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 behandele 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/5413* 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. Darauf-

hin 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 vom Datum 27.12.2021 ausgehen.

englisch	deutsch	Bedeutung	Ergebnis 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	YYYY	Jahresangabe vierstellig	2021

Tabelle 4.1 Die wichtigsten Formatkürzel für das Datum

Die Konvertierung der in Englisch einzugebenden Formatcodes erfolgt ganz 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.

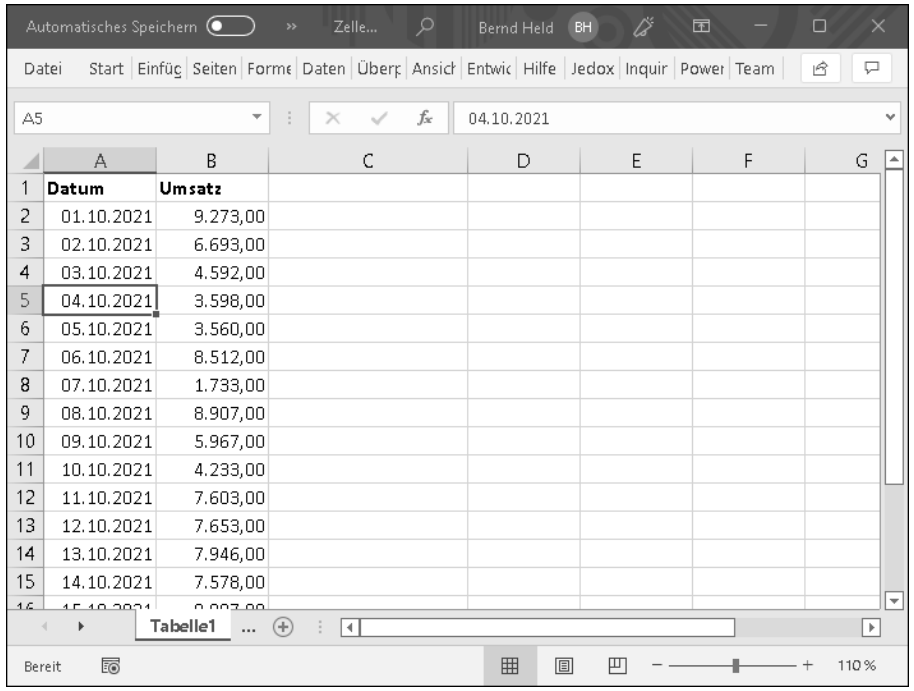


Abbildung 4.1 Die Formatierung wurde vorgenommen.

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.

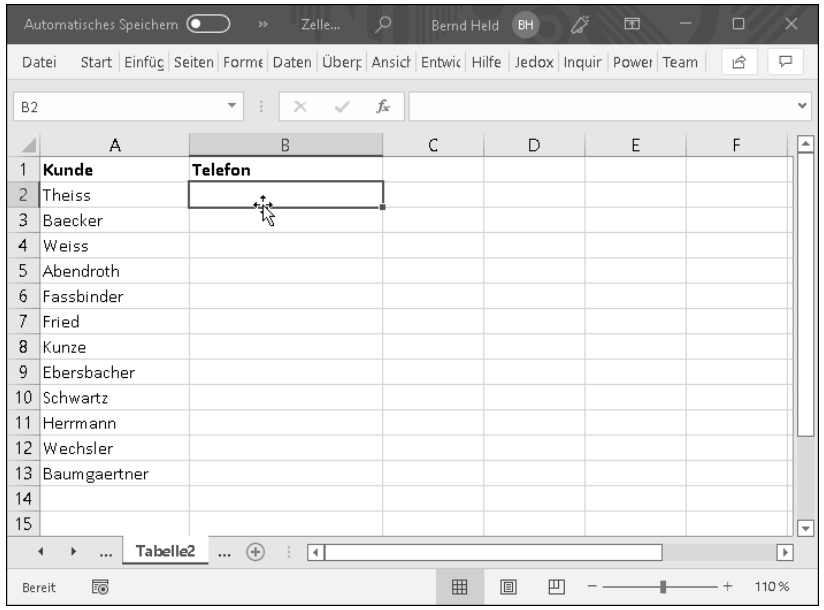


Abbildung 4.2 In Spalte B sollen Telefonnummern eingetragen werden.

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, das heißt, die führenden Nullen werden nicht von Excel 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 für diese 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 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 Bereich eine Spalte versetzt nach rechts 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.

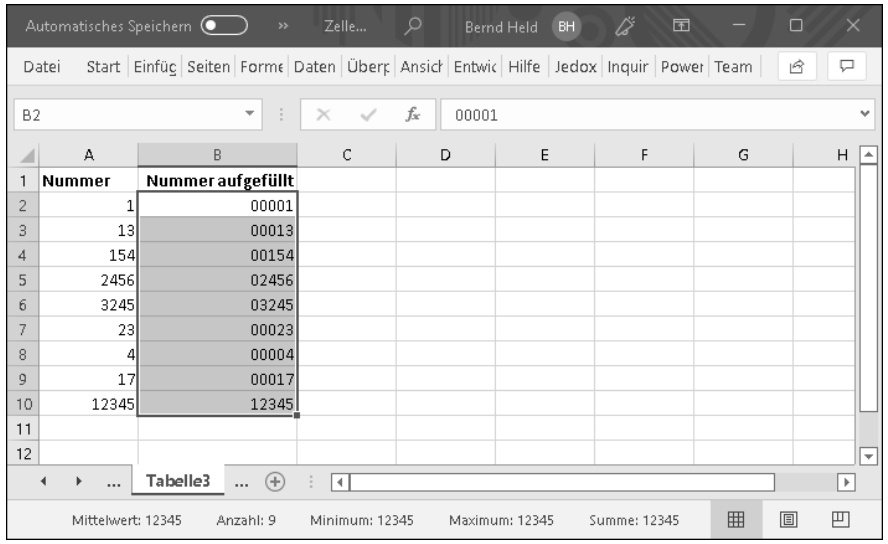


Abbildung 4.3 Die zu kurzen Werte wurden mit vorangestellten Nullen komplettiert.

4.1.5 Einheitliches Datumsformat einstellen

Es ist wie verhext – jeder macht gerade das, was er so will. In Abbildung 4.4 sehen Sie, dass Tabelle4 Datumsangaben in den unterschiedlichsten Formen enthält. Mehrere Datumseingaben werden gar nicht 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

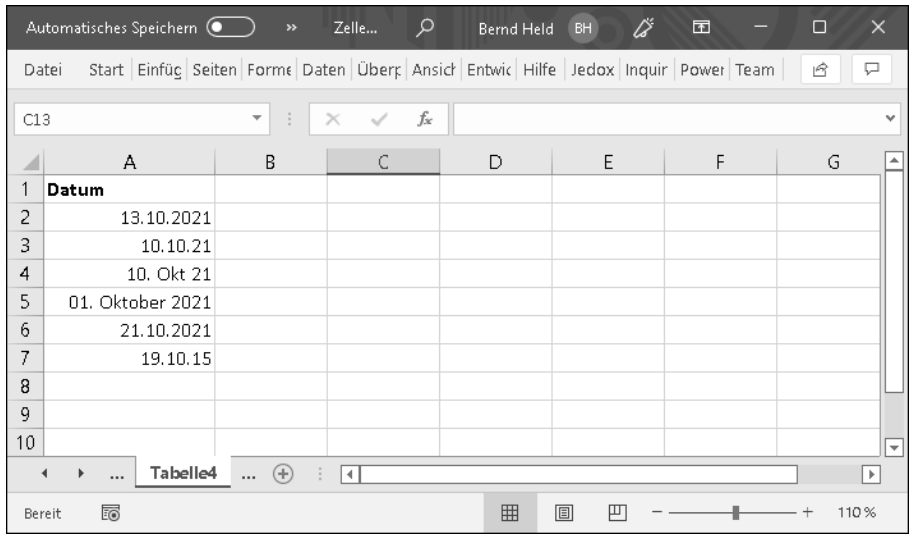


Abbildung 4.4 Die Datumsangaben müssen vereinheitlicht und vor allem von Excel erkannt werden.

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.

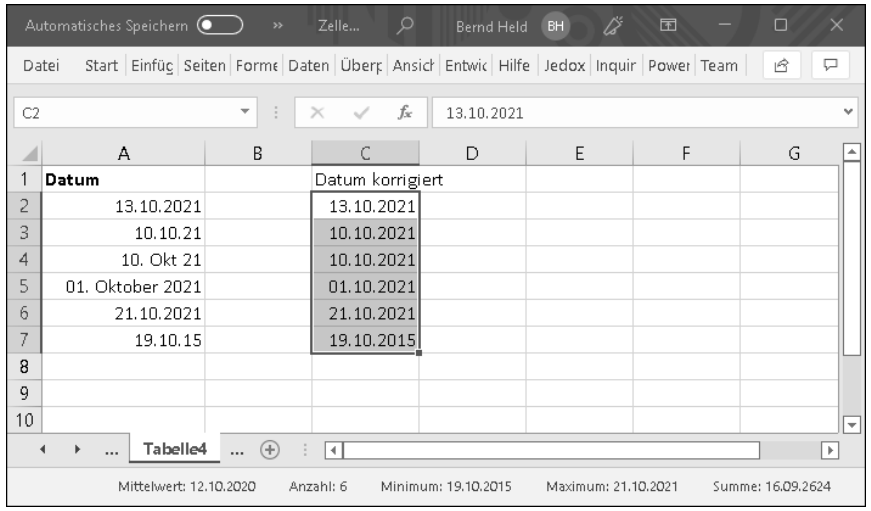


Abbildung 4.5 Alle Datumsangaben wurden konvertiert und einheitlich formatiert.

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, überhaupt 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 sind.

```
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

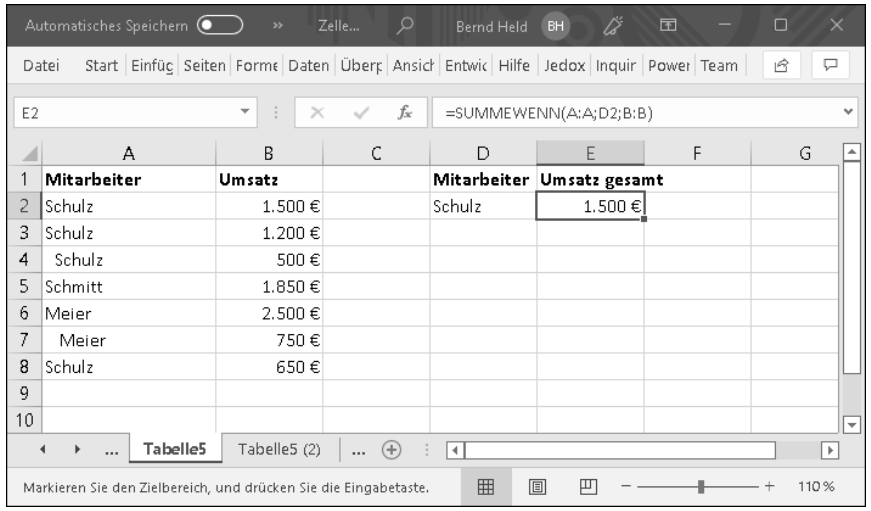


Abbildung 4.6 Dass hier etwas nicht stimmt, ist offensichtlich.

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.

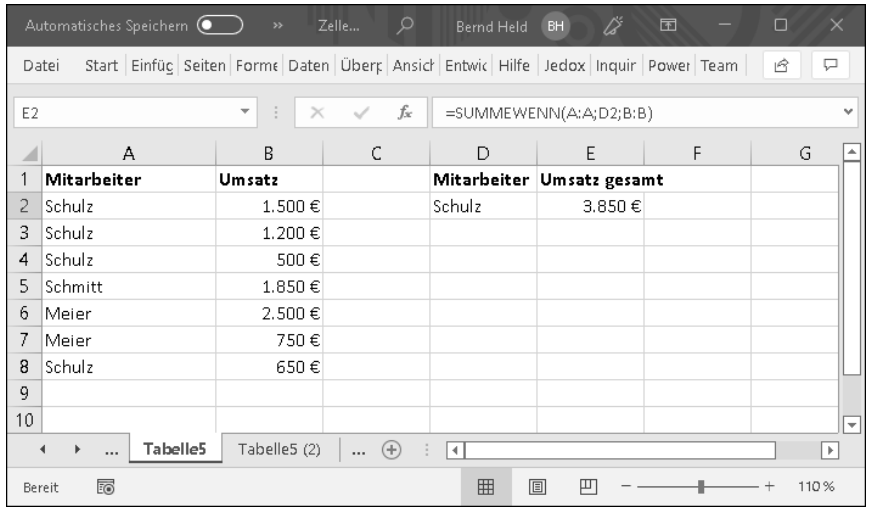


Abbildung 4.7 Alle Leerzeichen sind raus – Excel rechnet richtig.

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.

Ja, genau. Excel erkennt diese importierten Daten nicht. Was nun? Schreiben Sie ein Makro, 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

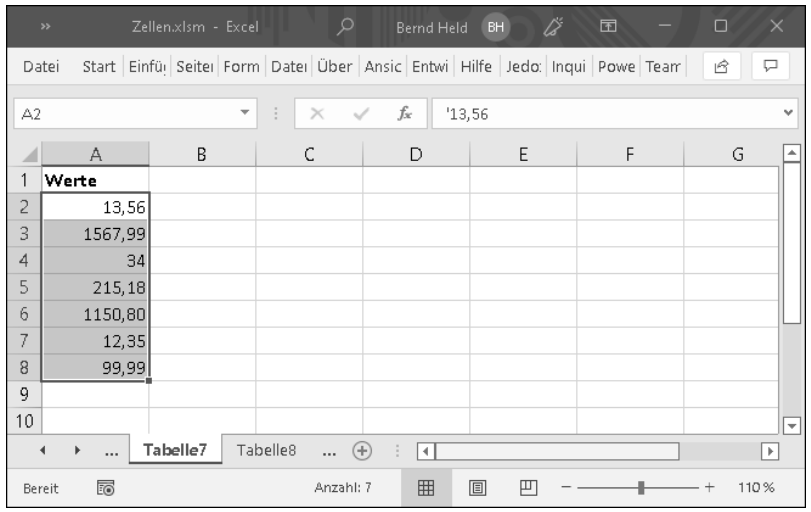


Abbildung 4.8 Nanu? Da wird ja gar nichts mehr berechnet!

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 erken-

nen, indem Sie sie mit dem Wert 1 multiplizieren. Dadurch führt Excel eine Neuberechnung der Zelle durch, und der Wert wird richtig erkannt.

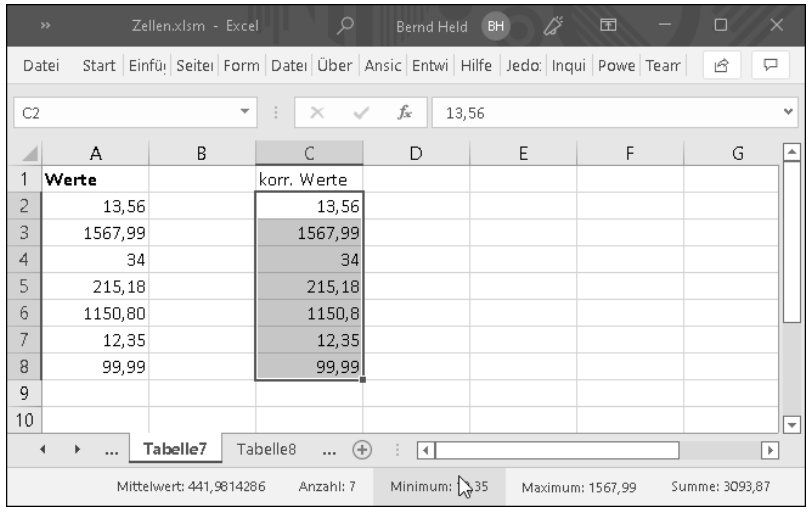


Abbildung 4.9 Excel erkennt die Zahlenwerte jetzt.

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 aus Abbildung 4.10 etwas genauer an.

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.

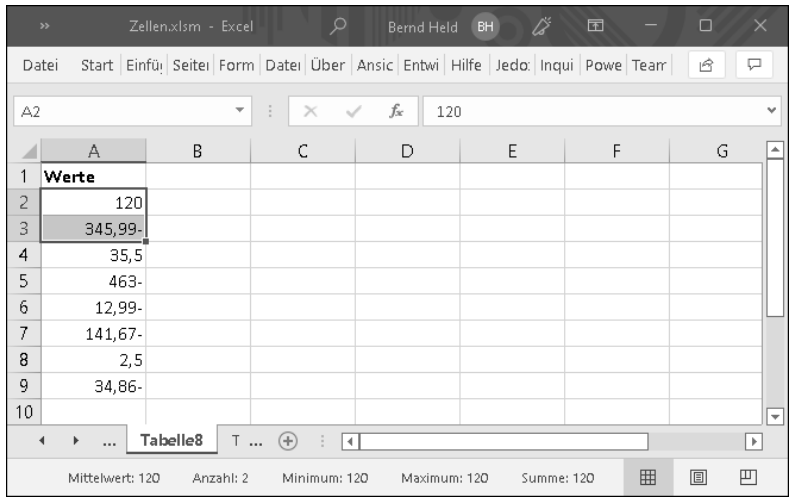


Abbildung 4.10 Das Ergebnis in der Statusleiste stimmt ja in 100 Jahren nicht!

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

```
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.

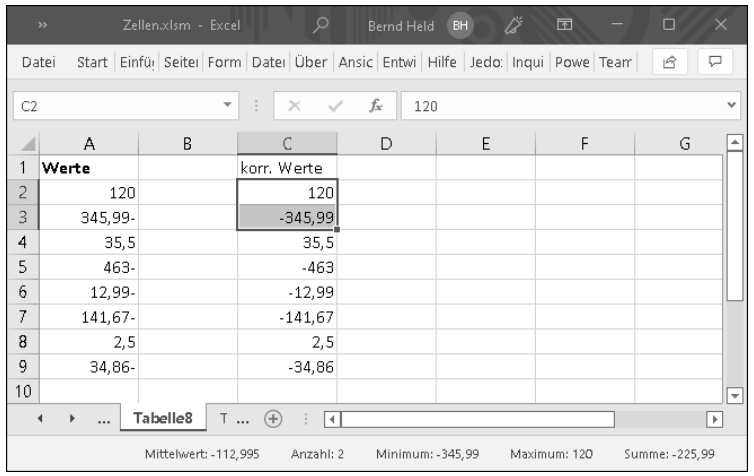


Abbildung 4.11 Excel kann jetzt alle Zellen in Spalte C berechnen.

4.1.9 Daten umschlüsseln

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

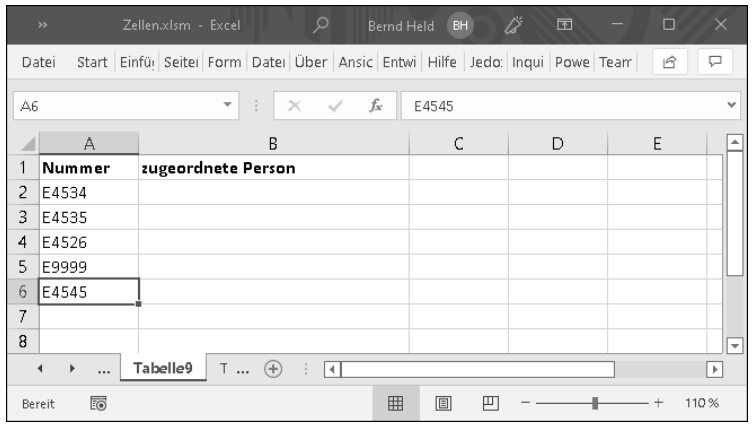


Abbildung 4.12 Welche Nummern werden von welcher Person verarbeitet?

