorher auf den kompletten Zielbereich referenzieren, wird die »Formel« als Festwert in allen Zellen eingefügt und entsprechend angepasst.



	A	B	C	D	E
1	Kostenstelle	Konto	Schlüssel		
2	5135	1000	51351000		
3	5135	1001	51351001		
4	5135	1002	51351002		
5	8790	1003	87901003		
6	8790	1004	87901004		
7	8790	1005	87901005		
8	8790	1006	87901006		
9	8790	1007	87901007		
10	8790	1008	87901008		
11					

Abbildung 4.15 Schneller geht's nicht – der eindeutige Schlüssel wurde generiert.

Alternativ können Sie die gleiche Aufgabe auch über eine andere Variante lösen, wie in Listing 4.11 gezeigt.

```
Sub SpaltenVerknuepfenFuerEindeutigenSchluesselV2()

  With Tabelle10

    With .Range("C2:C" & .UsedRange.Rows.Count)
        .NumberFormat = "0"
        .FormulaR1C1 = "=RC[-2]&RC[-1]"
        .Copy
        .PasteSpecial Paste:=xlPasteValues
    End With
```

```
    Application.CutCopyMode = False

  End With

End Sub
```

Listing 4.11 Pfeilschnell zwei Spalten miteinander verknüpfen – Variante 2

Auch dieser Ansatz zum Verknüpfen mehrerer Spalten kommt ohne Schleife aus. Dabei wird die Eigenschaft `FormulaR1C1` genutzt und eine Formel geschrieben. Dabei steht `R` für *row* (Zeile) und `C` für *column* (Spalte). Der Bezug für die Formel, wenn Sie von Spalte C beginnen, ist `-2`, um Spalte A zu adressieren, und `-1` zeigt auf Spalte B. Einen Zeilenversatz gibt es dabei nicht, da der Schlüssel in der gleichen Zeile gebildet werden muss. Zunächst wird also die Formel in den verwendeten Bereich von Spalte C geschrieben, dann wird dieser Bereich kopiert und über die Methode `PasteSpecial` über Einsatz der Konstanten `xlPasteValues` als Festwert eingefügt. Über die Eigenschaft `CutCopyMode`, der Sie den Wert `False` zuweisen, sorgen Sie dafür, dass der Laufrahmen um den Bereich, der beim Kopieren entsteht, wieder zurückgesetzt wird.

4.2 Zellen, Rahmen und Schriften formatieren

Neben den bereits vorgestellten Zahlenformaten gibt es die normalen Formate, die für Zellen, Schriften und Rahmen eingesetzt werden können. Einige davon habe ich im Buch schon behandelt. An dieser Stelle erfolgt nun eine Vertiefung.

Hinweis

Sie finden alle folgenden Makros in der Datei *Zellen.xlsm* im Modul `mdl_Formatieren`.

4.2.1 Schriftart ermitteln

In Windows stehen Ihnen Hunderte verschiedener Schriftarten zur Verfügung. Möchten Sie prüfen, welche Schriftart beispielsweise in der momentan aktiven Zelle verwendet wird, dann starten Sie das Makro aus Listing 4.12.

```
Sub SchriftartErmitteln()

  MsgBox ActiveCell.Font.Name

End Sub
```

Listing 4.12 Schriftart ermitteln

Kapitel 6
Tabellen und Diagramme programmieren

Das Objekt »Worksheet« symbolisiert das Tabellenblatt. Tabellenblätter lassen sich individuell modifizieren. Sie können Tabellenblätter einfügen, umbenennen, löschen, drucken, kopieren, verschieben und vieles mehr. Über das Objekt »ChartObject« erstellen Sie Diagramme, die Sie in Tabellen einbetten.

In diesem Kapitel erfahren Sie anhand ausgesuchter Beispiele aus der täglichen Praxis mehr über den Einsatz von Eigenschaften und Methoden des Objekts `Worksheet`. Auch die Themen Pivot-Tabellen und Diagramme werde ich in diesem Kapitel behandeln.

Kapitelbegleitende Beispiele zum Download

Sie finden alle Beispiele in der Datei *Tabellen.xlsm* aus dem Downloadpaket, das Sie von *www.rheinwerk-verlag.de/6028* herunterladen können.

6.1 Tabellen einfügen

Hinweis

Die Datei *Tabellen.xlsm* enthält im Modul `mdl_Allgemein` alle folgenden Makros.