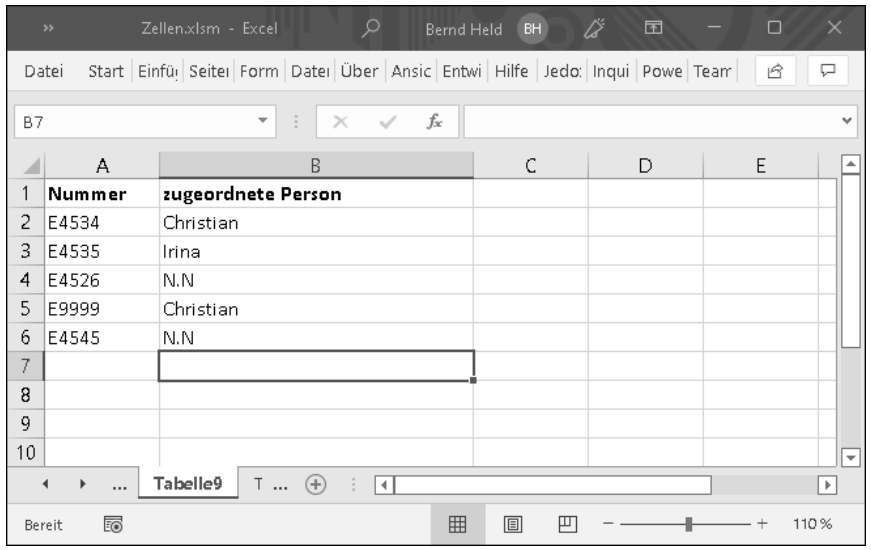
Beantworten Sie diese Frage, indem Sie das Makro aus Listing 4.9 starten.

```
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. 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.

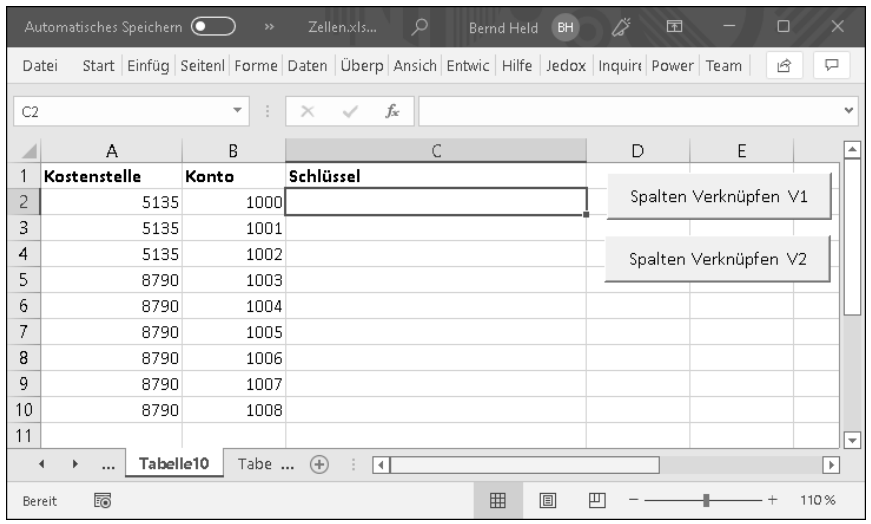


	A	B	C	D	E
1	**Nummer**	**zugeordnete Person**			
2	E4534	Christian			
3	E4535	Irina			
4	E4526	N.N			
5	E9999	Christian			
6	E4545	N.N			
7					
8					
9					
10					

Abbildung 4.13 Die Zuordnung passt und kann weiter angepasst werden.

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. Existieren 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.

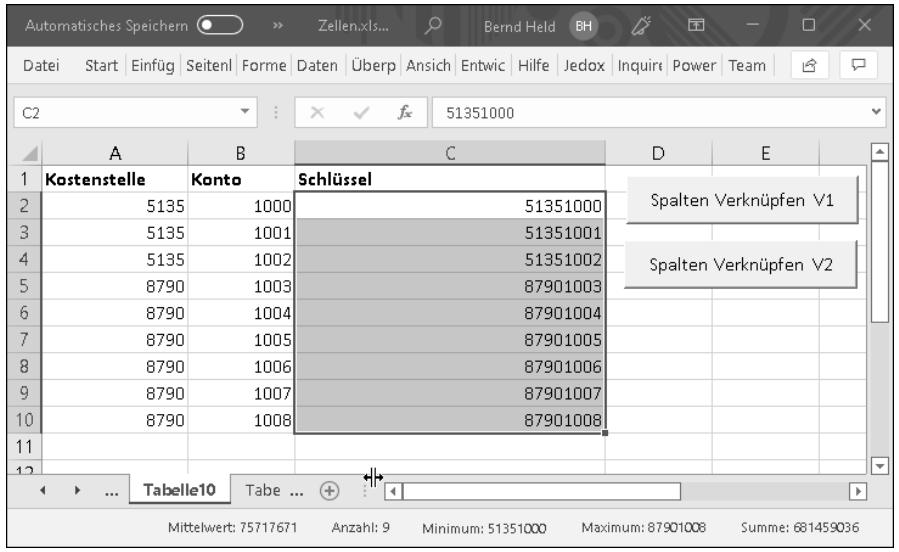


Abbildung 4.15 Schneller geht's nicht – der eindeutige Schlüssel wurde generiert.

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 vorher 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 von verschiedenen 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

Verwenden Sie die `Font`-Eigenschaft, um das `Font`-Objekt zurückzugeben. Über die Eigenschaft `Name` bekommen Sie heraus, um welche Schriftart es sich handelt.

4.2.2 Schriftart ändern

In der nächsten Praxisaufgabe belegen Sie innerhalb einer Markierung alle Zellen mit der Schriftart Courier und dem Schriftgrad 12. Das Makro für diese Aufgabe sehen Sie in Listing 4.13.

```
Sub SchriftenFormatieren()
Dim rngZelle As Range

  For Each rngZelle In Selection

    With Selection.Font
      .Name = "Courier"
      .Size = 12
      .Strikethrough = False
      .Superscript = False
      .Subscript = False
      .OutlineFont = False
      .Shadow = False
      .Underline = xlUnderlineStyleNone
      .ColorIndex = xlAutomatic
    End With

  Next rngZelle

End Sub
```

Listing 4.13 Schriftarten ändern

Das Objekt `Font` hat eine ganze Reihe Eigenschaften, die Sie anwenden können. Für die eben gestellte Aufgabe sind die beiden Eigenschaften `Name` und `Size` relevant. Bei der Angabe der Schriftart ist die korrekte Schreibweise der gewünschten Schriftart wichtig. Bei der Eigenschaft `Size` geben Sie die gewünschte Größe für die Schrift an.

In Tabelle 4.2 finden Sie weitere wichtige Eigenschaften des Objekts `Font`.

Eigenschaft	Beschreibung
`Bold`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text fett formatiert ist.
`Color`	Diese Eigenschaft gibt die Primärfarbe des Objekts wieder. Möglich sind hierbei folgende Konstanten: `vbBlack`, `vbRed`, `vbGreen`, `vbYellow`, `vbBlue`, `vbMagenta`, `vbCyan` und `vbWhite`.
`ColorIndex`	Diese Eigenschaft gibt die Farbe des Rahmens, der Schriftart oder des Innenraums zurück. In Excel existieren 56 Standardfarben.
`FontStyle`	Diese Eigenschaft sagt aus, welcher Schriftschnitt verwendet wird. Möglich sind u. a. Fett- und Kursivschrift.
`Italic`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text kursiv formatiert ist.
`OutLineFont`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text als Konturschriftart formatiert wird.
`Shadow`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text als schattierte Schriftart formatiert wird.
`Strikethrough`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text horizontal durchgestrichen dargestellt wird.
`Subscript`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text tiefgestellt formatiert wird.
`Superscript`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text hochgestellt formatiert wird.
`Underline`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text unterstrichen ist.

Tabelle 4.2 Die wichtigsten Eigenschaften für das Objekt »Font«

Schauen wir uns noch einmal die Eigenschaft `ColorIndex` an. Diese Eigenschaft wird für die Schriftfarbe, den Zelleninnenraum sowie für die Rahmen um Zellen und Bereiche eingesetzt. Die gängigsten Farbnummern habe ich Ihnen in Tabelle 4.3 zusammengetragen.

Farb-Index	Farbe
1	Schwarz
2	Weiß
3	Rot
4	Grün
5	Blau
6	Gelb
7	Magenta
8	Cyan
9	Braun
15	Grau
16	Grau, etwas dunkler
19	Hellgelb
20	Hellblau

Tabelle 4.3 Die wichtigsten Farben mit dazugehörigem Index

Neben der Eigenschaft `ColorIndex` gibt es eine weitere Eigenschaft mit dem Namen `Color`, über die Sie Objekte einfärben können.

```
Sub FarbWahlAlternative()

  ActiveCell.Font.Color = vbRed

End Sub
```

Listing 4.14 Schriftfarbe über eine VB-Konstante einstellen

Weitere Grundfarben und ihre Farbkonstanten entnehmen Sie Tabelle 4.4.

Farbkonstante	Farbe
`vbBlack`	Schwarz
`vbRed`	Rot

Tabelle 4.4 Die Farbkonstanten und ihre Bedeutung

Farbkonstante	Farbe
`vbGreen`	Grün
`vbYellow`	Gelb
`vbBlue`	Blau
`vbMagenta`	Magenta
`vbCyan`	Cyan
`vbWhite`	Weiß

Tabelle 4.4 Die Farbkonstanten und ihre Bedeutung (Forts.)

4.2.3 Zelleninhalte löschen

Stellen Sie sich vor, Sie müssten eine Liste abarbeiten, fehlerhafte Werte manuell kennzeichnen und diese markierten Werte später automatisch löschen. Hier empfiehlt es sich, die fehlerhaften Werte mit der Schriftfarbe Rot einheitlich zu markieren. Anschließend können Sie mit dem Makro aus Listing 4.15 die markierten Werte komfortabel automatisch löschen.

```
Sub InhalteLöschenBeiRoterSchrift()
  Dim rngZelle As Range

  For Each rngZelle In Tabelle11.UsedRange
    If rngZelle.Font.ColorIndex = 3 Then
       rngZelle.ClearContents
    End If
  Next rngZelle

End Sub
```

Listing 4.15 Zelleninhalte löschen, wenn eine Bedingung gegeben ist

Definieren Sie zuerst eine Variable vom Datentyp `Range`. Wenden Sie danach eine `For Each`-Schleife an, die im benutzten Bereich alle Zellen durchsucht und die Zellenformatierung überprüft. Die Eigenschaft `UsedRange` hilft Ihnen dabei, den verwendeten Bereich auf dem Tabellenblatt zu ermitteln. Aus allen Zellen, die mit der Schriftfarbe Rot formatiert sind und die somit dem `ColorIndex` mit der Nummer `3` entsprechen, werden die Werte gelöscht.

4.2.4 Schriftfarbe teilweise ändern

Wenn Sie möchten, können Sie die Schriftfarbe innerhalb einer Zelle sogar mitten in einem Wort ändern. In der folgenden Aufgabenstellung sollen innerhalb einer Markierung alle Texte dahingehend verändert werden, dass die ersten vier Zeichen jeder Zelle mit der Schriftfarbe Rot formatiert werden und die restlichen Zeichen in der Standardfarbe Schwarz verbleiben.

```
Sub SchriftFarbeTeilweiseAnpassen()
 Dim rngZelle As Range

  For Each rngZelle In Tabelle12.UsedRange.Columns(1)
    rngZelle.Characters(1, 4).Font.Color = RGB(255, 0, 0)
  Next rngZelle

End Sub
```

Listing 4.16 Schriftfarbe mitten im Wort ändern

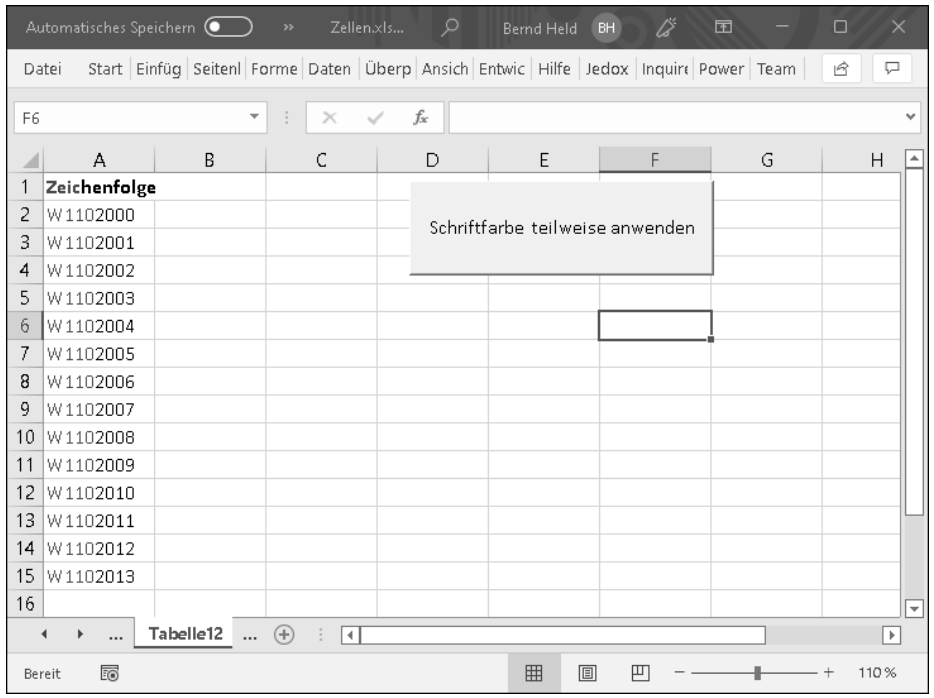


Abbildung 4.16 Die ersten vier Zeichen der Zeichenfolge wurden mit roter Schriftfarbe formatiert.

Definieren Sie im Makro aus Listing 4.16 als Erstes eine Variable vom Datentyp `Range`. Danach wenden Sie in einer Schleife der Form `For Each ... Next` die Eigenschaft

`Characters` an, um die Zellentexte in einzelne Buchstaben zu zerlegen. Der Eigenschaft `Characters` müssen Sie einen Startwert sowie die Anzahl der Zeichen mitteilen, für die die Formatierung gelten soll. Anschließend wird der jeweiligen Zelle die Farbe Rot für die ersten vier Zeichen mithilfe der Funktion `RGB` zugewiesen.

Im Makro aus Listing 4.16 wurde die `RGB`-Funktion verwendet, um die Schrift zu färben. Diese Funktion eignet sich hervorragend, wenn Sie beispielsweise aufgrund von Firmenvorgaben eine bestimmte Farbe benötigen, die in den 56 Standardfarben nicht enthalten ist.

Die Funktion `RGB` hat drei Argumente: Red – Green – Blue. In jedem dieser Argumente können Sie eine Zahl von 0 bis 255 eingeben. Diese Mischung ergibt die gewünschte Farbe. Die gängigsten Mischungen finden Sie in Tabelle 4.5.

Farbe	Rotwert	Grünwert	Blauwert
Schwarz	0	0	0
Blau	0	0	255
Grün	0	255	0
Cyan	0	255	255
Rot	255	0	0
Magenta	255	0	255
Gelb	255	255	0
Weiß	255	255	255

Tabelle 4.5 Farben zusammenmischen über die Funktion »RGB«

4.2.5 Automatisch runden und formatieren

Bei der folgenden Aufgabenstellung soll in Tabelle13 bei Erfassung einer Zahl mit Nachkommastellen die Eingabe sofort gewandelt werden. Dabei soll die Zahl auf eine Stelle hinter dem Komma gerundet werden. Des Weiteren soll geprüft werden, ob der erfasste Betrag größer als 100 ist. Wenn ja, dann soll dieser Wert mit dem Schriftschnitt »fett« formatiert werden.

Dazu stellen Sie ein Tabellenereignis ein, über das Sie die Eingaben abfangen. Gehen Sie wie folgt vor:

1. Klicken Sie in der Excel-Oberfläche den Tabellenreiter von Tabelle13 mit der rechten Maustaste an, und wählen Sie den Befehl Code anzeigen aus dem Kontextmenü.

2. In der Entwicklungsumgebung stellen Sie über die beiden Dropdowns oberhalb des Codefensters das Ereignis `Worksheet_Change` ein.
3. Ergänzen Sie das noch leere Ereignis wie folgt:

```
Sub Worksheet_Change(ByVal Target As Range)
 Dim rngZelle As Range

  If Target.Column <> 1 Then Exit Sub

  For Each rngZelle In Target.Cells

    If IsNumeric(rngZelle.Value) = True Then

        Application.EnableEvents = False
        rngZelle.Value = Application.Round(rngZelle.Value, 1)
        Application.EnableEvents = True

        If rngZelle.Value > 100 Then
            rngZelle.Font.Bold = True
        Else
            rngZelle.Font.Bold = False
        End If

     End If

    Next rngZelle

End Sub
```

Listing 4.17 Automatische Formatierung und Runden direkt nach der Eingabe

Prüfen Sie zunächst im Ereignismakro aus Listing 4.17, ob die Eingabe überhaupt in der ersten Spalte stattgefunden hat. Wenn ja, dann liefert die Eigenschaft `Column` den Wert `1`. Findet eine Eingabe außerhalb von Spalte A statt, dann greift die `Exit Sub`-Anweisung, und das Ereignis wird sofort beendet.

Im anderen Fall setzen Sie eine Schleife des Typs `For Each ... Next` auf, damit auch die Funktionalität des Ausfüllkästchens, wenn Sie eine Zahl nach unten ziehen, um weitere Zellen zu füllen, gewährleistet ist. Prüfen Sie mithilfe der Funktion `IsNumeric`, ob die Eingabe numerisch ist. Ist dies der Fall, dann kann die Eingabe auf eine Stelle hinter dem Komma gerundet werden. Dazu kommt die Funktion `Round` zum Einsatz. Dieser Funktion wird automatisch der gerade erfasste Wert übergeben und im zweiten Argument mit der `1` die Rundung auf eine Stelle hinter dem Komma festgelegt. Da der

Wert der Zelle gerundet neu geschrieben werden muss, müssen Sie davor die Ereignissteuerung komplett und kurzfristig lahmlegen, um eine Endlosschleife zu verhindern. Das Ereignis würde sich ansonsten immer wieder erneut aufrufen, wenn Sie den gerundeten Wert in die Zelle schreiben lassen. Dazu weisen Sie der Eigenschaft `EnableEvents` den Wert `False` zu. Nach dem Schreiben des gerundeten Wertes müssen Sie die Ereignissteuerung wieder einschalten, indem Sie der Eigenschaft `EnableEvents` den Wert `True` zuweisen.

Die zweite Bedingung – für den Fall, dass der erfasste Wert größer als 100 ist – erfassen Sie gleich danach. Eine Formatierung von Zellen löst das Ereignis `Worksheet_Change` nicht aus. Ergibt die Prüfung also, dass der Wert größer als die Vorgabe ist, dann formatieren Sie die Zelle mit dem Schriftschnitt »fett«. Dabei wird der Eigenschaft `Bold` der Wert `True` zugewiesen.

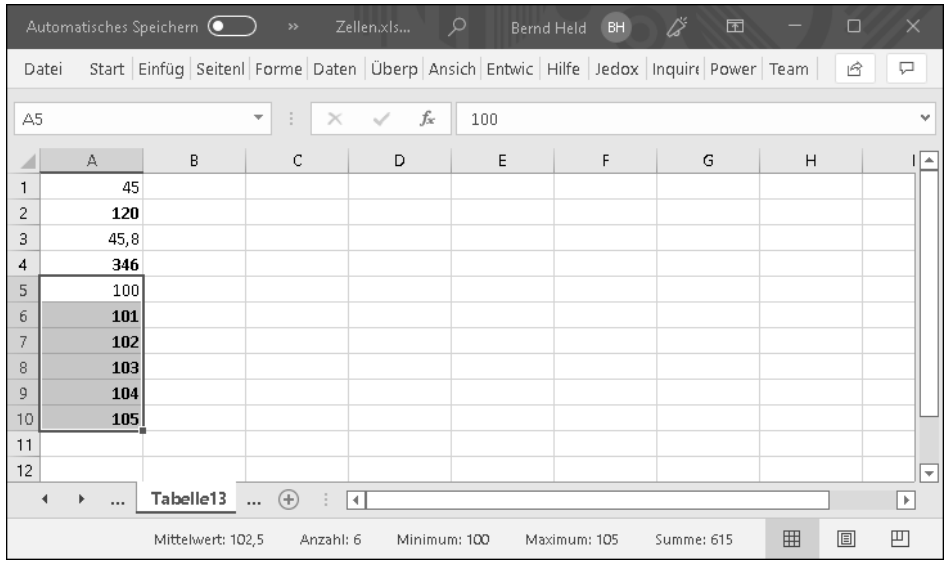


Abbildung 4.17 Alle Eingaben werden gerundet und entsprechend der Vorgabe formatiert.

> **Zur Info**
>
> Mehr zum Thema Ereignisprogrammierung können Sie in Kapitel 9, »Ereignisse programmieren«, nachlesen.

4.2.6 Zwei Bereiche miteinander vergleichen

Bei der folgenden Aufgabenstellung liegen zwei gleich große Bereiche auf den Blättern Tabelle14 und Tabelle15 vor, wie Abbildung 4.18 zeigt.

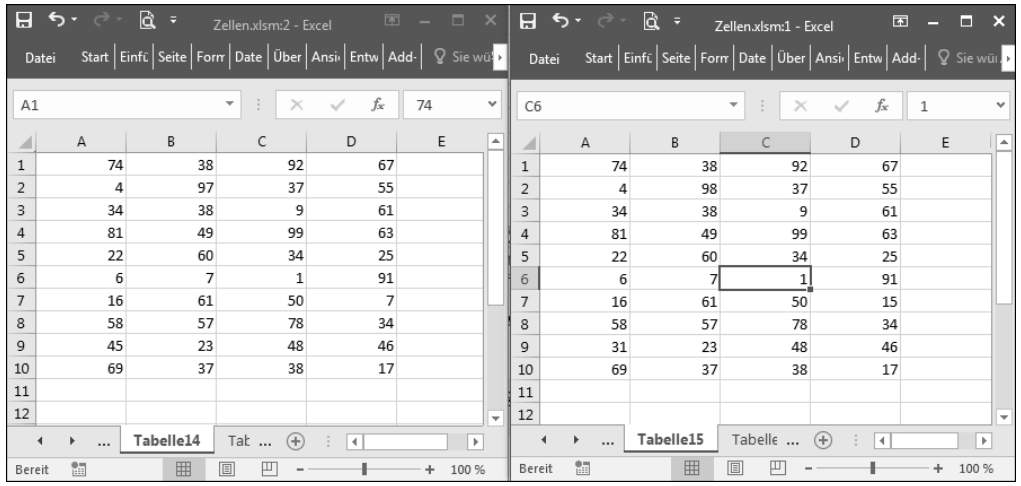


Abbildung 4.18 In welchen Zellen haben sich Werte geändert?

Beantworten Sie diese Frage, indem Sie das Makro aus Listing 4.18 starten.