Standardmäßig bietet Excel Ihnen bei der Erstellung einer neuen Arbeitsmappe drei Tabellenblätter an. Wenn Sie weitere hinzufügen möchten, setzen Sie die Methode `Add` ein wie in Listing 6.1. Das neu eingefügte Tabellenblatt wird immer vor dem aktiven Tabellenblatt der Arbeitsmappe eingefügt.

```
Sub TabelleEinfügen()

  Worksheets.Add

End Sub
```

Listing 6.1 Neues Tabellenblatt einfügen

Möchten Sie ein Tabellenblatt an einer bestimmten Position einfügen, starten Sie das Makro aus Listing 6.2.

```
Sub TabelleAnPositionEinfügen()

  Worksheets.Add Before:=ThisWorkbook.Worksheets(1)

End Sub
```

Listing 6.2 Neues Tabellenblatt als erstes Blatt in eine Mappe einfügen

In Listing 6.2 wurde die neue Tabelle zu Beginn der Arbeitsmappe, also als erste Tabelle, eingefügt. Das bisherige Tabellenblatt mit dem Index 1 wird dann eine Position nach rechts geschoben. Möchten Sie die neue Tabelle ganz am Ende einfügen, also ganz rechts, setzen Sie das Makro aus Listing 6.3 ein.

```
Sub TabelleAmEndeEinfügen()

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

End Sub
```

Listing 6.3 Neues Tabellenblatt am Ende der Arbeitsmappe einfügen

Um die gewünschte Einfügeposition des neuen Tabellenblattes zu ermitteln, müssen Sie zuerst herausfinden, wie viele Tabellenblätter bereits in der Arbeitsmappe enthalten sind. Dabei hilft Ihnen die Eigenschaft `Count`. Sie liefert die Anzahl der Tabellenblätter. Danach brauchen Sie nur noch den Parameter `After` anzugeben, und das neue Tabellenblatt wird als letztes Tabellenblatt in die Arbeitsmappe eingefügt.

6.2 Tabellenblätter benennen

Excel vergibt beim Einfügen von Tabellennamen selbstständig Namen, die sich aus dem Ausdruck TABELLE und einer fortlaufenden Zahl zusammensetzen. Wenn Sie andere Namen verwenden möchten, können Sie dies jederzeit tun.

6.2.1 Eine neue Mappe erstellen, zwölf Monatstabellen anlegen und benennen

Bei der nächsten Aufgabe (siehe Listing 6.4) soll eine neue Mappe mit zwölf Tabellen erstellt werden. Diese Tabellen sollen nach den Monatsnamen benannt werden.

```
Sub MappeMit12MonatenAnlegen()
 Dim intAnz As Integer
 Dim wkbMappe As Workbook
 Dim wksBlatt As Worksheet

 intAnz = Application.SheetsInNewWorkbook
 Application.SheetsInNewWorkbook = 12
 Set wkbMappe = Workbooks.Add
 Application.SheetsInNewWorkbook = intAnz

 For Each wksBlatt In wkbMappe.Worksheets

   wksBlatt.Name = MonthName(wksBlatt.Index)

 Next wksBlatt

End Sub
```

Listing 6.4 Eine neue Mappe mit zwölf Tabellen anlegen und diese nach Monatsnamen benennen

Ermitteln Sie zunächst über die Eigenschaft `SheetsInNewWorkbook`, welche Applikationseinstellung bezüglich der Tabellenanzahl festgelegt ist, wenn eine neue Mappe angelegt wird. Speichern Sie diesen Wert in der Variablen `intAnz` zwischen. Jetzt ändern Sie den Wert dieser Eigenschaft in zwölf Tabellen. Durch die Methode `Add`, die auf das Auflistungsobjekt `Workbooks` angewendet wird, wird nun eine neue Arbeitsmappe mit zwölf Tabellen erstellt. Stellen Sie jetzt am besten gleich wieder die vorher eingestellte Anzahl der angebotenen Tabellen ein. Dazu weisen Sie der Eigenschaft `SheetsInNewWorkbook` den Inhalt der Variablen `intAnz` zu.

Setzen Sie danach eine Schleife des Typs `For Each ... Next` ein, in der Sie alle zwölf Tabellen nacheinander verarbeiten. In der Schleife benennen Sie die Tabellen. Dazu setzen Sie die Funktion `MonthName` ein. Diese Funktion benötigt einen Wert zwischen 1 und 12, um den gewünschten Monat direkt aus der Windows-Systemsteuerung in der dort eingestellten Landessprache zu holen. Diesen Wert leiten Sie über die Eigenschaft `Index` der jeweiligen Tabelle ab.