```
Sub BereicheVergleichen()
  Dim rngZelle As Range
  Dim rngBereich As Range
  Dim wksQuelle As Worksheet
  Dim wksZiel As Worksheet

  Set wksQuelle = Tabelle14
  Set wksZiel = Tabelle15

  With wksQuelle

  Set rngBereich = .Range("A1:D10")

   For Each rngZelle In rngBereich

     If rngZelle.Value <> wksZiel.Range(rngZelle.Address).Value Then

         rngZelle.Interior.ColorIndex = 3
         wksZiel.Range(rngZelle.Address).Interior.ColorIndex = 3

     End If

   Next rngZelle
```



```
  End With

End Sub
```

Listing 4.18 Zwei Bereiche miteinander vergleichen und Unterschiede kennzeichnen

Deklarieren Sie zu Beginn des Makros aus Listing 4.18 zwei Objektvariablen `rngZelle` und `rngBereich` vom Typ `Range`. Diese Variablen benötigen Sie später, um Zelle für Zelle durch die Bereiche zu gehen. Des Weiteren benötigen Sie zwei Objektvariablen `wksQuelle` und `wksZiel` vom Typ `Worksheet`. Geben Sie danach an, mit welchen Tabellen Sie arbeiten möchten. Verwenden Sie dazu die Anweisung `Set`. Danach können Sie die beiden Tabellen über den kürzeren Variablennamen ansprechen.

In einer Schleife der Art `For Each ... Next` arbeiten Sie sich zunächst im Bereich der Tabelle `wksQuelle` durch die einzelnen Zellen. Innerhalb der Schleife vergleichen Sie jeweils die entsprechenden Zellen in beiden Bereichen. Unterscheiden diese sich, dann färben Sie sowohl in der Quelle als auch im Ziel die beiden entsprechenden Zellen. Dazu setzen Sie die Eigenschaft `ColorIndex` ein, die Sie auf das Objekt `Interior` der Zellen anwenden.

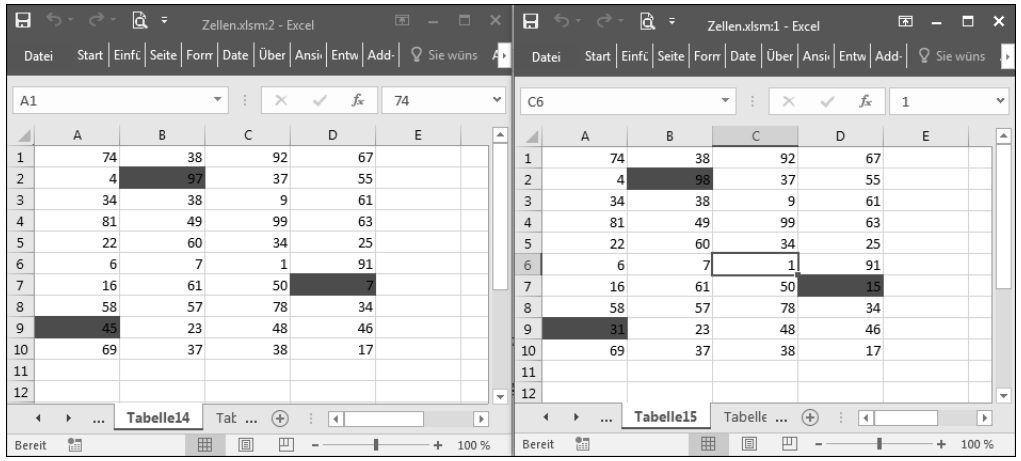


Abbildung 4.19 Die Unterschiede wurden eingefärbt.

4.2.7 Einen Bereich »mustern«

Neben der kompletten Einfärbung einer Zelle haben Sie auch die Möglichkeit, einen Bereich mit einem Muster auszustatten. Schauen Sie sich dazu einmal die Makros aus Listing 4.19 und Listing 4.20 an.

```
Sub ZellenMitKreuzmusterBelegen()

 With Tabelle16
```

```
  .Range(.Cells(1, 1), .Cells(10, 4)).Interior.Pattern = xlPatternCrissCross
 End With

End Sub
```

Listing 4.19 Einen Bereich mit einem Muster belegen

Mithilfe der Eigenschaft `Pattern` definieren Sie ein Muster für den Innenbereich der Zelle. Die Eigenschaft wird mittels einer Konstanten, in diesem Fall `xlPatternCrissCross`, festgelegt. Weitere mögliche Konstanten entnehmen Sie Tabelle 4.6.

Name	Wert	Beschreibung
xlPatternAutomatic	-4105	Das Muster wird von Excel gesteuert.
xlPatternChecker	9	Schachbrettmuster
xlPatternCrissCross	16	Schraffurlinien
xlPatternDown	-4121	von links oben nach rechts unten verlaufende dunkle diagonale Linien
xlPatternGray16	17	16 % Grau
xlPatternGray25	-4124	25 % Grau
xlPatternGray50	-4125	50 % Grau
xlPatternGray75	-4126	75 % Grau
xlPatternGray8	18	8 % Grau
xlPatternGrid	15	Raster
xlPatternHorizontal	-4128	dunkle horizontale Linien
xlPatternLightDown	13	von links oben nach rechts unten verlaufende helle diagonale Linien
xlPatternLightHorizontal	11	helle horizontale Linien
xlPatternLightUp	14	von links unten nach rechts oben verlaufende helle diagonale Linien
xlPatternLightVertical	12	helle vertikale Striche
xlPatternNone	-4142	kein Muster
xlPatternSemiGray75	10	75 % dunkles Moiré

Tabelle 4.6 Die verfügbaren Konstanten der Eigenschaft »Pattern«

Name	Wert	Beschreibung
xlPatternSolid	1	Volltonfarbe
xlPatternUp	-4162	von links unten nach rechts oben verlaufende dunkle diagonale Linien
xlPatternVertical	-4166	dunkle vertikale Striche

Tabelle 4.6 Die verfügbaren Konstanten der Eigenschaft »Pattern« (Forts.)

Die Eigenschaft `Pattern` kann entweder über die Konstante oder den Zahlenwert bestimmt werden.

Um ein Zellenmuster wieder zu entfernen, starten Sie beispielsweise das Makro aus Listing 4.20.

```
Sub ZellenMusterEntfernen()

  With Tabelle16
    .Range(.Cells(1, 1), .Cells(10, 4)).Interior.Pattern = xlNone
  End With

End Sub
```

Listing 4.20 Das eingestellte Zellenmuster entfernen

Ein eingestelltes Muster entfernen Sie, indem Sie der Eigenschaft `Pattern` die Konstante `xlNone` zuweisen.

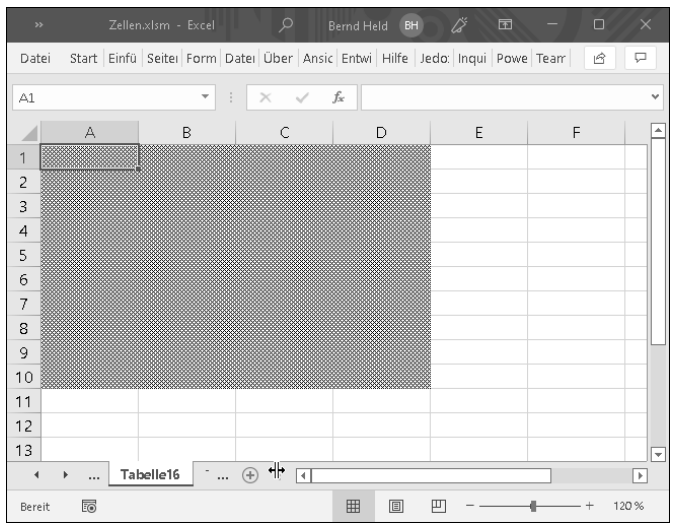


Abbildung 4.20 Das Kreuzmuster wurde im Bereich angewendet.

4.2.8 Einen Bereich einrahmen

Beim folgenden Beispiel wird in Tabelle17 ein Bereich eingerahmt. Dabei sollen die Koordinaten für die Rahmung aus zwei Zellen bezogen werden. Sehen Sie sich dazu das Makro aus Listing 4.21 an.

```
Sub RahmenSetzen()
  Dim rngBereich As Range
  Dim lngZeile As Long
  Dim lngSpalte As Long

  With Tabelle17

   lngZeile = .Range("G2").Value
   lngSpalte = .Range("G1").Value

   Set rngBereich = .Range(.Cells(1, 1), .Cells(lngZeile, lngSpalte))

    With rngBereich.Borders
     .ColorIndex = 10
     .LineStyle = xlDouble
    End With

  End With

End Sub
```

Listing 4.21 Einen Rahmen auf Basis von zwei steuernden Zellen setzen

Zu Beginn des Makros aus Listing 4.21 legen Sie eine Objektvariable mit dem Namen `rangBereich` mit dem Datentyp `Range` an. Danach benötigen Sie noch zwei weitere Variablen vom Typ `Long`, in denen Sie später festhalten, auf welchen Zellenbereich der Rahmen angewendet werden soll. Im nächsten Schritt weisen Sie diesen beiden Variablen die gewünschte Anzahl von Zeilen und Spalten, die in den Zellen G2 und G1 stehen, zu. Danach setzen Sie die Variable `rngBereich` über die Anweisung `Set`. Dabei wird mithilfe der Eigenschaft `Range` ein flexibler Bereich definiert, der in Zelle A1 beginnt und eben flexibel endet. Die Eigenschaft `Cells` steckt dabei die Ecken ab.

Über die `Borders`-Auflistung können Sie jetzt auf alle Rahmen zugreifen. Über die Eigenschaft `ColorIndex` färben Sie den Rahmen. Mithilfe der Eigenschaft `LineStyle` können Sie die Art der Rahmenlinie über das Zuweisen einer Konstanten bestimmen. Im vorliegenden Beispiel habe ich die Konstante `xlDouble` verwendet, um eine doppelte Rahmenlinie einzufügen. Entnehmen Sie Tabelle 4.7 weitere Konstanten für die Gestaltung der Rahmenlinie.

Konstante	Beschreibung
xlContinuous	durchgezogene Linie
xlDash	gestrichelte Linie
xlDashDot	Linie aus Strichen und Punkten
xlDashDotDot	Linie aus Strich – Punkt – Punkt
xlDot	gepunktete Linie
xlDouble	Linie doppelt
xlSlantDashDot	Linie aus Wellenzeichen und Punkt
xlLineStyleNone	keine Linie

Tabelle 4.7 Die verschiedenen Möglichkeiten bei der Rahmenliniengestaltung

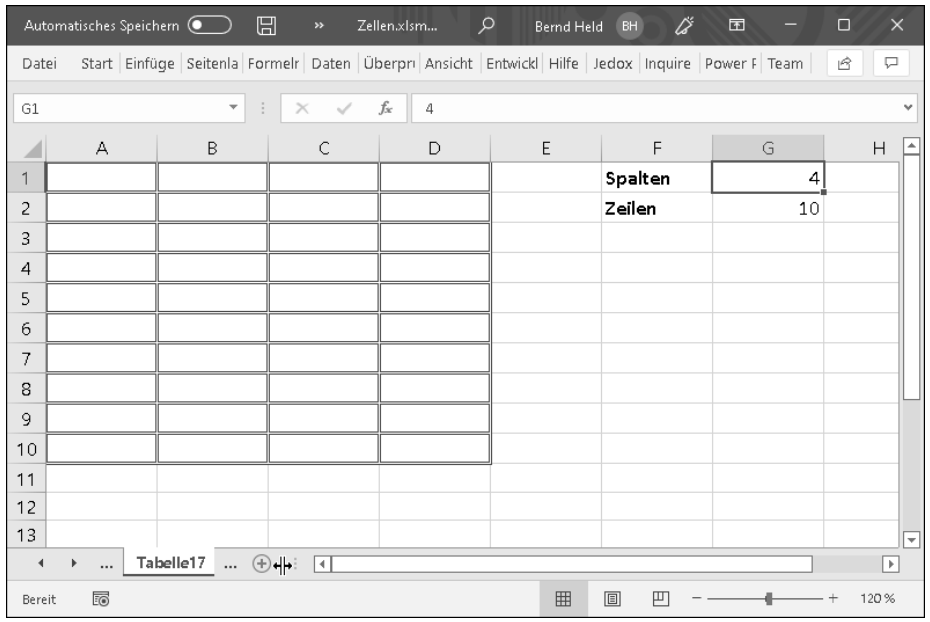


Abbildung 4.21 Ein doppelter Zellenrahmen wurde angelegt.

4.2.9 Einen Bereich umrahmen

Beim folgenden Beispiel soll ein Bereich in Tabelle20 mit einem Außenrahmen versehen werden. Im Innern des Bereichs sollen horizontale und vertikale Linien einge-

fügt werden. Des Weiteren sollen in dem Bereich automatisiert Zufallszahlen eingefügt werden. Alle diese Aufgaben erledigt das Makro aus Listing 4.22.

```
Sub BereichEinrahmen()
 Dim rngBereich As Range

  Set rngBereich = Tabelle20.Range("B2:G10")

  With rngBereich

   .Borders(xlInsideHorizontal).LineStyle = xlContinuous
   .Borders(xlInsideVertical).LineStyle = xlContinuous

   .BorderAround Weight:=xlThick, ColorIndex:=3
   .Formula = "=INT(RAND()*100)"

  End With

End Sub
```

Listing 4.22 Einen Bereich umrahmen und Werte eintragen

Deklarieren Sie zu Beginn des Makros aus Listing 4.22 eine Objektvariable mit dem Namen `rngBereich` vom Datentyp `Range`. Danach geben Sie über die Anweisung `Set` bekannt, auf welcher Tabelle dieser Bereich liegt und wie groß der Bereich sein soll.

Danach sprechen Sie das Objekt `Borders` an. Sprechen Sie in der `Borders`-Auflistung, die standardmäßig alle Rahmen enthält, nur die horizontalen und vertikalen Linien über die Konstanten `xlInsideHorizontal` und `xlInsideVertical` an. Über die Eigenschaft `LineStyle` entscheiden Sie, welche Linienart Sie einsetzen möchten. Die verfügbaren Linienarten listet Tabelle 4.7 auf.

In Tabelle 4.8 finden Sie die verfügbaren Rahmen der `Borders`-Auflistung.

Konstante	Index-Wert	Beschreibung
xlDiagonalDown	5	Rahmen von der linken oberen Ecke zur rechten unteren Ecke jeder Zelle im Bereich
xlDiagonalUp	6	Rahmen von der linken unteren Ecke zur rechten oberen Ecke jeder Zelle im Bereich
xlEdgeBottom	9	Rahmen am unteren Rand des Bereichs
xlEdgeLeft	7	Rahmen am linken Rand des Bereichs

Tabelle 4.8 Die verfügbaren Rahmen der »Borders«-Auflistung

Konstante	Index-Wert	Beschreibung
xlEdgeRight	10	Rahmen am rechten Rand des Bereichs
xlEdgeTop	8	Rahmen am oberen Rand des Bereichs
xlInsideHorizontal	12	horizontale Rahmenlinien für alle Zellen im Bereich, mit Ausnahme von Rahmen am Außenrand des Bereichs
xlInsideVertical	11	vertikale Rahmenlinien für alle Zellen im Bereich, mit Ausnahme von Rahmen am Außenrand des Bereichs

Tabelle 4.8 Die verfügbaren Rahmen der »Borders«-Auflistung (Forts.)

Mithilfe der Methode `BorderAround` fügen Sie einen den Bereich umrahmenden Rahmen ein. Dabei stehen Ihnen unter anderem die Parameter `Weight` und `ColorIndex` zur Verfügung.

Über die Eigenschaft `Formula` stellen Sie die Formel zur Erstellung von Zufallszahlen ein. Dabei erstellt die Funktion `Rand` einen Wert zwischen 0 und 1 mit Nachkommastellen. Dieser Wert wird mit 100 multipliziert. Danach wird aus der berechneten Zahl mit der Funktion `Int` ein ganzzahliger Wert gebildet.

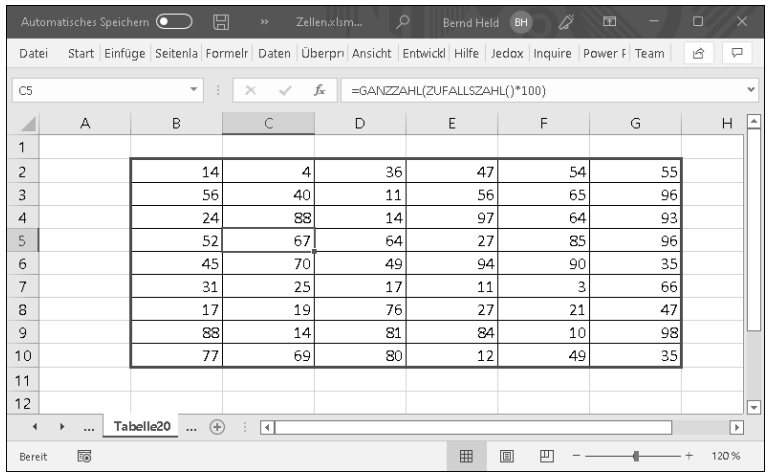


Abbildung 4.22 Der Bereich wurde umrahmt und mit Zufallszahlen ausgestattet.

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 Download-Paket, das Sie von *www.rheinwerk-verlag.de/5413* 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. 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, 12 Monatstabellen anlegen und benennen

Bei der nächsten Aufgabe – siehe Listing 6.4 – soll eine neue Mappe mit 12 Tabellen erstellt werden. Diese Tabellen sollen danach 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 12 Tabellen anlegen und diese nach Monatsnamen benennen

Ermitteln Sie zunächst über die Eigenschaft `SheetsInNewWorkbook`, welche Applikationseinstellung bezüglich der Anzahl Tabellen festgelegt ist, wenn eine neue Mappe angelegt ist. Speichern Sie diesen Wert in der Variablen `intAnz` zwischen. Jetzt ändern Sie den Wert dieser Eigenschaft in 12 Tabellen. Durch die Methode `Add`, die auf das Auflistungsobjekt `Workbooks` angewendet wird, wird nun eine neue Arbeitsmappe mit 12 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 12 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.

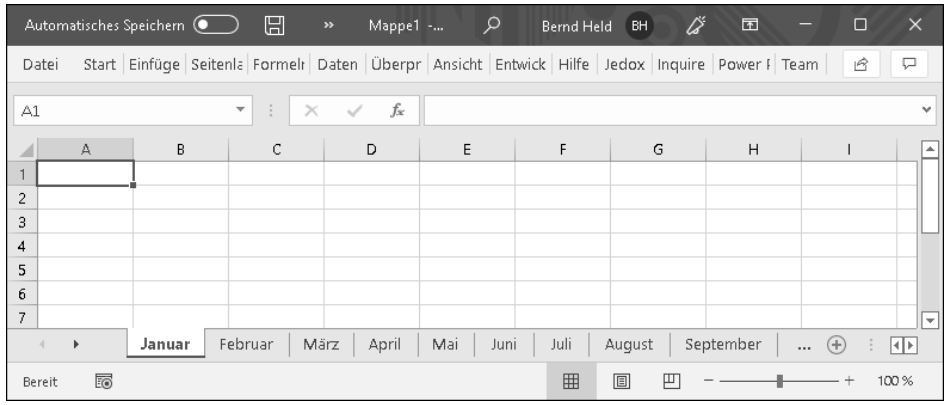


Abbildung 6.1 Die Monatstabellen wurden automatisch angelegt.

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.

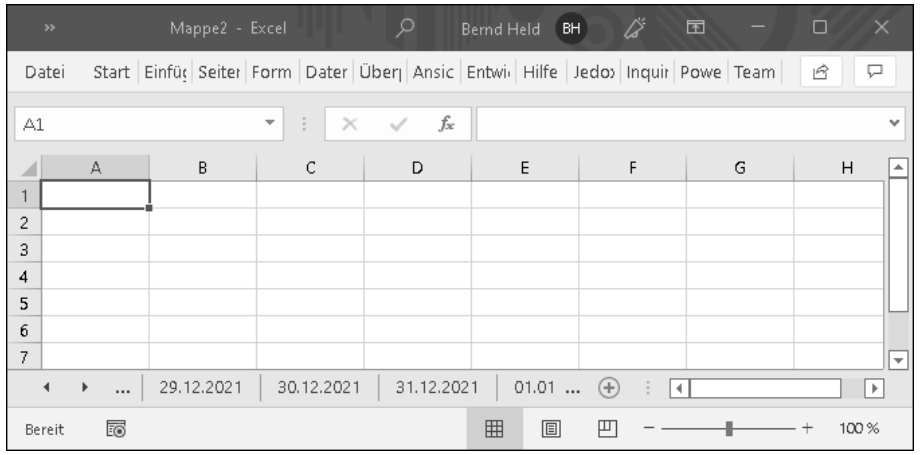


Abbildung 6.2 Eine neue Mappe für die nächsten 14 Tage steht bereit.

6.2.3 Tabelle einfügen und gleichzeitig benennen

Selbstverständlich können Sie das Einfügen von neuen Tabellenblättern und deren Benennung auch in einem Aufwasch erledigen:

```
Sub TabelleEinfügenUndBenennen()

  Worksheets.Add.Name = "Tabelle10"

End Sub
```

Listing 6.6 Neue Tabelle einfügen und benennen in einem Schritt

Allerdings ist hierbei zu beachten, dass Sie sich sicher sein müssen, ob 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. 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 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:

```
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 eine Tabelle sich in der Arbeitsmappe befindet. In diesem Fall bräuchten Sie keine Fehlerbehandlung mehr. In meinen Software-Projekten 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 existiert

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, dass Excel nicht 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, die mit rotem Tabellenreiter formatiert wurden, ohne Rückfrage aus der Arbeitsmappe.

```
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 (englisch `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`.

```
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 2021 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 Tabellenblatts funktioniert in VBA wie folgt:

```
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, 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.

6.7 Tabellenblätter schützen

Haben Sie wichtige Daten auf Ihrem Tabellenblatt erfasst und möchten Sie sie vor Veränderung durch andere schützen, so können Sie Ihre Tabelle mit einem Passwort belegen:


```
Sub BlattschutzEinschalten()

  ActiveSheet.Protect Password:="test", _
  DrawingObjects:=True, Contents:=True, Scenarios:=True

End Sub
```

Listing 6.25 Tabellen schützen

Die Syntax

Es lohnt sich, die Syntax der Methode `Protect` einmal näher zu betrachten. Dabei beschränken wir uns zunächst auf die Argumente, die in allen Excel-Versionen von 97 bis 2021 sowie Microsoft 365 verfügbar sind:

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

Die Argumente der Methode »Protect«

- Im Argument `Passwort` geben Sie eine Zeichenfolge an, die das groß-/kleinschreibungsabhängige Kennwort für das Blatt oder die Arbeitsmappe festlegt. Wenn Sie dieses Argument weglassen, kann der Schutz des Blattes oder der Arbeitsmappe ohne Angabe eines Kennworts aufgehoben werden. Weisen Sie dagegen ein Kennwort zu, muss das Kennwort angegeben werden, um den Schutz des Blattes oder der Arbeitsmappe aufzuheben.
- Mit dem Argument `DrawingObjects` legen Sie fest, ob Sie zusätzlich zu Ihren Zellen auch Formen – wie z. B. Blockpfeile, Sterne oder Banner – schützen möchten. Diese Formen werden standardmäßig jedoch nicht geschützt. Wenn Sie Formen schützen möchten, setzen Sie das Argument auf den Wert `True`.
- Bei dem Argument `Contents`, das standardmäßig auf `True` gesetzt ist, werden die Zellen eines Tabellenblattes geschützt.
- Das Argument `Scenarios` gilt nur für Arbeitsblätter und bedeutet, dass bestimmte Ansichten und Einstellungen, wie z. B. der eingestellte Zoom, geschützt werden. Die Standardeinstellung ist dabei ebenfalls `True`.
- Das letzte Argument mit dem Namen `UserInterFaceOnly` nimmt den Wert `True` an. Damit schützen Sie die Benutzeroberfläche, jedoch nicht die Makros. Ohne die Angabe dieses Arguments gilt der Schutz sowohl für Makros als auch für die Benutzeroberfläche.

6.7.1 Tabellenschutz aufheben

Zum Deaktivieren des Tabellenschutzes reicht es, wenn Sie bei der Methode `Unprotect` das Passwort angeben. Sollten Sie Ihr Tabellenblatt ohne Passwort geschützt haben, reicht der Befehl `ActiveSheet.Unprotect`.

```
Sub BlattschutzAusschalten()

   ActiveSheet.Unprotect ("test")

End Sub
```

Listing 6.26 Tabellenschutz aufheben

6.7.2 Alle Tabellen einer Arbeitsmappe schützen

Wenn Sie alle Tabellenblätter einer Arbeitsmappe schützen und dabei dasselbe Passwort verwenden möchten, können Sie das Makro aus Listing 6.27 nutzen:

```
Sub PasswortFürAlleBlätterEinstellen()
  Dim intTabz As Integer
  Dim intz As Integer

    intTabz = ActiveWorkbook.Worksheets.Count

    For i = intz To intTabz

      Worksheets(intz).Protect DrawingObjects:=True, _
      Contents:=True, Scenarios:=True, Password:="test"

    Next intz

End Sub
```

Listing 6.27 Alle Tabellenblätter einer Arbeitsmappe schützen

Um den Blattschutz für alle Tabellenblätter in der Mappe wieder aufzuheben, starten Sie das Makro aus Listing 6.28:

```
Sub PasswortAlleBlätterEntfernen()
  Dim intTabz As Integer
  Dim intz As Integer

    intTabz = ActiveWorkbook.Worksheets.Count

    For intz = 1 To Tabz

      Worksheet(intz).Unprotect "test"
    Next intz

End Sub
```

Listing 6.28 Blattschutz auf allen Tabellenblättern der Mappe entfernen

6.7.3 Weitere Schutzfunktionen ab Excel 2002

Eine sehr gute Verbesserung gegenüber den Vorversionen von Excel können Sie auch beim Schützen Ihrer Tabellen ab der Version Excel 2002 feststellen. Sie haben seitdem die Möglichkeit, zwar einen Blattschutz einzustellen, aber einzelne Aktionen trotz eingestelltem Blattschutz zu ermöglichen. So können Sie zum Beispiel festlegen, dass Anwender in einer geschützten Tabelle die Filter verwenden sowie Formatierungen durchführen und Zeilen und Spalten einfügen dürfen. Diese und weitere Möglichkeiten sehen Sie, wenn Sie aus dem Menü Extras den Befehl Blatt schützen wählen (Excel 2002 bis Excel 2003). In den Excel-Versionen 2007 bis 2021 sowie in Microsoft 365 klicken Sie im Ribbon Überprüfen auf die Schaltfläche Blatt schützen.

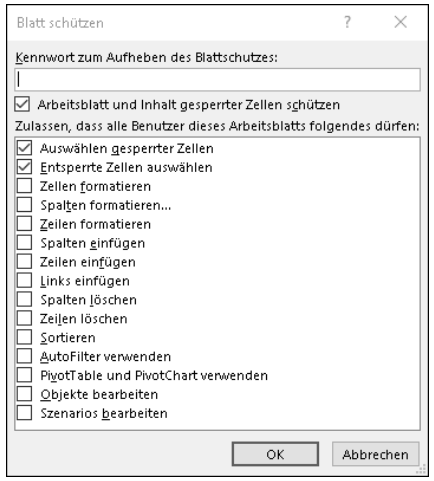


Abbildung 6.3 Erweiterte Schutzmöglichkeiten ab Excel 2002

Diese manuellen Einstellungen können Sie aber auch über ein Makro durchführen. Das folgende Makro lässt in einer geschützten Tabelle alle aktivierten Optionen zu:

```
Sub TabelleSchuetzen()

  Tabelle2.Protect _
  DrawingObjects:=True, _
  Contents:=True, Scenarios:=True, _
```

```
    AllowFormattingCells:=True, _
    AllowFormattingColumns:=True, _
    AllowFormattingRows:=True, _
    AllowInsertingColumns:=True, _
    AllowInsertingRows:=True, _
    AllowInsertingHyperlinks:=True, _
    AllowDeletingColumns:=True, _
    AllowDeletingRows:=True, _
    AllowSorting:=True, _
    AllowFiltering:=True, _
    AllowUsingPivotTables:=True

End Sub
```

Listing 6.29 Tabelle schützen (ab Excel 2002)

6.7.4 Passwort – Einstellungsdialog mit verschlüsseltem Passwort aufrufen

Das Makro aus Listing 6.30 ruft den Dialog BLATT SCHÜTZEN bereits mit voreingestelltem Passwort auf:

```
Sub DialogBlattSchutzAufrufen()

    Application.Dialogs(xlDialogProtectDocument).Show 1, 1, "TEST"

End Sub
```

Listing 6.30 Blattschutzdialog mit verschlüsseltem Passwort aufrufen

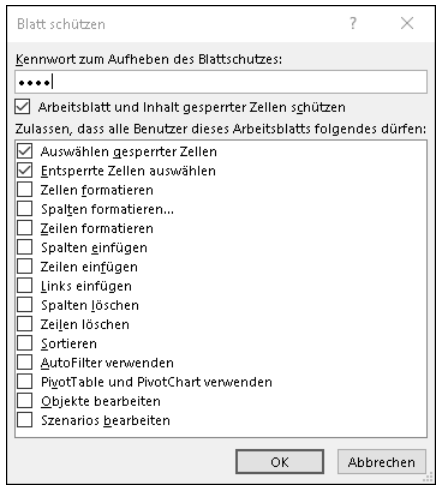


Abbildung 6.4 Der integrierte Dialog »Blatt schützen« wurde aufgerufen.

6.8 Tabellen einstellen

Wenn Sie mehrere Tabellen in einer Arbeitsmappe verwalten, dann ist es mitunter nützlich, bestimmte Einstellungen für alle Tabellen gleichermaßen vorzunehmen. Ich denke da beispielsweise an eine einheitliche Zoomeinstellung, eine überall gleiche Kopf- und Fußzeile, an eine einheitliche Positionierung des Cursors und des Bildlaufs und vieles mehr.

Hinweis

Sie finden alle folgenden Makros in der Datei *Tabellen.xlsm* im Modul `mdl_Einstellen`.

6.8.1 Registerlaschen ein- und ausblenden

Standardmäßig werden die Registerlaschen am unteren Bildrand von Excel angezeigt. Wenn diese Standardeinstellung Sie stört, dann können Sie die Anzeige der Registerlaschen ausblenden. Im Makro aus Listing 6.31 werden die Registerlaschen in der kompletten Arbeitsmappe ein- bzw. ausgeblendet:

```
Sub RegisterAusEinblenden()

    ActiveWindow.DisplayWorkbookTabs = Not ActiveWindow.DisplayWorkbookTabs

End Sub
```

Listing 6.31 Tabellenregister ein- oder ausblenden

Über die Eigenschaft `DisplayWorkbookTabs` können Sie die Registerlaschen Ihrer Tabelle ein- und ausblenden. Setzen Sie diese Eigenschaft auf den Wert `True`, wenn Sie die Registerlaschen anzeigen möchten. Weisen Sie der Eigenschaft den Wert `False` zu, um die Registerlaschen wieder auszublenden. Den dynamischen Wechsel zwischen Anzeigen und Ausblenden der Registerlaschen bekommen Sie über eine Gegenüberstellung hin. Dabei können Sie sich diese Gegenüberstellung wie einen Lichtschalter vorstellen, also an und aus.

6.8.2 Tabellenansicht anpassen

In einer Tabelle können Sie sich die Daten jederzeit etwas größer oder auch verkleinert anzeigen lassen. Um diese Aufgabe schrittweise zu erledigen, können Sie ein Makro einsetzen, das die Ansicht um jeweils 10 % vergrößert oder verkleinert:

```
Sub TabelleVergrößern()
 Dim intFaktor As Integer
```

```
    intFaktor  = ActiveWindow.Zoom
    ActiveWindow.Zoom = intFaktor  + 10

End Sub
```

Listing 6.32 Tabellenansicht vergrößern

Über die Eigenschaft `Zoom` können Sie einen Vergrößerungsfaktor bis zu 400 % einstellen, aber bei der Verkleinerung einer Tabelle können Sie keine niedrigeren Werte als 10 % einstellen. Bei 100 % ist eine 1:1-Ansicht gegeben.

```
Sub VerkleinernTabelle()
 Dim intFaktor  As Integer

  intFaktor  = ActiveWindow.Zoom
  ActiveWindow.Zoom = intFaktor  - 10

End Sub
```

Listing 6.33 Tabellenansicht verkleinern

6.8.3 Einen einheitliche Zoomeinstellung vornehmen

Beim nächsten Beispiel aus Listing 6.34 stellen wir für alle Tabellen für eine bessere Lesbarkeit die Zoomeinstellung auf 120 % ein.

```
Sub ZoomEinstellungVornehmen()
  Dim wksBlatt As Worksheet
  Dim IntTab As Integer

  IntTab = ActiveSheet.Index
  For Each wksBlatt In ActiveWorkbook.Worksheets

    wksBlatt.Activate
    ActiveWindow.Zoom = 120

  Next wksBlatt

  Worksheets(IntTab).Select

End Sub
```

Listing 6.34 Eine einheitliche Zoomeinstellung für alle Tabellen vornehmen

Da die Zoomeinstellung immer nur für eine Tabelle einstellbar ist, setzen Sie eine Schleife des Typs `For Each ... Next` auf und durchlaufen Tabelle für Tabelle. Innerhalb der Schleife wenden Sie die Methode `Activate` an, um die jeweilige Tabelle zu aktivieren. Danach weisen Sie der Eigenschaft `Zoom` des aktiven Fensters die gewünschte Einstellung zu.

Da Sie im Prinzip durch die ganze Mappe hüpfen, sollten Sie sich zu Beginn des Makros in einer Variablen merken, von welcher Tabelle aus Sie starten. Dies können Sie über die Eigenschaft `Index` abfragen. Am Ende selektieren Sie diese Startabelle wieder, indem Sie sie über die Methode `Select` auswählen.

Hinweis

Die Methode `Activate` funktioniert übrigens auch bei ausgeblendeten Tabellen.

Die Methode `Select` verursacht einen Laufzeitfehler, wenn versucht wird, eine ausgeblendete Tabelle zu selektieren.

6.8.4 Tabellenblätter sortieren

In umfangreichen Excel-Arbeitsmappen geht leicht einmal der Überblick verloren. Aus diesem Grund ist es vorteilhaft, die Tabellen alphabetisch nach Tabellennamen sortiert in der Arbeitsmappe anzuordnen.

Das Makro für die Sortierung der Tabellenblätter lautet:

```
Sub ArbeitsblätterSortieren()
  Dim intMax As Integer
  Dim intz As Integer
  Dim intn As Integer

  Application.ScreenUpdating = False

  intMax = ActiveWorkbook.Worksheets.Count

  For intz = 1 To intMax

    For intn = intz To intMax

      If UCase(Worksheets(intn).Name) < UCase(Worksheets(intz).Name) Then
         Worksheets(intn).Move before:=Worksheets(intz)
      End If
```

```
    Next intn

  Next intz

 Application.ScreenUpdating = True

End Sub
```

Listing 6.35 Alle Tabellen werden alphabetisch in der Arbeitsmappe angeordnet.

Um Arbeitsblätter zu sortieren, durchlaufen Sie zwei verschachtelte `For ... Next`-Schleifen. Beide haben als Endbedingung immer die Anzahl der Tabellen, die in der Mappe enthalten sind. Innerhalb der zweiten Schleife werden die Namen der Tabellenblätter verglichen.

Beim Vergleich der Tabellennamen werden diese erst einmal in Großbuchstaben umgewandelt, um sicherzustellen, dass die Groß- und Kleinschreibung beim Sortiervorgang keine Rolle spielen wird. Je nach Vergleichsergebnis werden die einzelnen Tabellen anschließend innerhalb der Arbeitsmappe mithilfe der Methode `Move` verschoben oder an ihrer Stelle gelassen.

6.8.5 Kopf- und Fußzeilen einrichten

Standardmäßig werden in Excel 2000 bis 2021 keine Kopf- und Fußzeilen ausgedruckt. Um diese müssen Sie sich selbst kümmern. Dazu verwenden Sie das Objekt `PageSetup`, das Sie für das Tabellenblatt anwenden können.

Fußzeile mit Anwendernamen

So fügen Sie beispielsweise den Namen des Anwenders, den genauen Speicherpfad, das heutige Datum oder andere Angaben aus den Dokumenteigenschaften als Kopf- oder Fußzeile ein.