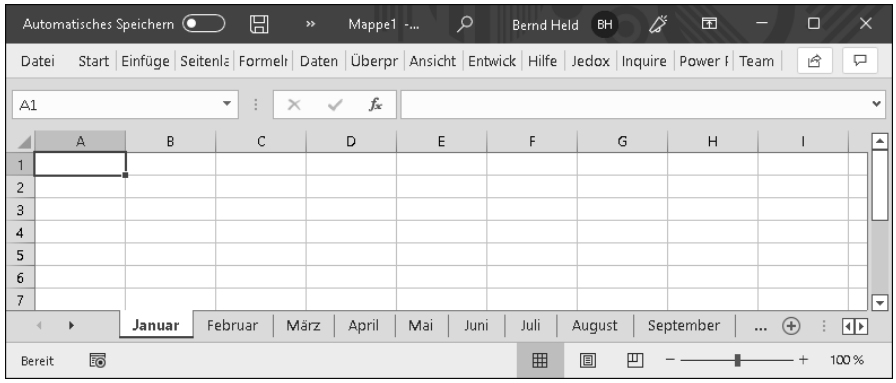


Abbildung 6.1 Die Monatstabellen wurden automatisch angelegt.

6.2.2 Eine neue Mappe mit den nächsten 14 Tagen anlegen

Im Beispiel aus Listing 6.5 wird eine neue Arbeitsmappe zunächst mit nur einer Tabelle angelegt. Danach werden über eine Schleife 14 weitere Tabellen hinzugefügt und mit einem fortlaufenden Datum versehen.

```
Sub TabellenMitDatumEinfügen()
 Dim inTabz As Integer
 Dim intAnz As Integer
 Dim wkbMappe As Workbook
 Dim wksBlatt As Worksheet

 intAnz = Application.SheetsInNewWorkbook
 Application.SheetsInNewWorkbook = 1
 Set wkbMappe = Workbooks.Add
 Application.SheetsInNewWorkbook = intAnz

    For inTabz = 1 To 14
        wkbMappe.Worksheets.Add after:=wkbMappe.Worksheets(Worksheets.Count)
        wkbMappe.Worksheets(Worksheets.Count).Name = Date + inTabz
    Next inTabz

End Sub
```

Listing 6.5 Eine neue Mappe für die nächsten 14 Tage wird angelegt.

Setzen Sie die Eigenschaft `SheetsInNewWorkbook` auf den Wert `1`, und fügen Sie anschließend in einer `For ... Next`-Schleife weitere 14 Tabellen über den Einsatz der Methode

`Add` hinzu. Innerhalb der Schleife benennen Sie Tabellen beginnend vom aktuellen Tagesdatum. Mit jedem Schleifendurchlauf wird der Schleifenzähler, der gleichzeitig der Tageszähler ist, um den Wert 1 inkrementiert.

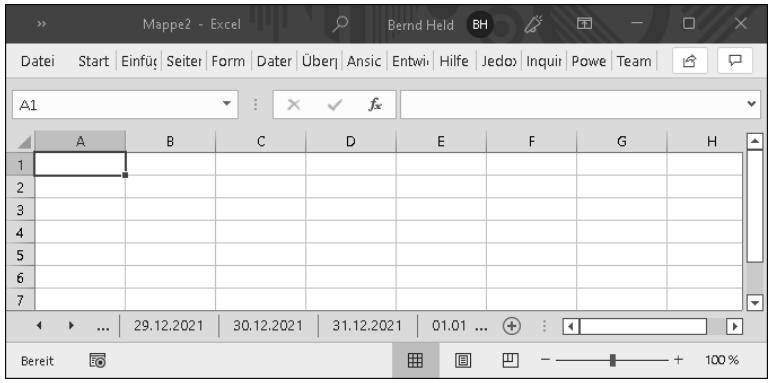


Abbildung 6.2 Eine neue Mappe für die nächsten 14 Tage steht bereit.

6.2.3 Tabelle einfügen und gleichzeitig benennen

Selbstverständlich können Sie das Einfügen von neuen Tabellenblättern und deren Benennung auch in einem Aufwasch erledigen, wie Listing 6.6 unter Beweis stellt.

```
Sub TabelleEinfügenUndBenennen()

  Worksheets.Add.Name = "Tabelle10"

End Sub
```

Listing 6.6 Neue Tabelle einfügen und benennen in einem Schritt

Allerdings müssen Sie sich sicher sein, dass der Name nicht schon in Verwendung ist, da es sonst zu einem Laufzeitfehler kommt.

6.3 Tabellen markieren

Um eine einzige Tabelle zu markieren, können Sie den Befehl `Worksheets("Tabelle2").Select` anwenden. Sollen es ein paar Tabellen mehr sein, dann wenden Sie das Makro aus Listing 6.7 an.

```
Sub MehrereTabellenMarkieren()

  Sheets(Array("Tabelle1", "Tabelle2")).Select

End Sub
```

Listing 6.7 Mehrere Tabellen markieren

Mithilfe der Funktion `Array` bilden Sie ein Datenfeld, in das Sie die Namen der Tabellen aufnehmen, die Sie markieren möchten.

Soll diese Lösung ein wenig dynamischer sein, dann markieren Sie in der nächsten Aufgabe einmal alle Tabellen einer Arbeitsmappe mit Ausnahme der ersten Tabelle. Wie das geht, entnehmen Sie dem Makro aus Listing 6.8.

```
Sub MehrereTabellenMarkierenÜberArray()
  Dim lngZ As Long
  Dim intTab As Integer
  Dim Vardat() As Long

    intTab = ThisWorkbook.Worksheets.Count

    ReDim Vardat(1 To intTab - 1)

    For lngZ = 2 To intTab
      Vardat(lngZ - 1) = lngZ
    Next lngZ

    ThisWorkbook.Worksheets(Vardat).Select

End Sub
```

Listing 6.8 Mehrere Tabellen markieren (nur nicht die erste)

Ermitteln Sie im ersten Schritt die Gesamtzahl der Tabellen, die sich in der aktiven Arbeitsmappe befinden, und speichern Sie diese Information in der Variablen `intTab`. Danach definieren Sie mit der Anweisung `ReDim` ein Datenfeld in der Größe der Anzahl der Tabellen in der Arbeitsmappe. Von dieser ermittelten Größe subtrahieren Sie den Wert 1, da Sie die erste Tabelle nicht markieren möchten. In einer Schleife füllen Sie dann das Datenfeld. Am Ende der Schleife stehen die Namen aller Tabellen im Datenfeld. Markieren Sie anschließend alle im Datenfeld stehenden Tabellen mit der Methode `Select`.

6.4 Tabellenblätter gruppieren

In Excel haben Sie die Möglichkeit, Ihre Arbeit an einem Tabellenblatt automatisch auch für andere Tabellenblätter gültig zu machen. Dazu gruppieren Sie die einzelnen Tabellenblätter. Manuell klappt das, indem Sie die Strg-Taste gedrückt halten und die einzelnen Tabellenregister mit der linken Maustaste anklicken. Das Ergebnis dieser Aktion erreichen Sie selbstverständlich auch mit VBA. Im Folgenden erfahren Sie, wie Sie das machen.

6.4.1 Mehrere Tabellen gruppieren

Die Funktion `Array` ermöglicht es Ihnen, eine durch Kommas getrennte Liste von Werten (hier Tabellennamen) anzugeben (siehe Listing 6.9). Auch hier ist wieder die `On Error`-Anweisung wichtig, um eine Fehlermeldung zu vermeiden, falls eines der Tabellenblätter nicht vorhanden ist.

```
Sub MehrereTabellenblätterMarkieren()

  On Error Resume Next
  Sheets(Array("Tabelle2", "Tabelle3", "Tabelle5")).Select

End Sub
```

Listing 6.9 Mehrere Tabellenblätter gruppieren

6.4.2 Alle Tabellen gruppieren

Möchten Sie alle Tabellenblätter einer Arbeitsmappe gruppieren, können Sie die Tabellenblätter in ein Array einlesen und anschließend gruppieren. Dazu wenden Sie das Makro aus Listing 6.10 an.

```
Sub AlleTabellenMarkieren()
  Dim lngZ As Long
  Dim lngTab As Long
  Dim TabArray() As Long

    lngTab = ThisWorkbook.Worksheets.Count

    ReDim TabArray(1 To lngTab)
    On Error Resume Next
```

```
    For lngZ = 1 To lngTab
        TabArray(lngZ) = 1
    Next lngZ

    ThisWorkbook.Worksheets(TabArray).Select

End Sub
```

Listing 6.10 Alle Tabellenblätter einer Arbeitsmappe gruppieren

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

6.4.3 Gruppierte Tabellen übertragen

Im nächsten Beispiel in Listing 6.11 werden alle gruppierten Tabellen in eine neue Arbeitsmappe eingefügt.

```
Sub GruppierteTabellenInNeueMappeTransferieren()
 Dim wksBlatt As Worksheet

  For Each wksBlatt In ActiveWindow.SelectedSheets
    wksBlatt.Copy
  Next wksBlatt

End Sub
```

Listing 6.11 Gruppierte Tabellen in neue Arbeitsmappe überführen

Setzen Sie die Eigenschaft `SelectedSheets` ein, um alle markierten Tabellenblätter zu ermitteln. Kopieren Sie all diese Tabellen mithilfe der Methode `Copy`.