```
Sub BenutzerNameInFußzeile()

  ActiveSheet.PageSetup.RightFooter = Environ("username")

End Sub
```

Listing 6.36 Fußzeile mit Benutzername generieren

Fußzeile mit Pfad

Wenn Sie eine Fußzeile mit dem Namen der Arbeitsmappe definieren, können Sie leider aus dieser Angabe nicht ersehen, wo diese Arbeitsmappe gespeichert ist. Daher erstellen Sie ein Makro, das Ihnen eine Fußzeile mit dem Namen des kompletten Pfades der Datei ausgibt:

```
Sub FußzeileMitPfad()

  ActiveSheet.PageSetup.LeftFooter = _
  ActiveWorkbook.FullName

End Sub
```

Listing 6.37 Fußzeile mit kompletter Pfadangabe der Datei erstellen

Kopfzeile mit Datums- und Zeitangabe

Im nächsten Beispiel fügen Sie ein vierstelliges Datum in die Kopfzeile sowie die aktuelle Uhrzeit in die Fußzeile ein:

```
Sub KopfzeileMit4stelligemDatum()

  With ActiveSheet.PageSetup
    .LeftHeader = ""
    .CenterHeader = Format(Date, "dd.mm.yyyy")
    .RightHeader = ""
    .LeftFooter = ""
    .CenterFooter = Time
    .RightFooter = ""
  End With

  ActiveWindow.SelectedSheets.PrintPreview

End Sub
```

Listing 6.38 Kopf- und Fußzeilen mit Datums- und Zeitangaben bestücken

Mit der Anweisung `With` führen Sie eine Reihe von Anweisungen für ein bestimmtes Objekt aus, ohne den Namen des Objekts mehrmals angeben zu müssen. Dadurch sparen Sie eine Menge Schreibarbeit, und das Ganze sieht auch noch übersichtlicher aus. Um das Datum in eine bestimmte Form zu bringen, setzen Sie die Funktion `Format` ein. Möglich wäre auch die Anweisung

```
CenterHeader = Format(Date, "Long Date")
```

die zur Folge hätte, dass das Datum ausgeschrieben wird (z. B. »Donnerstag, 16. Oktober 2017«).

Mit der Methode `PrintPreview` zeigen Sie direkt nach dem Festlegen der Kopf- und Fußzeilen das Ergebnis, so wie es in der Seitenansicht aussieht.

Fußzeile mit Dokumenteigenschaften füllen

Im nächsten Beispiel greifen Sie auf die Dokumenteigenschaften Ihrer Excel-Arbeitsmappe zurück.

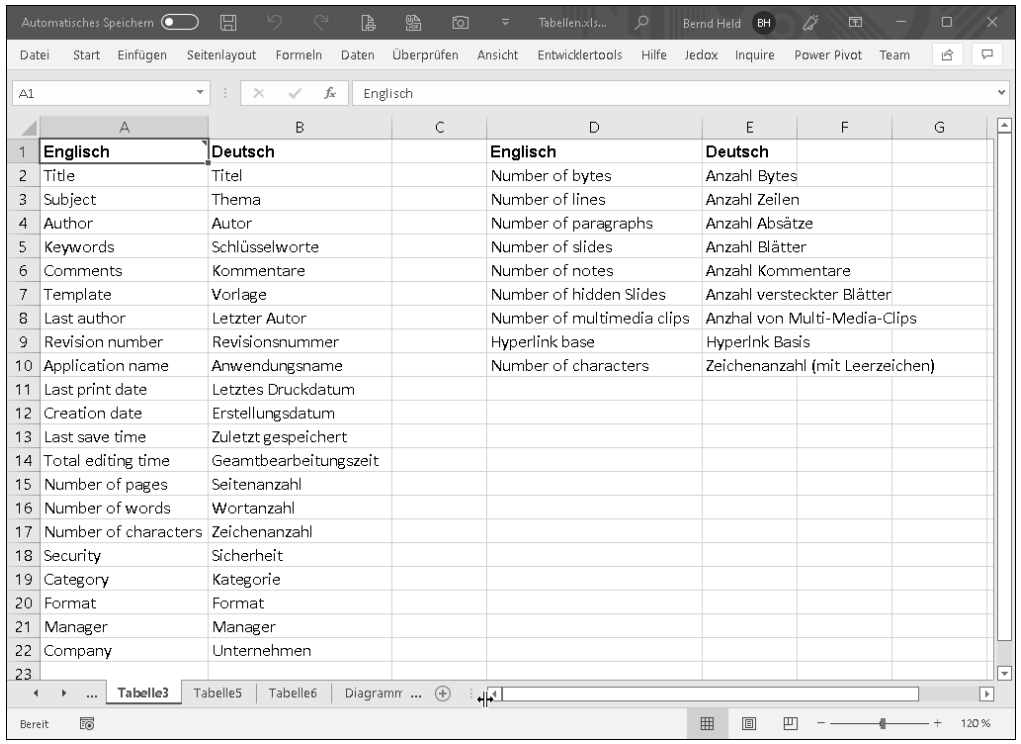
In der Version Excel 2007 klicken Sie auf die runde Office-Schaltfläche links oben und wählen aus dem Menü den Befehl VORBEREITEN • EIGENSCHAFTEN.

In den Versionen Excel 2010 bis 2021 sowie in Microsoft 365 klicken Sie auf die DATEI-Schaltfläche links oben und wählen aus dem Menü den Befehl INFORMATIONEN. Dort klicken Sie auf EIGENSCHAFTEN und wählen ERWEITERTE EIGENSCHAFTEN. In dem neuen Fenster EIGENSCHAFTEN wählen Sie das Register ZUSAMMENFASSUNG.

```
Sub DateieigenschaftInFusszeile()

  With ActiveSheet.PageSetup
    .LeftFooter = _
     Activeworkbook.BuiltinDocumentProperties("Company")
    .RightFooter = _
     Activeworkbook.BuiltinDocumentProperties("Author")
  End With

  ActiveWindow.SelectedSheets.PrintPreview

End Sub
```

Listing 6.39 Fußzeile mit Dokumenteigenschaften füllen

Da Sie die Dokumenteigenschaften auf Englisch ansprechen müssen und in der Onlinehilfe lediglich die deutschen Begriffe aufgeführt werden, orientieren Sie sich an Abbildung 6.5.



	A	B	C	D	E	F	G
1	**Englisch**	**Deutsch**		**Englisch**	**Deutsch**		
2	Title	Titel		Number of bytes	Anzahl Bytes		
3	Subject	Thema		Number of lines	Anzahl Zeilen		
4	Author	Autor		Number of paragraphs	Anzahl Absätze		
5	Keywords	Schlüsselworte		Number of slides	Anzahl Blätter		
6	Comments	Kommentare		Number of notes	Anzahl Kommentare		
7	Template	Vorlage		Number of hidden Slides	Anzahl versteckter Blätter		
8	Last author	Letzter Autor		Number of multimedia clips	Anzhal von Multi-Media-Clips		
9	Revision number	Revisionsnummer		Hyperlink base	HyperInk Basis		
10	Application name	Anwendungsname		Number of characters	Zeichenanzahl (mit Leerzeichen)		
11	Last print date	Letztes Druckdatum					
12	Creation date	Erstellungsdatum					
13	Last save time	Zuletzt gespeichert					
14	Total editing time	Geamtbearbeitungszeit					
15	Number of pages	Seitenanzahl					
16	Number of words	Wortanzahl					
17	Number of characters	Zeichenanzahl					
18	Security	Sicherheit					
19	Category	Kategorie					
20	Format	Format					
21	Manager	Manager					
22	Company	Unternehmen					
23							

Abbildung 6.5 Die Gegenüberstellung der Dokumenteigenschaften (Englisch – Deutsch)

Kopfzeile mit Logo einrichten

Sie können ab der Excel-Version 2002 standardmäßig Grafiken in die Kopf- und Fußzeile integrieren. Um beispielsweise eine Grafik in die Kopfzeile der aktiven Tabelle einzufügen, nutzen Sie den folgenden Code:

```
Sub GrafikInKopfzeileEinfügen()

  With ActiveSheet.PageSetup
   .RightHeaderPicture.Filename = ThisWorkbook.Path & "\Logo.Jpg"
   .RightHeader = "&G"
  End With

End Sub
```

Listing 6.40 Kopfzeile mit einem Logo ausstatten (Tabelle)

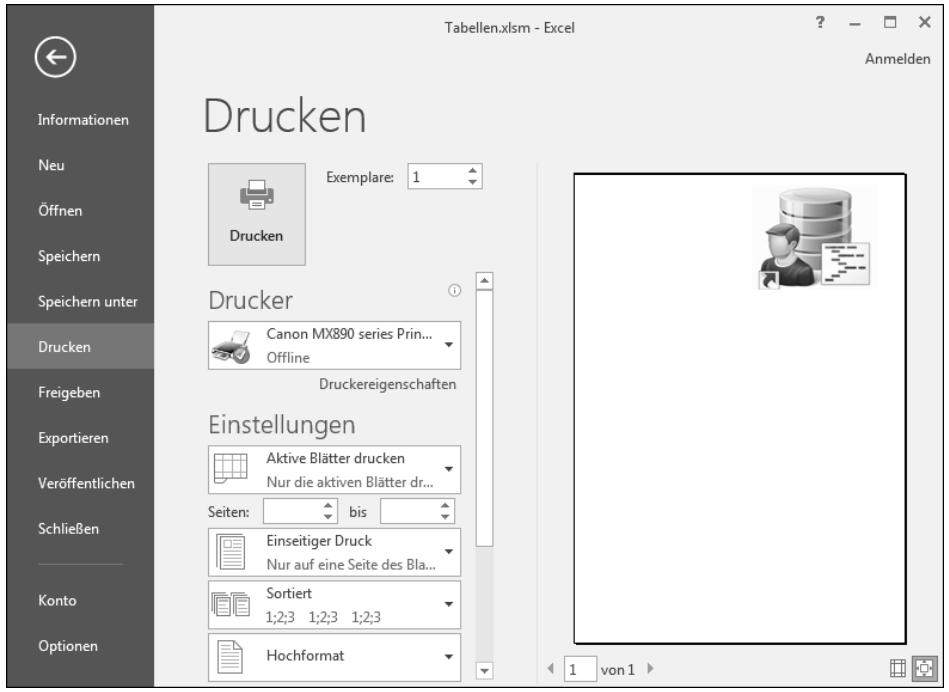


Abbildung 6.6 Ein Logo in die Kopfzeile der aktiven Tabelle einfügen

Über die Eigenschaft `RightHeaderPicture` weisen Sie dem rechten Rand der Kopfzeile Ihrer Tabelle die angegebene Grafik zu. Neben dieser Eigenschaft gibt es fünf weitere, die Sie in Tabelle 6.1 entdecken können:

Eigenschaft	Beschreibung
`RightHeaderPicture`	Bild rechts in der Kopfzeile
`CenterHeaderPicture`	Bild in der Mitte der Kopfzeile
`LeftFooterPicture`	Bild links in der Fußzeile
`CenterFooterPicture`	Bild in der Mitte der Fußzeile
`RightFooterPicture`	Bild rechts in der Fußzeile

Tabelle 6.1 Die Positionen in den Kopf- und Fußzeilen bestimmen

Mithilfe der Eigenschaft `FileName` geben Sie bekannt, wo die Grafik zu finden ist und wie diese heißt. Mit der Eigenschaft `RightHeader` definieren Sie, was Sie konkret in der Kopfzeile machen möchten. Dazu weisen Sie dieser Eigenschaft einen Formatcode zu. In Tabelle 6.2 finden Sie die dabei möglichen Formatcodes und deren Bedeutung.

Formatcode	Beschreibung
`&L`	Richtet folgende Zeichen links aus.
`&C`	Zentriert das folgende Zeichen.
`&R`	Richtet folgende Zeichen rechts aus.
`&E`	Schaltet doppeltes Unterstreichen ein oder aus.
`&X`	Schaltet Hochstellen ein oder aus.
`&Y`	Schaltet Tiefstellen ein oder aus.
`&B`	Schaltet Fettdruck ein oder aus.
`&I`	Schaltet Kursivdruck ein oder aus.
`&U`	Schaltet Unterstreichen ein oder aus.
`&S`	Schaltet Durchstreichen ein oder aus.
`&D`	das aktuelle Datum
`&T`	Druckt die aktuelle Zeit.
`&F`	Druckt den Namen des Dokuments.
`&A`	Druckt den Namen des Registers einer Arbeitsmappe.
`&P`	Druckt die Seitenzahl.
`&P+Zahl`	Druckt die Seitenzahl zuzüglich der angegebenen Zahl.
`&&`	Druckt ein einzelnes kaufmännisches Und-Zeichen (&).
`&"Schriftart"`	Druckt die folgenden Zeichen in der angegebenen Schriftart; die Schriftart muss von Anführungszeichen eingeschlossen sein.
`&nn`	Druckt die folgenden Zeichen im angegebenen Schriftgrad. Geben Sie für den Schriftgrad eine zweistellige Zahl an.
`&N`	Druckt die Gesamtanzahl der Seiten eines Dokuments.

Tabelle 6.2 Die Formatcodes für die Kopf- und Fußzeile

Sie brauchen sich übrigens keine Sorgen zu machen, wenn Sie eine Arbeitsmappe mit integrierten Grafiken verschicken. Sie müssen keineswegs auch die Grafiken mitversenden. Ist die Grafik einmal in der Kopf- oder Fußzeile integriert, bleibt sie auch darin.

Bei der Lösung aus Listing 6.40 haben wir das Logo lediglich in die Kopfzeile der aktiven Tabelle integriert. Möchten Sie das Logo auf allen Tabellen Ihrer Arbeitsmappe einfügen, dann starten Sie das Makro aus Listing 6.41:

```
Sub GrafikInKopfzeileinfügenAlleTabellen()
 Dim wksBlatt As Worksheet

  For Each wksBlatt In ActiveWorkbook.Worksheets

    wksBlatt.PageSetup.LeftHeaderPicture.Filename = _
    ActiveWorkbook.Path & "\Logo.Jpg"
    wksBlatt.PageSetup.LeftHeader = "&G"

  Next wksBlatt

End Sub
```

Listing 6.41 Kopfzeile mit Logo ausstatten (ganze Arbeitsmappe)

Mehrzeilige Fußzeile anlegen

Oft findet man bei Geschäftspapieren und offiziellen Briefen eine mehrzeilige Fußzeile. Diese Fußzeile ist aber in den meisten Fällen bereits fest auf das Papier aufgedruckt und wird demnach von Excel nicht mehr erstellt. Die Standardeinstellung bei Kopf- und Fußzeilen in Excel sieht normalerweise mehrzeilige Fußzeilen nicht vor, beziehungsweise es ist relativ aufwendig, solche Mehrzeiler zu erstellen, weil dazu zum einen die Schriftgröße herabgesetzt und zum anderen mehr Platz für die Fußzeile einkalkuliert werden muss. Darüber hinaus kommt es darauf an, welche Informationen Sie in der Fußzeile ausgeben möchten. Um beispielsweise das Erstellungsdatum oder das letzte Änderungsdatum einer Arbeitsmappe in die Fußzeile zu bringen, bedarf es schon eines Makros:

```
Sub FußzeileSpezialAktiveTabelle()

  With ActiveSheet.PageSetup
    .BottomMargin = 56
    .FooterMargin = 42
    .LeftFooter = "&8" & _
    Application.WorksheetFunction.Rept("_", 60) & vbCr & _
    "Erstellungsdatum:   " & _
    ActiveWorkbook.BuiltinDocumentProperties _
    ("Creation date") & vbCr & _
    "Letzte Änderung:     " & _
    ActiveWorkbook.BuiltinDocumentProperties _
```



```
    ("last save time") & vbCr & _
    "Ersteller der Mappe:  " & _
    ActiveWorkbook.BuiltinDocumentProperties _
    ("author") & vbCr & "Pfad: " & ActiveWorkbook.FullName
  End With

End Sub
```

Listing 6.42 Mehrzeilige Fußzeile erstellen (aktive Tabelle)

Im ersten Schritt des Makros wird die Anweisung `With` auf das Objekt `PageSetup` der aktiven Tabelle (`ActiveSheet`) angewendet, um den Code übersichtlicher zu machen und um Schreibarbeit zu sparen. Danach müssen Sie den Befehl `ActiveSheet.PageSetup` nicht in jeder Zeile wiederholen. Stattdessen genügt es, wenn Sie einen Punkt als erstes Zeichen vor die Eigenschaften setzen, die auf die Seiteneinrichtung angewendet werden sollen. Über die Eigenschaft `BottomMargin` geben Sie den Abstand zum unteren Papierrand an. Mithilfe der Eigenschaft `FooterMargin` stellen Sie den Abstand der Fußzeile ebenfalls vom unteren Papierrand ein. Diese Abstände müssen Sie in der Einheit *Punkt* angeben. Ein Punkt entspricht dabei in etwa 0,35 mm. So entsprechen 56 Punkt ungefähr 2 cm, und 42 Punkt sind in etwa 1,5 cm.

Über die Eigenschaft `LeftFooter` legen Sie den Inhalt des linken Teils der Fußzeile fest. Standardmäßig ist die Fußzeile in drei Teile gegliedert: Der linke Teil wird durch die Eigenschaft `LeftFooter`, der mittlere Teile durch die Eigenschaft `CenterFooter` und der rechte Teil durch die Eigenschaft `RightFooter` repräsentiert. Insgesamt dürfen jedoch nicht mehr als 255 Zeichen in der Fußzeile stehen. Aus diesem Grund wird im vorgestellten Beispiel nur der linke Teil der Fußzeile befüllt, und die restlichen Teile bleiben leer. Mit dem Steuerzeichen `"&8"` sorgen Sie dafür, dass der Schriftgrad auf 8 herabgesetzt wird, um Platz zu sparen. Alle folgenden Informationen werden danach in der Schriftgröße 8 (Standard ist 10) in die Fußzeile geschrieben.

In der ersten Zeile wird ein horizontaler Trennstrich eingefügt. Dazu wird die Tabellenfunktion WIEDERHOLEN eingesetzt, die in VBA über die Anweisung `WorkSheetFunction.Rept` angesprochen wird. Dieser Funktion übergeben Sie im ersten Argument das Zeichen (hier den Unterstrich), das wiederholt werden soll. Im zweiten Argument geben Sie die Anzahl der Wiederholungen an. Immer im Hinblick darauf, dass insgesamt nur 255 Zeichen in der Fußzeile/Kopfzeile verwendet werden dürfen, kann dieses zweite Argument je nach sonstigem Füllgehalt der Fußzeile höher oder niedriger eingestellt werden. Über die Konstante `VbCr` werden die folgenden Informationen jeweils in der nächsten Zeile der Fußzeile ausgegeben. Unter anderem wird das Erstellungsdatum der aktiven Arbeitsmappe mithilfe der Dokumenteigenschaft »Erstellt am« ermittelt. Um diese Dokumenteigenschaft aus der Arbeitsmappe per VBA zu ermitteln, setzen Sie das Auflistungsobjekt `BuiltinDocumentProperties` ein,

dem Sie die gewünschte Dokumenteigenschaft als Text übergeben. Um das Erstellungsdatum einer Arbeitsmappe zu ermitteln, übergeben Sie den Text `"Creation date"`. Das Datum der letzten Änderung ermitteln Sie über den Text `"last save time"`. Über den Text `"author"` fragen Sie den Ersteller der Arbeitsmappe ab. In der letzten Zeile der Fußzeile werden der komplette Pfad und der Dateiname mithilfe der Eigenschaft `FullName` ermittelt.

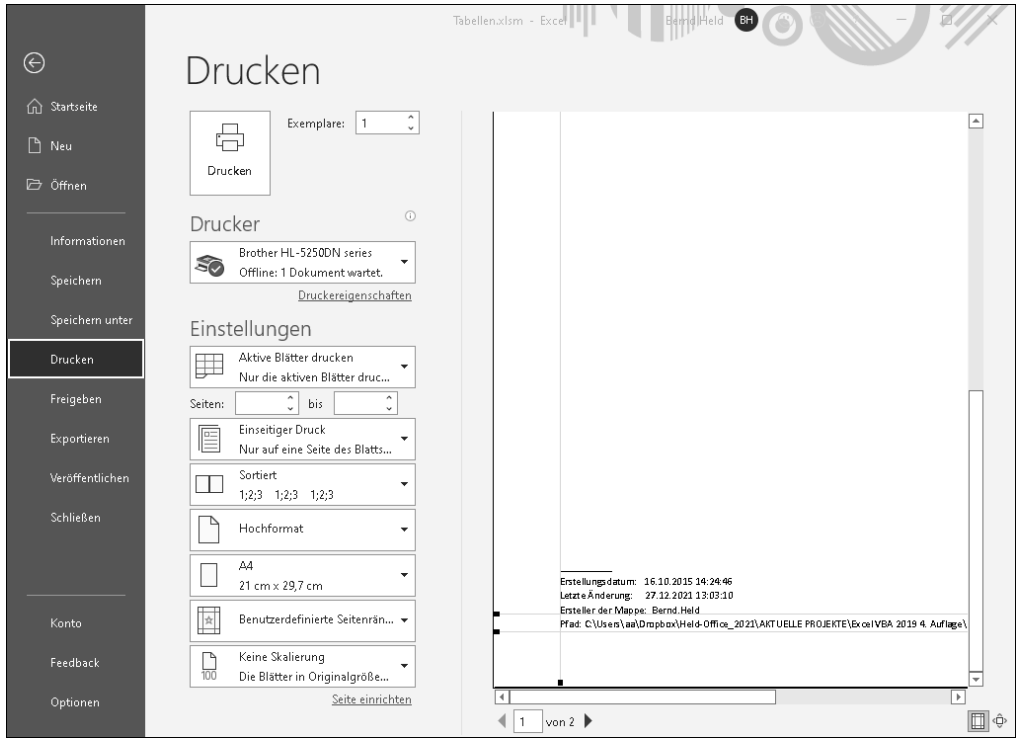


Abbildung 6.7 Eine mehrzeilige Fußzeile einfügen

Kopf- und Fußzeileneinträge leeren

Das Makro aus Listing 6.43 entfernt alle Einträge aus der Kopf- und Fußzeile, sogar das vorher eingefügte Logo.