6.4.4 Gruppierte Tabellen ermitteln

Möchten Sie herausfinden, welche Tabellen in Ihrer Arbeitsmappe gruppiert sind, dann starten Sie das folgende Makro aus Listing 6.12.

```
Sub GruppierteBlätterErmitteln()
  Dim wksBlatt As Worksheet

  For Each wksBlatt In ActiveWorkbook.Windows(1).SelectedSheets
```

```
    MsgBox wksBlatt.Name
  Next wksBlatt

End Sub
```

Listing 6.12 Gruppierte Tabellen ermitteln

Mithilfe der Eigenschaft `Name` ermitteln Sie die Namen der gruppierten Tabellen. Gruppierte Tabellen fragen Sie über die Eigenschaft `SelectedSheets` ab.

6.5 Tabellenblätter löschen

Wie Sie Tabellenblätter einfügen, wissen Sie jetzt. Aber wie löschen Sie Tabellenblätter? Dafür setzen Sie die Methode `Delete` ein.

6.5.1 Eine Tabelle löschen

Beim Beispiel aus Listing 6.13 wird Tabelle1 in der Arbeitsmappe gelöscht. Hierbei müssen Sie zwischen dem »normalen« Tabellennamen und dem Codenamen der Tabelle unterscheiden.

```
Sub TabellenblattLöschen()
  On Error GoTo fehler
  Sheets("Tabelle1").Delete

  'oder eben über den Codenamen
  'Tabelle1.Delete

  Exit Sub

fehler:
  MsgBox "Es gibt keine Tabelle1 zum Löschen"

End Sub
```

Listing 6.13 Ein benanntes Tabellenblatt löschen

Zu Beginn sorgt die Anweisung `On Error` dafür, dass im Fehlerfall sofort zur Textmarke `fehler` gesprungen wird. Ein Fehler kann z. B. auftreten, wenn die Tabelle gar nicht in der Arbeitsmappe enthalten ist. Danach wird versucht, die Tabelle Tabelle1 über den Einsatz der Methode `Delete` zu löschen. Sollte der Vorgang erfolgreich sein, wird die nächste Zeile abgearbeitet, wenn nicht, wird zur Textmarke `fehler` gesprungen. Die Anweisung `Exit Sub` sorgt dafür, dass nach dem erfolgreichen Löschen des Tabellen-

blattes das Makro sofort beendet, also die Textmarke `fehler` nicht mehr abgearbeitet wird. Die Textmarke `fehler` leitet die Fehlerbehandlung ein. Sie wird nur ausgeführt, wenn z. B. versucht wurde, eine Tabelle zu löschen, die es gar nicht mehr gibt. Als Fehlerreaktion wird eine einfache Meldung auf dem Bildschirm ausgegeben.

Diese `On Error`-Geschichte sollten Sie aber auch nicht übertreiben. Oft werden mir Quellcodes zur Begutachtung vorgelegt, bei denen es davon nur so wimmelt. Einen möglichen Fehler mehrfach zu ignorieren ist keine gute Reaktion auf einen Fehler. Besser wäre eine Funktion, die immer vorher prüft, ob sich eine Tabelle in der Arbeitsmappe befindet. In diesem Fall bräuchten Sie keine Fehlerbehandlung mehr. In meinen Softwareprojekten habe ich immer eine solche Funktion zur Verfügung (siehe Listing 6.14).

```
Function TabDa(strBlatt As String) As Boolean
 Dim wksBlatt As Worksheet

 For Each wksBlatt In ThisWorkbook.Worksheets

  If wksBlatt.Name = strBlatt Then
     TabDa = True
     Exit For
  End If

 Next wksBlatt

End Function
```

Listing 6.14 Funktion, die prüft, ob eine bestimmte Tabelle in der Mappe vorhanden ist

Immer, wenn Sie eine Tabelle löschen oder auf eine Tabelle zugreifen müssen, rufen Sie vorher die Funktion `TabDa` auf und übergeben ihr den Namen der entsprechenden Tabelle. In der Funktion selbst wird eine Schleife des Typs `For Each ... Next` durchlaufen, die den Namen der an die Funktion übergebenen Tabelle mit dem jeweiligen Namen der Tabelle vergleicht, den Sie durch die Schleife ansprechen. Wird die gesuchte Tabelle in der Arbeitsmappe gefunden, dann setzen Sie den Rückgabewert der Funktion auf `True` und verlassen vorzeitig die Schleife, indem Sie die Anweisung `Exit For` einsetzen. Damit springen Sie direkt im Anschluss auch aus der Funktion und landen wieder im aufrufenden Makro aus Listing 6.15.

```
Sub TabelleBenennenMitVorherigerPrüfung()

  If TabDa("Tabelle10") = False Then
    Worksheets.Add.Name = "Tabelle10"
```

```
  Else
    MsgBox "Die Tabelle10 ist bereits in der Mappe!"
  End If

End Sub
```

Listing 6.15 Das Makro ruft eine Funktion auf und verarbeitet ihre Rückmeldung.