```
Sub KopfFussLeeren()
  Dim wksBlatt As Worksheet

  For Each wksBlatt In ActiveWorkbook.Worksheets

      With wksBlatt.PageSetup
          .LeftHeader = ""
          .CenterHeader = ""
          .RightHeader = ""
          .LeftFooter = ""
          .CenterFooter = ""
          .RightFooter = ""
      End With

  Next wksBlatt

End Sub
```

Listing 6.43 Alle Inhalte der Kopf- und Fußzeile leeren

In einer Schleife des Typs `For Each ... Next` arbeiten Sie alle Tabellen der Arbeitsmappe ab. Über das Objekt `PageSetup` haben Sie Zugriff auf alle Kopf- und Fußzeilen der jeweiligen Tabelle. Weisen Sie den Eigenschaften jeweils eine leere Zeichenfolge zu, um die Inhalte der Kopf- und Fußzeile zu entfernen.

6.8.6 Druckbereiche festlegen

Um Papier zu sparen, können Sie vor dem Drucken einen Druckbereich festlegen. Im ersten Beispiel setzen Sie einen Druckbereich, der der momentanen Markierung entspricht. Markieren Sie also den Bereich, den Sie drucken möchten, und starten Sie danach folgendes Makro:

```
Sub DruckbereichSetzen()

  ActiveSheet.PageSetup.PrintArea = Selection.Address

End Sub
```

Listing 6.44 Druckbereich in Tabelle festlegen

Mit der Eigenschaft `PrintArea` legen Sie den Druckbereich fest. Wenn Sie Ihren Druckbereich fix gestalten möchten, setzen Sie das Makro aus Listing 6.45 ein:

```
Sub DruckbereichFestlegen()

  Worksheets("Tabelle1").PageSetup.PrintArea = "$A$1:$E$80"
  'oder
  Tabelle1.PageSetup.PrintArea = "$A$1:$E$80"

End Sub
```

Listing 6.45 Druckbereich in Tabelle konstant festlegen

Eine weitere Variante ist, den Druckbereich nach dem verwendeten Bereich zu bestimmen. Dazu setzen Sie die Eigenschaft `CurrentRegion` ein. Diese Eigenschaft ermittelt, beginnend von einer Zelle, den umliegenden Bereich. Sobald eine Leerzeile bzw. Leerspalte kommt, wird der Bereich abgeschlossen.

```
Sub DruckbereichNachVerwendungFestlegen()

  With Tabelle1

    .PageSetup.PrintArea = .Range("A1").CurrentRegion.Address

  End With

End Sub
```

Listing 6.46 Druckbereich nach Verwendung festlegen

Tipp

Um einen Druckbereich wieder aufzuheben, setzen Sie die Eigenschaft `PrintArea` auf den Wert `False` oder auf die leere Zeichenfolge (""). Damit wird das gesamte Blatt wieder als Druckbereich festgelegt.

6.8.7 Das Tabellengitternetz ein- und ausschalten

Sollten Sie das Gitternetz einer Tabelle ausschalten wollen, können Sie hierfür die Eigenschaft `DisplayGridlines` verwenden.

Das Makro aus Listing 6.47 schaltet die Anzeige der Gitternetzlinien für die aktive Tabelle ein und aus.

```
Sub UmschaltenGitternetzEinAus()

  ActiveWindow.DisplayGridlines = _
  Not ActiveWindow.DisplayGridlines

End Sub
```

Listing 6.47 Die Gitternetzanzeige für Tabellen ein- und ausschalten

6.8.8 Zeilen- und Spaltenköpfe ein- und ausblenden

Die Sichtbarkeit der Zeilen- und Spaltenbeschriftung regeln Sie über die Eigenschaft `DisplayHeadings`. Das Makro aus Listing 6.48 blendet die Spalten- und Zeilenköpfe im Wechsel ein und aus.


```
Sub SpaltenUndZeilenEinUndAusblenden()

    If ActiveWindow.DisplayHeadings = False Then
        ActiveWindow.DisplayHeadings = True
    Else
        ActiveWindow.DisplayHeadings = False
    End If

End Sub
```

Listing 6.48 Spalten- und Zeilenköpfe ein- und ausschalten

6.8.9 Cursor einstellen auf Zelle A1

Bei der Lösung aus Listing 6.49 wird bei allen Tabellen der Arbeitsmappe der Cursor in Zelle A1 gesetzt. Gegebenenfalls wird auch nach oben gescrollt.

```
Sub CursorEinstellenA1()
 Dim wksBlatt As Worksheet

  For Each wksBlatt In ActiveWorkbook.Worksheets

    Application.GoTo Reference:=wksBlatt.Range("A1"), scroll:=True

  Next wksBlatt

End Sub
```

Listing 6.49 Cursor in jeder Tabelle einheitlich positionieren

Über eine Schleife des Typs `For Each ... Next` werden alle Tabellen der Arbeitsmappe nacheinander verarbeitet. Innerhalb der Schleife wenden Sie die Methode `GoTo` an, um in der jeweiligen Tabelle den Cursor in Zelle A1 zu setzen. Über den Parameter `Scroll` legen Sie fest, ob Sie einen Bildlauf durchführen, also scrollen möchten.

6.9 Tabellenblätter drucken und PDF erstellen

Wenn Sie die Kopf- und Fußzeilen gesetzt und eventuell auch einen Druckbereich eingestellt haben, gehen Sie zum Thema Drucken über. Drucken können Sie entweder ein oder mehrere Tabellenblätter, die ganze Arbeitsmappe, einen Druckbereich oder eine Markierung. Je nach Wunsch schreiben Sie dazu Makros.

Seit der Excel-Version 2007 gibt es eine PDF-Schnittstelle, die Sie über die Methode `ExportAsFixedFormat` aufrufen.

Hinweis

Sie finden alle folgenden Makros in der Datei *Tabellen.xlsm* im Modul `mdl_Drucken`.

```
Sub TabellenblattDrucken()

  Sheets("Tabelle1").PrintOut
  'oder
  Tabelle1.PrintOut

End Sub
```

Listing 6.50 Eine bestimmte Tabelle drucken

6.9.1 Mehrere Kopien drucken

Mithilfe der Methode `PrintOut` drucken Sie aus. Möchten Sie gleich mehrere Kopien ausgeben, so stellen Sie die gewünschte Anzahl beim Parameter `Copies` ein.

```
Sub TabellenblattDruckenMitKopie()

  Worksheets("Tabelle1").PrintOut Copies:=2

  'oder

  Tabelle1.PrintOut Copies:=2

End Sub
```

Listing 6.51 Mehrere Kopien einer Tabelle drucken

6.9.2 Markierte Bereiche drucken

Beim folgenden Makro markieren Sie entweder vorher den Bereich, den Sie drucken möchten, mit der Maus, oder Sie weisen den Bereich per VBA zu, zum Beispiel: `Range("A1:D10").Select`. Führen Sie dann das Makro aus Listing 6.52 aus:

```
Sub MarkierungDrucken()

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

End Sub
```

Listing 6.52 Markierten Bereich drucken

6.9.3 Mehrere Tabellenblätter drucken

Möchten Sie mehrere Tabellenblätter drucken, so bilden Sie ein Array mit den gewünschten Tabellenblättern und schicken den Druckauftrag auf den Weg:

```
Sub MehrereTabellenblätterDrucken()

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

End Sub
```

Listing 6.53 Mehrere Tabellenblätter drucken

6.9.4 Tabelle als PDF ablegen

Bei der Lösung aus Listing 6.54 wird von Tabelle3 zunächst eine eigenständige Excel-Arbeitsmappe erstellt und danach ein PDF gezogen:

```
Sub ExportAlsPDFundxlsx()
  'Warnmeldungen temporär ausschalten
  Application.DisplayAlerts = False

  'Bildschirmaktualisierung temporär ausschalten
  Application.ScreenUpdating = False

  'prüfen, ob ein Unterordner Exporte verfügbar ist.
  If Dir(ThisWorkbook.Path & "\Exporte", vbDirectory) = "" Then
    MkDir ThisWorkbook.Path & "\Exporte"
  End If

'Kopiert die Tabelle automatisch in eine neue Mappe
'(Copy ohne weitere Argumente)
  Tabelle3.Copy

  'Export als xlsx
  ActiveWorkbook.SaveAs Filename:=ThisWorkbook.Path & _
  "\Exporte\" & Format(Date, "YYYY.MM.DD_") & ActiveSheet.Name & ".xlsx"

  'Export als PDF
  ActiveWorkbook.ExportAsFixedFormat Type:=xlTypePDF, _
  Filename:=ThisWorkbook.Path & "\Exporte\" & _
  Format(Date, "YYYY.MM.DD_") & ActiveSheet.Name & ".pdf"

  ActiveWorkbook.Close
```

```
  Application.DisplayAlerts = True
  Application.ScreenUpdating = True

End Sub
```

Listing 6.54 Tabelle exportieren, ein PDF erstellen und Tabelle als Mappe ablegen

Schalten Sie zu Beginn des Makros aus Listing 6.54 die Rückfragen von Excel temporär aus, indem Sie der Eigenschaft `DisplayAlerts` den Wert `False` zuweisen. Auch die Bildschirmaktualisierung können Sie ausschalten. Setzen Sie dazu die Eigenschaft `ScreenUpdating` auf den Wert `False`.

Prüfen Sie als Nächstes, ob der gewünschte Speicherpfad für das PDF und die Excel-Mappe überhaupt zur Verfügung steht. Dazu setzen Sie die Funktion `Dir` ein. Sollte das Zielverzeichnis noch nicht vorhanden sein, dann wenden Sie die Anweisung `MkDir` an, um es zu erstellen.

Danach wenden Sie die Methode `Copy` an, um Tabelle3 aus der Mappe herauszukopieren und in einer neuen Arbeitsmappe anzubieten.

Verwenden Sie dann die Methode `SaveAs`, um die Mappe zu speichern. Ergänzen Sie dabei mithilfe der Funktionen `Format` und `Date` den Dateinamen um einen Datumsstempel.

Mithilfe der Methode `ExportAsFixedFormat` wandeln Sie diese neue Mappe, die nur aus einer Tabelle besteht, in ein PDF um.

Schließen Sie am Ende des Makros die aktive Arbeitsmappe, indem Sie die Methode `Close` einsetzen. Vergessen Sie nicht, die beiden Schalter, die Sie zu Beginn des Makros ausgeschaltet haben, wieder einzuschalten.

6.10 Tabelleninhaltsverzeichnis erstellen

Bei der nächsten Aufgabe aus Listing 6.55 wird mithilfe eines Makros ein Inhaltsverzeichnis erstellt. Dabei wird zu Beginn der Arbeitsmappe eine neue Tabelle eingefügt. Danach werden die Namen aller in der Mappe enthaltenen Tabellen untereinander eingefügt und verlinkt.

```
Sub TabellenVerzeichnisErstellenHyperlinks()
  Dim wkbMappe As Workbook
  Dim wksBlatt As Worksheet
  Dim intTab As Integer
  Dim intZeile As Integer

   Set wkbMappe = ActiveWorkbook
```



```
  Set wksBlatt = wkbMappe.Worksheets.Add(Before:=Worksheets(1))
  intZeile = 1

   For intTab = 2 To wkbMappe.Worksheets.Count
       wksBlatt.Cells(intZeile, 1).Value = _
       Worksheets(intTab).Name

       wksBlatt.Cells(intZeile, 1).Hyperlinks.Add _
           Anchor:=wksBlatt.Cells(intZeile, 1), _
           Address:="", SubAddress:="'" & _
           Worksheets(intTab).Name & "'!A1", _
           ScreenTip:= _
           "Klicken Sie auf den Hyperlink", _
           TextToDisplay:=Worksheets(intTab).Name

       intZeile = intZeile + 1

   Next intTab

End Sub
```

Listing 6.55 Ein Tabelleninhaltsverzeichnis erstellen und mit Hyperlinks belegen

Über den Einsatz der Methode `Add` fügen Sie zu Beginn des Makros eine neue, noch leere Tabelle ein. Danach durchlaufen Sie mithilfe einer `For ... Next`-Schleife, beginnend bei der zweiten Tabelle, alle in der Mappe enthaltenen Tabellen.

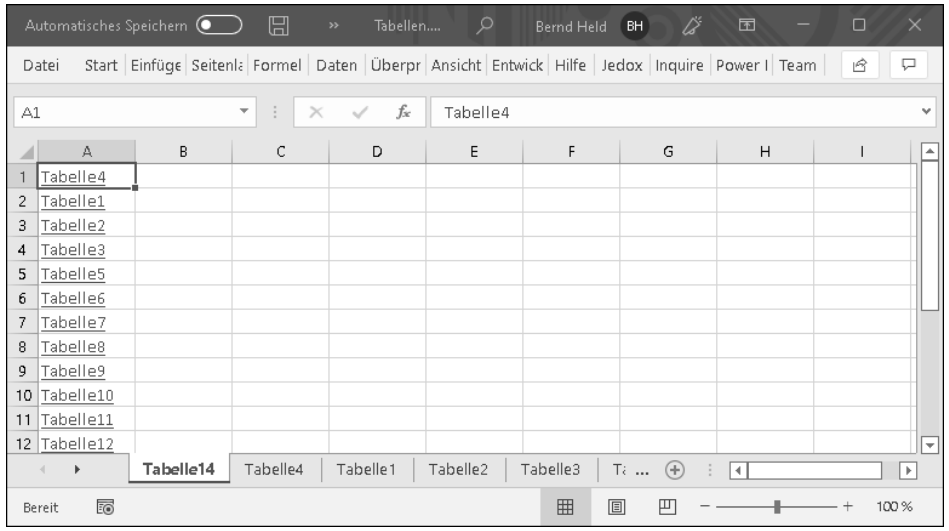


Abbildung 6.8 Das Inhaltsverzeichnis der Tabellen wurde erstellt.

Die Namen der Tabellen fügen Sie dann untereinander auf der ersten Tabelle in der Mappe ein und verlinken sie über den Einsatz der Methode `Add`. Diese Methode enthält einige Parameter: Im Parameter `SubAddress` geben Sie die Zieltabelle an, die per Klick auf den Hyperlink angesprungen werden soll. Über den Parameter `ScreenTip` definieren Sie den Text, der angezeigt werden soll, wenn über den Hyperlink gestrichen wird. Über den Parameter `TextToDisplay` definieren Sie die Beschriftung des Hyperlinks. Hierbei wird der Tabellenname angegeben.

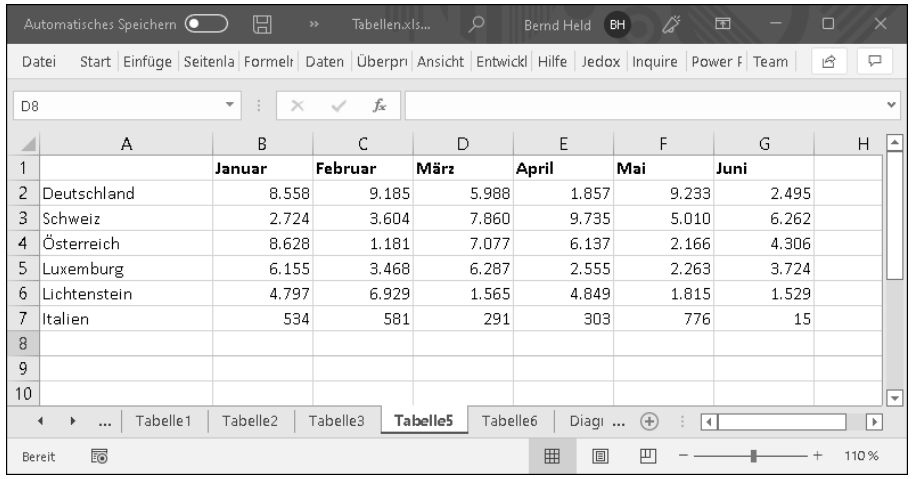
6.11 Intelligente Tabellen

In Excel haben Sie die Möglichkeit, einen Datenbestand als Tabelle (nicht zu verwechseln mit Tabellenblättern) zu kennzeichnen. Bis Excel 2003 wurde diese Funktionalität als *Liste* bezeichnet. Die schnelle Sortierung und Filterung der Daten, der Einsatz von strukturierten Verweisen und die vielfältigen Möglichkeiten der Formatierung sind nur einige der Vorteile dieses in der Praxis allerdings nur recht selten eingesetzten Konzeptes.

Hinweis

Sie finden alle folgenden Makros in der Datei *Tabellen.xlsm* im Modul `mdl_IntelligenteTab`.

Bei den folgenden Aufgaben gehen wir von der Ausgangstabelle aus Abbildung 6.9 aus.



	A	B	C	D	E	F	G	H
1		**Januar**	**Februar**	**März**	**April**	**Mai**	**Juni**	
2	Deutschland	8.558	9.185	5.988	1.857	9.233	2.495	
3	Schweiz	2.724	3.604	7.860	9.735	5.010	6.262	
4	Österreich	8.628	1.181	7.077	6.137	2.166	4.306	
5	Luxemburg	6.155	3.468	6.287	2.555	2.263	3.724	
6	Lichtenstein	4.797	6.929	1.565	4.849	1.815	1.529	
7	Italien	534	581	291	303	776	15	
8								
9								
10								

Abbildung 6.9 Die Ausgangssituation für die intelligente Tabelle

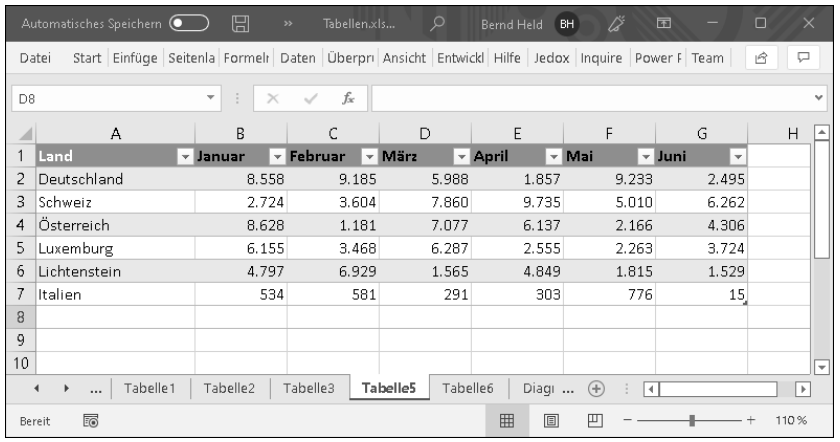
6.11.1 Tabelle umwandeln

Im ersten Schritt muss aus der »normalen« Tabelle eine sogenannte *intelligente Tabelle* erstellt werden.