6.5.2 Bestimmte Tabellen aus einer Mappe entfernen

Im Beispiel aus Listing 6.16 werden alle Tabellen aus der Arbeitsmappe entfernt, die im Tabellennamen das Kürzel »neu« aufweisen.

```
Sub BestimmteTabellenEntfernen()
  Dim wksBlatt As Worksheet

  Application.DisplayAlerts = False

  For Each wksBlatt In ActiveWorkbook.Worksheets

      If wksBlatt.Name Like "*neu*" Then

          wksBlatt.Delete

      End If

  Next wksBlatt

  Application.DisplayAlerts = True

End Sub
```

Listing 6.16 Alle Tabellen mit einer bestimmten Zeichenfolge im Namen werden gelöscht.

Mithilfe der Eigenschaft `DisplayAlerts` schalten Sie die Excel-Warnmeldungen temporär aus, indem Sie dieser Eigenschaft den Wert `False` zuweisen. Dies ist wichtig, da Sie sonst die Löschung einer jeden Tabelle bestätigen müssten. In einer Schleife der Form `For Each ... Next` arbeiten Sie alle Tabellen der Arbeitsmappe nacheinander ab. Im Innern der Schleife prüfen Sie mit dem Operator `Like`, ob im Tabellennamen der Begriff »neu« vorkommt. Dabei unterscheidet dieser Operator zwischen Groß- und Kleinschreibung. Wird eine Tabelle gefunden, deren Name die Zeichenfolge »neu« enthält, dann wird die Methode `Delete` verwendet, um die Tabelle zu löschen.

Hinweis

Möchten Sie nicht, dass Excel zwischen Groß- und Kleinschreibung unterscheidet, dann tauschen Sie die Bedingung in Listing 6.16 durch die folgende Bedingung aus:

```
If UCase(wksBlatt.Name) Like "*NEU*" Then
```

6.5.3 Tabellen mit gefärbten Registerlaschen entfernen

Zur Identifizierung zu löschender Tabellen könnten Sie auch die Farbe der Tabellenreiter heranziehen. Das Makro aus Listing 6.17 entfernt alle Tabellen ohne Rückfrage aus der Arbeitsmappe, die mit rotem Tabellenreiter formatiert wurden.

```
Sub FarbTabellenEntfernen()
  Dim wksBlatt As Worksheet

  Application.DisplayAlerts = False

  For Each wksBlatt In ActiveWorkbook.Worksheets

      If wksBlatt.Tab.ColorIndex = 3 Then

          wksBlatt.Delete

      End If

  Next wksBlatt

  Application.DisplayAlerts = True

End Sub
```

Listing 6.17 Alle Tabellen mit einem roten Tabellenreiter werden entfernt.

Über das Objekt `Tab` sprechen Sie den Tabellenreiter einer Tabelle an. Mithilfe der Eigenschaft `ColorIndex` lesen Sie die Farbe des Tabellenreiters aus. Ist der Tabellenreiter rot eingefärbt, dann gibt die Eigenschaft die Farbnummer `3` zurück. In diesem Fall wenden Sie die Methode `Delete` an, um die Tabelle zu löschen.

Hinweis

Möchten Sie nicht nur rote, sondern alle farbigen Tabellen aus der Mappe entfernen, dann passen Sie das Löschkriterium wie folgt an:

```
If wksBlatt.Tab.ColorIndex > 0 Then
```

6.5.4 Leere Tabellen aus Arbeitsmappen entfernen

Bei der nächsten Lösung sehen Sie im Makro aus Listing 6.18, wie Sie leere Tabellen aus einer Arbeitsmappe entfernen.

```
Sub LeereTabellenAusMappeEntfernen()
   Dim Blatt As Worksheet
        Application.DisplayAlerts = False
    For Each Blatt In ActiveWorkbook.Worksheets
      If Application.WorksheetFunction.CountA(Blatt.Cells) = 0 Then
        Blatt.Delete
      End If
        Application.DisplayAlerts = True
    Next Blatt

End Sub
```

Listing 6.18 Leere Tabellen aus der Arbeitsmappe entfernen

In einer Schleife des Typs `For Each ... Next` arbeiten Sie sich Tabelle für Tabelle durch die aktive Arbeitsmappe hindurch. Innerhalb der Schleife prüfen Sie mithilfe der Tabellenfunktion ANZAHL2 (engl. `CountA`), ob die Anzahl der Zellen, die einen Eintrag enthalten, null ist. Ist dies der Fall, dann sind in der Tabelle keine Daten vorhanden, und Sie wenden die Methode `Delete` an, um die leere Tabelle zu löschen.

6.6 Tabellenblätter ein- und ausblenden

Wenn Sie bestimmte Tabellenblätter nicht mit einem Passwort schützen, jedoch trotzdem einen gewissen Schutz Ihrer Daten erreichen möchten, können Sie Tabellenblätter auch ausblenden. Das Ein- und Ausblenden von Tabellenblättern erreichen Sie mit der Eigenschaft `Visible`, wie Listing 6.19 zeigt.

```
Sub TabelleAusblenden()

  On Error Resume Next

  Worksheets("Tabelle1").Visible = False

  'oder

  Tabelle1.Visible = xlSheetHidden

End Sub
```

Listing 6.19 Tabellenblatt ausblenden

Nachdem Sie das Makro `TabelleAusblenden` ausgeführt haben, wird die Tabelle in der Arbeitsmappe nicht mehr angezeigt.

Anwenderinnen und Anwender der Excel-Versionen 2007 bis 2024 sowie Microsoft 365 klicken mit der rechten Maustaste auf einen beliebigen Tabellenreiter und wählen den Befehl EINBLENDEN aus dem Kontextmenü aus.

Das Einblenden eines ausgeblendeten Tabellenblattes funktioniert in VBA wie in Listing 6.20.

```
Sub TabelleEinblenden()

  On Error Resume Next

  Sheets("Tabelle1").Visible = True

  'oder

  Tabelle1.Visible = xlSheetVisible

End Sub
```

Listing 6.20 Tabellenblatt wieder einblenden

6.6.1 Tabellenblätter sicher ausblenden

Wenn Sie verhindern möchten, dass die Anwenderin Ihre ausgeblendeten Tabellenblätter über die Benutzeroberfläche wieder einblendet, dann verwenden Sie bei der Eigenschaft `Visible` die Konstante `xlSheetVeryHidden`.

```
Sub TabelleSicherAusblenden()

  On Error Resume Next
  Tabelle1.Visible = xlSheetVeryHidden

End Sub
```

Listing 6.21 Tabelle ausblenden (sichere Methode)

In diesem Fall können Sie Ihre ausgeblendete Tabelle nur mit einem Makro wieder verfügbar machen. Dazu setzen Sie das Makro aus Listing 6.21 ein.

6.6.2 Tabellen je nach Status ein- oder ausblenden

In einer Arbeitsmappe sollen alle eingeblendeten Tabellenblätter ausgeblendet bzw. alle ausgeblendeten Tabellenblätter eingeblendet werden. Das Makro zur Umsetzung dieser Aufgabe sehen Sie in Listing 6.22.

```
Sub TabellenJeNachStatusEinAusblenden()
  Dim wksBlatt As Worksheet

    For Each wksBlatt In ActiveWorkbook.Worksheets

      Select Case wksBlatt.Visible
       Case xlSheetHidden: Blatt.Visible = xlSheetVisible
       Case xlSheetVisible: Blatt.Visible = xlSheetHidden
      End Select

    Next wksBlatt

End Sub
```

Listing 6.22 Tabellenblätter je nach Status ein- oder ausblenden

In einer Schleife der Art `For Each ... Next` überprüfen Sie mithilfe einer `Select Case`-Anweisung, wie der Status der Eigenschaft `Visible` für das jeweilige Tabellenblatt ist. Je nach Status wird der Eigenschaft dann entweder die Konstante `xlSheetVisible` bzw. `xlSheetHidden` zugewiesen.

> **Achtung**
>
> Achten Sie darauf, dass Sie die Anweisung `On Error` in das Makro integrieren. In einer Arbeitsmappe muss immer wenigstens eine Tabelle sichtbar bleiben. Versucht nun das Makro, das letzte Tabellenblatt auszublenden, kommt es zum Fehlerfall, den Sie aber mit dieser Anweisung abfangen.

6.6.3 Alle Tabellenblätter anzeigen

Ausgeblendete Tabellenblätter werden oft vergessen. Diese versteckten Tabellenblätter schlummern dann jahrelang in Arbeitsmappen. Eines Tages erfahren Sie mehr durch Zufall, dass es in der Arbeitsmappe versteckte Tabellenblätter gibt.

Schreiben Sie daher ein Makro wie das in Listing 6.23, das in der aktiven Arbeitsmappe alle Tabellenblätter wieder sichtbar macht.

```
Sub VersteckteBlätterEinblenden()
  Dim wksBlatt As Worksheet

    For Each wksBlatt In ActiveWorkbook.Worksheets
      wksBlatt.Visible = True
    Next wksBlatt

End Sub
```

Listing 6.23 Alle Tabellenblätter einblenden

In einer `For Each ... Next`-Schleife setzen Sie die Eigenschaft `Visible` aller Tabellenblätter auf den Wert `True`.

6.6.4 Alle Tabellen außer der aktiven Tabelle ausblenden

Wenn Sie möchten, können Sie alle Tabellenblätter einer Arbeitsmappe mit Ausnahme des aktiven Tabellenblattes ausblenden, indem Sie das Makro aus Listing 6.24 starten.

```
Sub NurAktivesBlattSichtbar()
  Dim wksBlatt As Worksheet

    For Each wksBlatt In ActiveWorkbook.Worksheets

      If wksBlatt.Name <> ActiveSheet.Name Then
         wksBlatt.Visible = xlSheetHidden
      End If

    Next wksBlatt

End Sub
```

Listing 6.24 Alle Tabellen außer der aktiven Tabelle werden ausgeblendet.

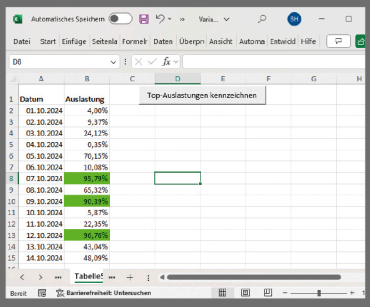
Definieren Sie zuerst eine Objektvariable vom Typ `Worksheet`. Danach greifen Sie in einer Schleife des Typs `For Each ... Next` auf das Auflistungsobjekt `Worksheets` zu, das alle Tabellenblätter der aktiven Arbeitsmappe enthält. Innerhalb der Schleife vergleichen Sie den Namen des aktiven Tabellenblattes mit dem jeweiligen Tabellenblatt aus dem Auflistungsobjekt. Mit der Eigenschaft `Visible`, die Sie auf den Wert `False` oder die Konstante `xlSheetHidden` setzen, blenden Sie alle Tabellenblätter aus der Arbeitsmappe mit Ausnahme des aktiven Tabellenblattes aus.

VBA mit Excel

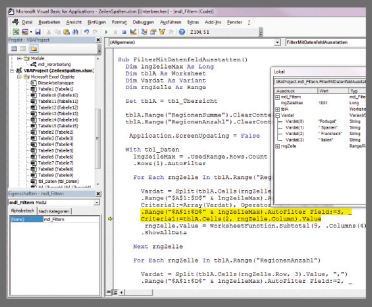
»Über 650 Makros zum sofortigen Einsatz«

Geprüfte VBA-Lösungen für die tägliche Praxis

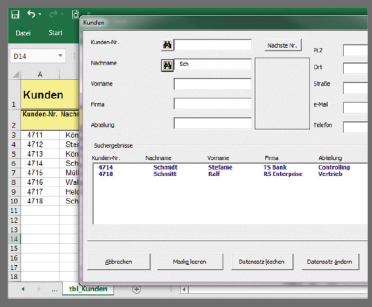
Dieses Standardwerk zeigt Ihnen zuverlässig, wie Sie wiederkehrende Aufgaben und Anwendungen in Excel programmieren – praxisnah und gut verständlich. Neben einer Einführung in Excel-VBA bietet Ihnen dieses Buch eine umfassende Sammlung an VBA-Programmen für alle Aufgaben. Geeignet für Excel 2016 bis 2024 sowie Microsoft 365.



Aufgaben mit VBA vereinfachen



Eigene Makros schreiben



UserForms entwickeln

Einsteigen und Excel automatisieren

Ob Sie Tabellen einrichten, Diagramme erstellen, Pivot-Tabellen aktualisieren oder nur Zeilen löschen wollen: Hier erfahren Sie, wie Sie all dies mit VBA erledigen. Die Aufgaben sind übersichtlich aufgelistet, sodass Sie schnell das Gewünschte finden.

Anwendungen entwickeln

Funktionen, Ereignisse, Dialoge, UserForms, Schaltflächen, Listen, Kontextmenüs und Ribbons: Sie erfahren alles über die Programmierung von Anwendungen für Excel.

Schnell, sicher, praktisch: geprüfte Makros

Hier finden Sie jederzeit die passende Lösung für Ihre beruflichen Excel-Aufgaben. Über 650 sorgfältig getestete Makros werden umfassend beschrieben und stehen Ihnen sofort für verschiedenste Automatisierungsaufgaben zur Verfügung.

Alle im Buch beschriebenen Makros können Sie herunterladen und direkt in Ihre Anwendungen integrieren.

Bernd Held ist Spezialist für Excel, VBA-Programmierung, Access und allgemeine Office-Themen. In Schulungen, Büchern, Fachartikeln, Foren, Lernvideos und seinem VBA-Newsletter gibt er sein Wissen weiter. Er wurde von Microsoft mehrfach als MVP für Excel ausgezeichnet.

Aus dem Inhalt

Rheinwerk Computing

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

€ 49,90 [D] € 51,30 [A]

Office

ISBN 978-3-367-10573-1