```
Sub TabelleDefinieren()
  Dim LO As ListObject
  Dim lngZeileMax As Long

  With Tabelle5

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

      Set LO = .ListObjects.Add(SourceType:=xlSrcRange, _
          Source:=Range("A1:F" & lngZeileMax), xlListObjecthasheaders:=xlYes)
            LO.Name = "Umsätze"

  End With

End Sub
```

Listing 6.56 Eine intelligente Tabelle erstellen

Über die Methode `Add` fügen Sie ein `ListObject` in das Tabellenblatt ein. Mit dem (optionalen) Parameter `Source` legen Sie fest, dass es sich bei der Datenquelle um einen Bereich in einer Arbeitsmappe handelt. Dem Parameter `Source` weisen Sie den Bereich zu, den Sie als Tabelle ausweisen möchten. Über den Parameter `xlListObjecthasheaders:=xlYes` teilen Sie Excel mit, dass es sich bei der ersten Zeile des festgelegten Bereichs um eine Überschrift handelt. Mit der Eigenschaft `Name` weisen Sie der Tabelle den Namen UMSÄTZE zu.



	A	B	C	D	E	F	G	H
1	**Land**	**Januar**	**Februar**	**März**	**April**	**Mai**	**Juni**	
2	Deutschland	8.558	9.185	5.988	1.857	9.233	2.495	
3	Schweiz	2.724	3.604	7.860	9.735	5.010	6.262	
4	Österreich	8.628	1.181	7.077	6.137	2.166	4.306	
5	Luxemburg	6.155	3.468	6.287	2.555	2.263	3.724	
6	Lichtenstein	4.797	6.929	1.565	4.849	1.815	1.529	
7	Italien	534	581	291	303	776	15	
8								
9								
10								

Abbildung 6.10 Die neue Tabelle in ansprechendem Design

6.11.2 Tabelle um eine Spalte ergänzen

Die Tabelle soll um eine Spalte 1. Halbjahr ergänzt werden, in der die Umsätze in den einzelnen Ländern über die ersten 6 Monate des Jahres summiert werden.

```
Sub SpalteAnTabelleHinzufügen()
  Dim LO As ListObject
  Dim LoSpalte As ListColumn

  Set LO = Tabelle5.ListObjects("Umsätze")

  Set LoSpalte = LO.ListColumns.Add

  With LoSpalte
    .Name = "1. Halbjahr"
    .DataBodyRange.Cells(1).FormulaR1C1 = "=SUM(Umsätze[@[Januar]:[Juni]])"
  End With

End Sub
```

Listing 6.57 Tabelle um eine Spalte »1. Halbjahr« ergänzen

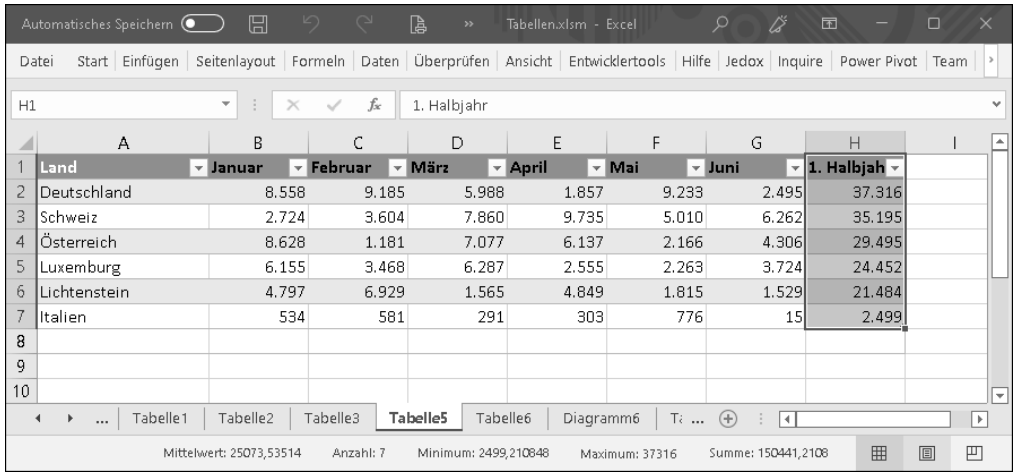


Abbildung 6.11 Eine Summenspalte wurde am rechten Rand der Tabelle eingefügt.

Mit der Methode `Add` erstellen Sie ein neues `ListColumns`-Objekt. Diesem Objekt, das die neue Spalte repräsentiert, weisen Sie über die Eigenschaft `Name` den Namen 1. Halbjahr zu. Dem Datenbereich der neuen Spalte fügen Sie über die Eigenschaft `DatabodyRange` die Formel zur Summierung der einzelnen Monate zu.

Da der Methode `Add` kein Parameter zur Position zugewiesen wurde, wird die neue Spalte am rechten Ende der Tabelle hinzugefügt.

6.11.3 Tabelle um eine Zeile ergänzen

Die Tabelle soll noch um eine neue Zeile mit einigen »zufälligen« Umsätzen aus Polen ergänzt werden. Greifen Sie hierzu auf Listing 6.58 zurück:

```
Sub ZeileAnTabelleHinzufügen()
 Dim LO As ListObject
 Dim loZeile As ListRow
 Dim lngZeileFrei As Long
 Dim lngZ As Long

 With Tabelle5
  lngZeileFrei = .Cells(.Rows.Count, 1).End(xlUp).Row
  Set LO = .ListObjects("Umsätze")

  Set loZeile = LO.ListRows.Add(Position:=lngZeileFrei)

    With loZeile
      .Range(1).Value = "Polen"

      For intz = 2 To 7
       .Range(intz).Value = Int(1000) * Rnd + 1
      Next intz

    End With

 End With

End Sub
```

Listing 6.58 Tabelle um die Umsätze aus Italien ergänzen

Wiederum mit der Methode `Add` erstellen Sie ein `ListRows`-Objekt. Die einzelnen Werte der neuen Zeilen legen Sie über die Eigenschaft `Range` mittels einer Schleife fest. Dabei wird die Zahl 1.000 genommen und mit einem Wert zwischen 0 und 1 multipliziert.

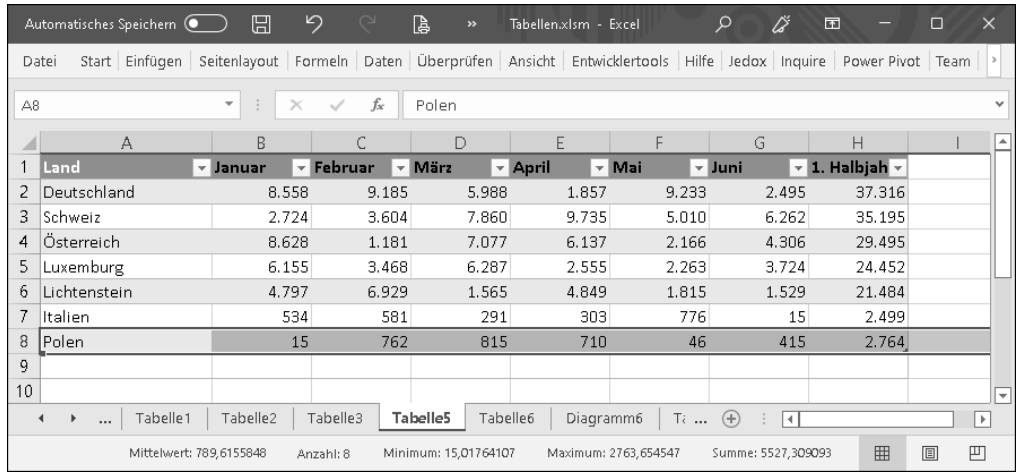


Abbildung 6.12 Die neue Zeile wurde unten angehängt.

6.11.4 Tabelle filtern

Ein Vorteil von Tabellen ist der bereits eingefügte AutoFilter. Selbstverständlich können Sie diesen auch über ein Makro ansteuern. In Listing 6.59 wird nach den Umsätzen von Österreich und Deutschland gefiltert.

```
Sub TabelleFiltern()
 Dim LO As ListObject

  Set LO = Tabelle5.ListObjects("Umsätze")

   LO.Range.AutoFilter Field:=1, Criteria1:="Österreich", _
                       Operator:=xlOr, _
                       Field:=1, Criteria2:="Deutschland"

End Sub
```

Listing 6.59 Tabellen mit dem Autofilter filtern

Mithilfe der Methode `AutoFilter` filtern Sie eine intelligente Tabelle. Dabei übergeben Sie die Spaltennummer und das Filterkriterium.

Wenn weitere Länder zur Filterung hinzukommen, dann müssen Sie aus den Ländern ein Datenfeld wie in Listing 6.60 gezeigt bilden.

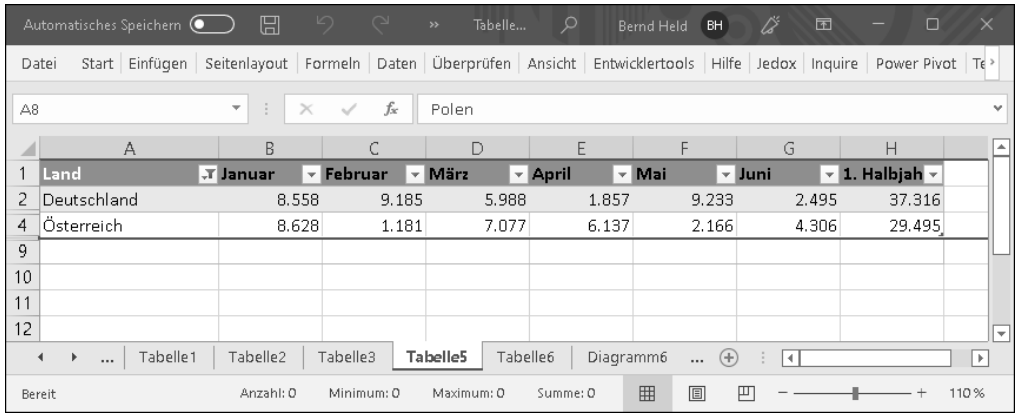


Abbildung 6.13 Die beiden Länder wurden ausgefiltert.

```
Sub TabellenlisteAutofilter()
 Dim LO As ListObject

 Set LO = Tabelle5.ListObjects("Umsätze")

  LO.Range.AutoFilter Field:=1, Criteria1:= _
  Array("Österreich", "Schweiz", "Deutschland"), Operator:=xlFilterValues

End Sub
```

Listing 6.60 Mehrere Einträge als Filterkriterium über ein Datenfeld einsteuern

Verwenden Sie ein Array, um mehrere Länder als Filterkriterium einzustellen.

Hinweis

Soll das AutoFilter-Symbol in der intelligenten Tabelle nicht angezeigt werden, dann starten Sie das Makro aus Listing 6.61.

```
Sub AutofilterAusschalten()
    Tabelle5.ListObjects("Umsätze").ShowAutoFilter = True
End Sub
```

Listing 6.61 Die Filterung in einer intelligenten Tabelle aufheben

Um die Filtersymbole aus- und wieder einzublenden, verwenden Sie die Eigenschaft `ShowAutoFilter`.

6.11.5 Tabellen sortieren

In der folgenden Prozedur wird die Tabelle nach den Umsätzen im ersten Halbjahr sortiert:

```
Sub TabelleSortieren()
 Dim LO As ListObject

  Set LO = Tabelle5.ListObjects("Umsätze")

  LO.Sort.SortFields.Clear

  LO.Sort.SortFields.Add _
  Range("Umsätze[[#All], [1. Halbjahr]]"), _
  SortOn:=xlSortOnValues, Order:=xlDescending

  With LO.Sort
   .Header = xlYes
   .Orientation = xlTopToBottom
   .Apply
  End With

End Sub
```

Listing 6.62 Sortierung nach den Umsätzen im ersten Halbjahr

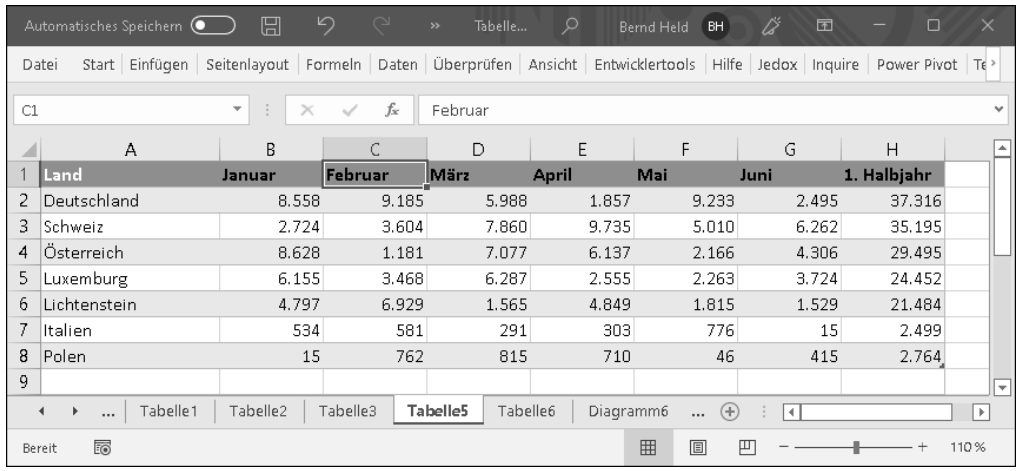


Abbildung 6.14 Die Umsätze liegen jetzt absteigend vor.

Diese Möglichkeit zur Sortierung existiert ab Excel 2007. Dabei werden eventuell bereits vorher eingestellte Sortieroptionen über die Methode `Clear` entfernt. Danach

fügen Sie eine neue Sortierung mithilfe der Methode `Add` hinzu. Im Parameter `SortOn` geben Sie an, dass Sie die Liste nach Werten sortieren möchten. Der Parameter `Order` bestimmt, wie Sie die Liste sortieren möchten. Zur Verfügung stehen dabei die Konstanten `xlAscending` und `xlDescending`.

Mithilfe der Eigenschaft `Header`, die Sie mit der Konstanten `xlYes` füttern, legen Sie fest, dass die Tabelle eine Überschrift enthält, die bei der Sortierung nicht mit sortiert werden soll. Die Eigenschaft `Orientation` mit der Konstanten `xlToBottom` sagt aus, dass Sie Zeilen sortieren möchten. Mithilfe der Methode `Apply` wird die Sortierung letztendlich ausgeführt.

6.11.6 Tabelle um Ergebniszeile erweitern

Wie Sie eine Ergebniszeile hinzufügen, die die Umsätze im ersten Halbjahr addiert, sehen Sie in Listing 6.63:

```
Sub ErgebnisZeileAnTabelleHinzufügen()
  Dim LO As ListObject

  Set LO = Tabelle5.ListObjects("Umsätze")
  LO.ShowTotals = True

End Sub
```

Listing 6.63 Hinzufügen einer Ergebniszeile

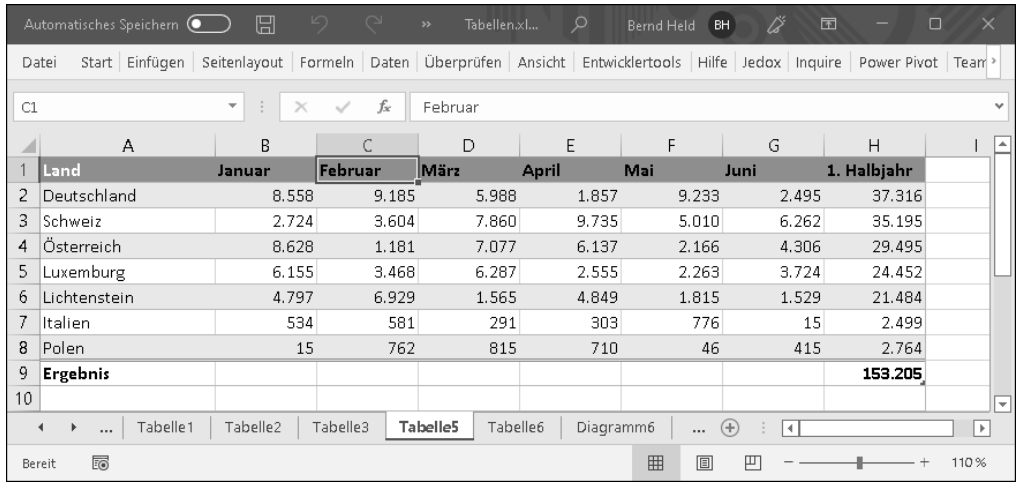


Abbildung 6.15 Eine Ergebniszeile wurde am unteren Rand der Tabelle hinzugefügt.

Mithilfe der Eigenschaft `ShowTotals` fügen Sie der intelligenten Tabelle eine Ergebniszeile hinzu.

6.11.7 Tabelle entfernen

Eine Tabelle können Sie auch leicht wieder entfernen:

```
Sub TabelleEntfernen()

  Tabelle5.ListObjects("Umsätze").Unlist

End Sub
```

Listing 6.64 Tabelle entfernen

Über die Methode `Unlist` nehmen Sie die »Intelligenz« der Tabelle wieder weg. Die eingestellten Formatierungen bleiben jedoch erhalten. Die Formeln werden dabei etwas umgestellt.

6.12 Pivot-Tabellen erstellen

In Excel haben Sie die Möglichkeit, sehr schnell aussagekräftige Berichte zu erstellen. Die wohl beste und sicherste Methode, in Excel Daten auszuwerten und aufzubereiten, ist die Anwendung von *Pivot-Tabellen*. Der Begriff »Pivot« kommt aus dem Französischen und bedeutet »Dreh- und Angelpunkt«. Das sagt schon einiges über die Funktion aus: Sie können es drehen und wenden, wie Sie wollen – Sie werden immer die richtigen Ergebnisse bekommen.

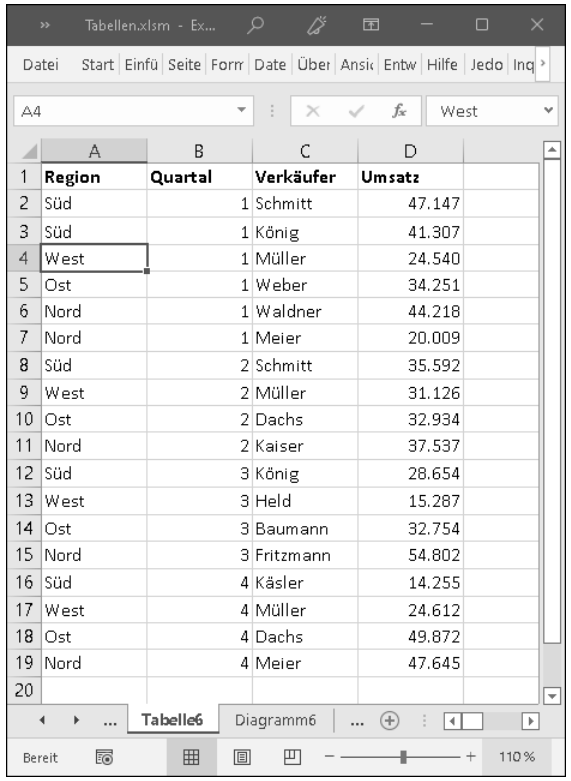
> **Hinweis**
>
> Sie finden alle folgenden Makros in der Datei *Tabellen.xlsm* im Modul `mdl_Pivot`.

Bei der Arbeit mit Pivot-Tabellen müssen Sie zwei Punkte beachten:

- Ist der Pivot-Tabellenbericht aktualisiert?
- Greift die Pivot-Tabelle wirklich auf den gewünschten Datenbereich zu? (Das heißt: Wurde der Datenbereich nachträglich vergrößert?)

Beide Fragen werde ich in diesem Kapitel noch beantworten.

Zuerst werden Sie über ein VBA-Makro einen Pivot-Tabellenbericht erstellen. Die Ausgangstabelle sehen Sie in Abbildung 6.16.



	A	B	C	D
1	**Region**	**Quartal**	**Verkäufer**	**Umsatz**
2	Süd	1	Schmitt	47.147
3	Süd	1	König	41.307
4	West	1	Müller	24.540
5	Ost	1	Weber	34.251
6	Nord	1	Waldner	44.218
7	Nord	1	Meier	20.009
8	Süd	2	Schmitt	35.592
9	West	2	Müller	31.126
10	Ost	2	Dachs	32.934
11	Nord	2	Kaiser	37.537
12	Süd	3	König	28.654
13	West	3	Held	15.287
14	Ost	3	Baumann	32.754
15	Nord	3	Fritzmann	54.802
16	Süd	4	Käsler	14.255
17	West	4	Müller	24.612
18	Ost	4	Dachs	49.872
19	Nord	4	Meier	47.645
20				

Abbildung 6.16 Eine Verkäufer-Umsatzliste pro Quartal und Region

In einem Pivot-Tabellenbericht soll nun dargestellt werden, welche Vorgesetzten welche Mitarbeiter an welchem Standort haben. Das Makro für diese Aufgabe sehen Sie in Listing 6.65:

```
Sub PivotErstellenUndPositionieren()
  Dim pc As PivotCache
  Dim pv As PivotTable
  Dim QuellBereich As Range
  Dim ZielBereich As Range

  With Tabelle6

    Set QuellBereich = .Range("A1").CurrentRegion
    Set ZielBereich = .Range("F1")

    Set pc = ActiveWorkbook.PivotCaches.Add _
    (SourceType:=xlDatabase, SourceData:=QuellBereich)
```

```
Set pv = pc.CreatePivotTable(TableDestination:=ZielBereich, _
         TableName:="PivotTable1")

With pv.PivotFields("Quartal")
   .Orientation = xlPageField
   .Position = 1
End With

With pv.PivotFields("Region")
   .Orientation = xlRowField
   .Position = 1
End With


With pv.PivotFields("Verkäufer")
   .Orientation = xlRowField
   .Position = 2
End With

With pv.PivotFields("Umsatz")
   .Orientation = xlDataField
   .Position = 1
   End With

 End With

End Sub
```

Listing 6.65 Einen Pivot-Tabellenbericht erstellen

Im ersten Schritt geben Sie die Bereiche bekannt, für die diese Auswertung erfolgen soll. Dabei verwenden Sie die Eigenschaft `CurrentRegion`, um alle Zellen, die um Zelle A1 herum liegen und gefüllt sind, auszuwählen.

Danach markieren Sie den Quellbereich und wenden die Methode `Add` an, um eine neue Pivot-Tabelle zu erstellen. Die Methode `Add` hat folgende Syntax:

```
PivotCaches.Add(SourceType, SourceData)
```

- Im Argument `SourceType` geben Sie die Quelle der Daten in dem Bericht an. Zulässig ist eine der in Tabelle 6.3 aufgeführten Konstanten.
- Beim Argument `SourceData` legen Sie den Datenbereich für den neuen Bericht fest. Hierfür haben Sie vorher den verwendeten Bereich auf dem Tabellenblatt in der Variablen `Bereich` definiert, die Sie hier nun als Argument angeben.

Konstante	Beschreibung
`xlConsolidation`	mehrere Konsolidierungsbereiche
`xlDatabase`	Microsoft-Excel-Datenbank oder -Liste (Standard)
`xlExternal`	externe Datenquelle
`xlPivotTable`	gleiche Quelle wie ein anderer `PivotTable`-Bericht

Tabelle 6.3 Die möglichen Datenquellen-Konstanten bei Pivot-Tabellenberichten

Beginnen Sie jetzt mit der Anordnung Ihrer Datenfelder, und nutzen Sie dazu das Auflistungsobjekt `PivotFields`. In diesem Auflistungsobjekt stehen alle verfügbaren Feldnamen der Pivot-Tabelle, die den Spaltenüberschriften des Quellbereiches entsprechen.

Hinweis

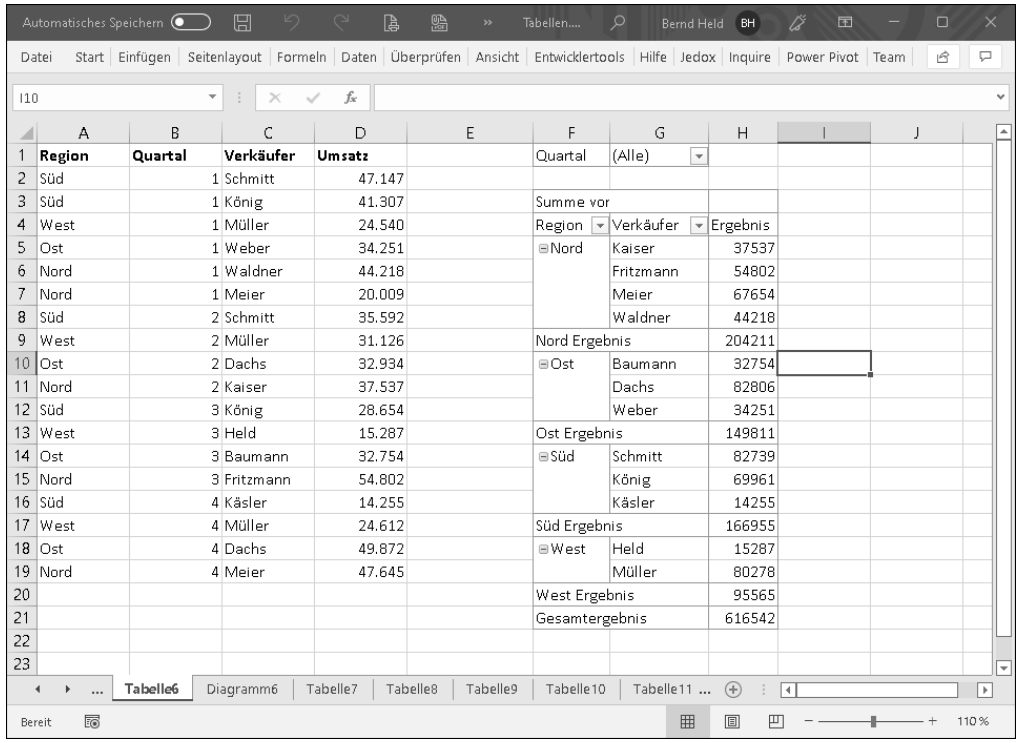
Wenn Sie nicht sicherstellen können, dass die Spaltenüberschriften Ihres Datenbereiches konstant bleiben, können Sie die einzelnen Feldnamen auch über einen Index ansprechen. Wenn die Spaltenüberschriften jedoch gleich bleiben, verwenden Sie besser die Feldnamen für die bessere Lesbarkeit des Makros.

Mit der Eigenschaft `Orientation` legen Sie die Position des Feldnamens in der Pivot-Tabelle fest. Tabelle 6.4 führt die Möglichkeiten auf, die Sie dabei haben.

Konstante	Beschreibung
`xlColumnField`	Im Spaltenbereich der Pivot-Tabelle wird der Standort ausgegeben.
`xlDataField`	Im Datenbereich der Pivot-Tabelle wird die Anzahl der Mitarbeiter gezählt.
`xlHidden`	Mit dieser Konstanten blenden Sie einzelne Datenfelder aus.
`xlPageField`	Im Seitenfeld der Pivot-Tabelle wird der gewünschte Vorgesetzte eingestellt.
`xlRowField`	Im Zeilenbereich werden die Namen der Mitarbeiter aufgelistet.

Tabelle 6.4 Die möglichen Ausrichtungskonstanten bei Pivot-Tabellenberichten

Mithilfe der Eigenschaft `Position` legen Sie die Reihenfolge fest, die in unserem Beispiel für die Zeilenbereiche REGION und VERKÄUFER getroffen werden muss.



	A	B	C	D	E	F	G	H	I	J
1	**Region**	**Quartal**	**Verkäufer**	**Umsatz**		Quartal	(Alle)			
2	Süd	1	Schmitt	47.147						
3	Süd	1	König	41.307		Summe vor				
4	West	1	Müller	24.540		Region	Verkäufer	Ergebnis		
5	Ost	1	Weber	34.251		⊟Nord	Kaiser	37537		
6	Nord	1	Waldner	44.218			Fritzmann	54802		
7	Nord	1	Meier	20.009			Meier	67654		
8	Süd	2	Schmitt	35.592			Waldner	44218		
9	West	2	Müller	31.126		Nord Ergebnis		204211		
10	Ost	2	Dachs	32.934		⊟Ost	Baumann	32754		
11	Nord	2	Kaiser	37.537			Dachs	82806		
12	Süd	3	König	28.654			Weber	34251		
13	West	3	Held	15.287		Ost Ergebnis		149811		
14	Ost	3	Baumann	32.754		⊟Süd	Schmitt	82739		
15	Nord	3	Fritzmann	54.802			König	69961		
16	Süd	4	Käsler	14.255			Käsler	14255		
17	West	4	Müller	24.612		Süd Ergebnis		166955		
18	Ost	4	Dachs	49.872		⊟West	Held	15287		
19	Nord	4	Meier	47.645			Müller	80278		
20						West Ergebnis		95565		
21						Gesamtergebnis		616542		
22										
23										

Abbildung 6.17 Die neue Pivot-Tabelle nach Region konsolidiert

6.12.1 Pivot-Tabellen aktualisieren

Bei Pivot-Tabellenberichten dürfen Sie nach einer Änderung der Quelldaten nicht vergessen, die Pivot-Tabelle auf den neuesten Stand zu bringen. Dieser Vorgang geschieht in Excel nicht automatisch; dafür sind Sie verantwortlich.

6.12.2 Eine einzelne Pivot-Tabelle aktualisieren

Wenn Sie ganz gezielt eine einzelne Pivot-Tabelle in einer Arbeitsmappe aktualisieren möchten, setzen Sie das Makro aus Listing 6.66 ein:

```
Sub EinzelnePivotTabelleAufBlattAktualisieren()

  On Error GoTo fehler
  Tabelle6.PivotTables(1).RefreshTable
  Exit Sub

fehler:
  MsgBox "Es konnte keine Pivot-Tabelle gefunden werden!"

End Sub
```

Listing 6.66 Einzelne Pivot-Tabelle auf Tabellenblatt aktualisieren

Auf dem ersten Tabellenblatt in der aktiven Arbeitsmappe wird die Pivot-Tabelle mit dem Index 1 mithilfe der Methode `RefreshTable` auf den aktuellen Stand gebracht. Die Anweisung `On Error` ist eine reine Sicherheitsmaßnahme, die verhindern soll, dass das Makro abstürzt, wenn die richtige Pivot-Tabelle nicht gefunden werden kann.

6.12.3 Mehrere Pivot-Tabellen auf einem Tabellenblatt aktualisieren

Oft werden auch mehrere Pivot-Tabellen auf einem einzigen Tabellenblatt geführt. Dabei können Sie alle Pivot-Tabellen auf einmal aktualisieren, indem Sie das Makro aus Listing 6.67 einsetzen:

```
Sub MehrerePivotTabellenAufBlattAktualisieren()
  Dim pt As PivotTable

   For Each pt In Tabelle6.PivotTables
    pt.RefreshTable
  Next pt

End Sub
```

Listing 6.67 Mehrere Pivot-Tabellen auf einem Tabellenblatt aktualisieren

Definieren Sie eine Objektvariable vom Typ `PivotTable`, und setzen Sie eine Schleife des Typs `For Each ... Next` auf. Innerhalb der Schleife wenden Sie die Methode `RefreshTable` an, um die einzelnen Pivot-Tabellenberichte zu aktualisieren.

6.12.4 Alle Pivot-Tabellen in Arbeitsmappe aktualisieren

Haben Sie in einer Arbeitsmappe mehrere Pivot-Tabellenberichte auf unterschiedlichen Tabellenblättern erstellt und möchten Sie sie nun automatisch aktualisieren, dann setzen Sie das Makro aus Listing 6.68 ein:

```
Sub AllePivotTabellenInArbeitsmappeAktualisieren()
  Dim wksBlatt As Worksheet
  Dim pt As PivotTable
```

```
  For Each wksBlatt In ActiveWorkbook.Worksheets
    For Each pt In wksBlatt.PivotTables
      pt.RefreshTable
    Next pt
  Next wksBlatt

End Sub
```

Listing 6.68 Alle Pivot-Tabellen in der aktiven Arbeitsmappe aktualisieren

Die Aktualisierung aller Pivot-Tabellenberichte Ihrer aktiven Arbeitsmappe erreichen Sie über zwei Schleifen. In der ersten Schleife werden alle Tabellenblätter durchlaufen. Mit der zweiten Schleife werden innerhalb eines durchlaufenen Blattes alle Pivot-Tabellenberichte aktualisiert.

6.12.5 Pivot-Tabellen dynamisch erweitern

Die zweite Gefahr neben derjenigen, die Aktualisierung der Pivot-Tabelle zu vergessen, ist, dass die Anwenderin den Datenbereich erweitert und versäumt, diese Erweiterung seiner Pivot-Tabelle mitzuteilen. Das manuelle Einstellen der Pivot-Tabelle auf den neuen Bereich ist recht mühselig, wenn Sie mehrere Pivot-Tabellen nacheinander auf den neuesten Stand bringen müssen. In beiden Fällen (fehlende Aktualisierung oder falscher Quellbereich) ist die Auswirkung gleich: Die Pivot-Tabelle liefert nicht die richtigen Ergebnisse.

Die Änderung des Quellbereichs, der die Daten für die Pivot-Tabelle enthält, hat zur Folge, dass auch der Pivot-Tabelle diese Änderung mitgeteilt werden muss. Um diese Aufgabe zu erledigen, benennen Sie Ihren Datenbereich vorher, und erweitern Sie die Pivot-Tabelle mithilfe des Makros aus Listing 6.69:

```
Sub PivotTabellenDatenbereichErweitern()
  Dim pt As PivotTable
  Dim rngBereich As Range

  Set rngBereich = Tabelle6.Range("A1").CurrentRegion

  ActiveWorkbook.Names.Add Name:="PivotBereich", _
  RefersTo:=rngBereich, Visible:=True

  For Each pt In Tabelle6.PivotTables

    pt.PivotTableWizard SourceType:=xlDatabase, SourceData:="PivotBereich"
    pt.RefreshTable
```



```
  Next pt

End Sub
```

Listing 6.69 Dynamische Erweiterung einer Pivot-Tabelle

Ermitteln Sie zuerst einmal den verwendeten Bereich auf Ihrem Tabellenblatt PERSONAL, und speichern Sie diesen Bereich in der Objektvariablen `Bereich`. Benennen Sie danach den ermittelten Bereich, geben Sie ihm den Namen PIVOTBEREICH, und markieren Sie ihn im Anschluss.

Mithilfe einer Schleife des Typs `For Each ... Next` greifen Sie auf alle Pivot-Tabellenberichte auf dem ersten Blatt Ihrer Arbeitsmappe zu, erweitern den Datenbereich und aktualisieren die Pivot-Tabellenberichte. Der neue Datensatz wird am Ende der Pivot-Tabelle angehängt. Die Pivot-Tabelle muss jetzt neu sortiert werden.

6.12.6 Pivot-Tabellen formatieren

Seit der Excel-Version 2000 haben Sie die Möglichkeit, die AutoFormate auch für Pivot-Tabellenberichte einzusetzen. Dabei haben Sie die Auswahl aus über vierzig verschiedenen AutoFormaten.

Eine komplette Liste der AutoFormate in Excel finden Sie in der VBA-Onlinehilfe unter dem Stichwort »AutoFormat«.

Um z. B. die Pivot-Tabelle mit der Mitarbeiterauswertung zu formatieren, können Sie das Makro aus Listing 6.70 einsetzen:

```
Sub PivotFormatEinstellen()
 Dim Pivot1 As PivotTable

  Set Pivot1 = Tabelle6.PivotTables(1)
  Pivot1.TableRange1.AutoFormat Format:=xlClassic3

End Sub
```

Listing 6.70 AutoFormate auf Pivot-Tabellen anwenden

Über die Eigenschaft `TableRange1` wählen Sie einen Pivot-Tabellenbericht aus. Dabei sind allerdings keine Seitenfelder (z. B. VORGESETZTER) inbegriffen. Möchten Sie die Formatierung jedoch für die komplette Pivot-Tabelle, also inklusive der Seitenfelder, anwenden, verwenden Sie die Eigenschaft `TableRange2`. Auf den so bekanntgemachten Bereich wenden Sie die Methode `AutoFormat` an, die den markierten Bereich mit einem vordefinierten Format belegt.

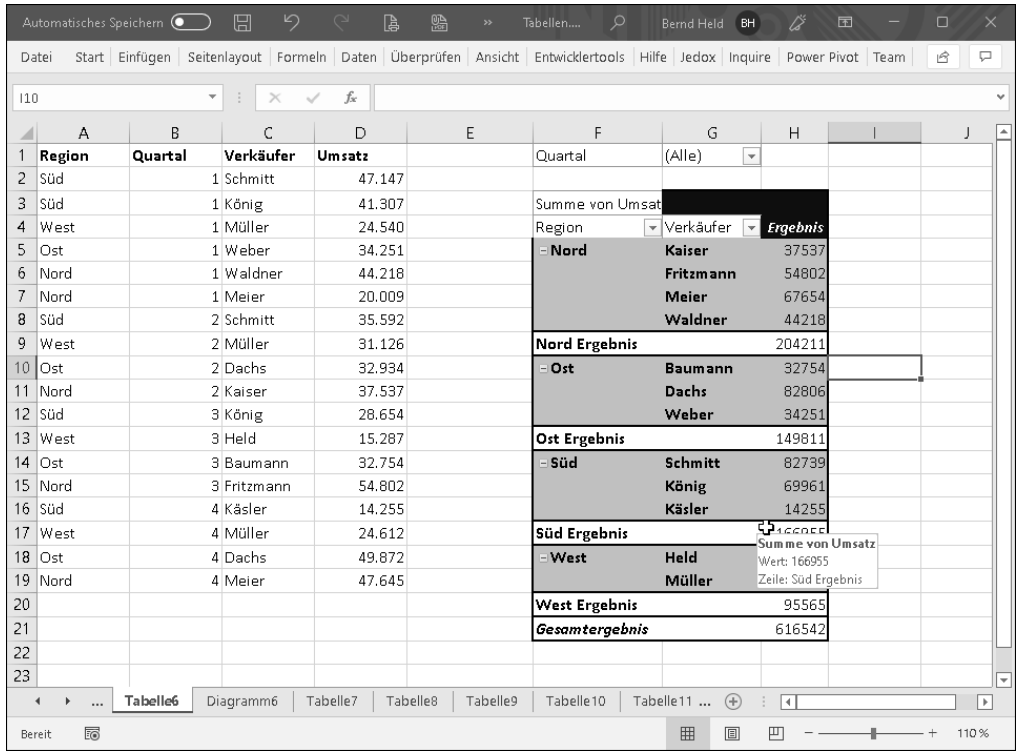


Abbildung 6.18 Ein AutoFormat für eine Pivot-Tabelle einstellen

6.12.7 Slicer einfügen und bedienen

Ab der Version Excel 2010 gibt es bei Pivot-Tabellen die Möglichkeit, einen sogenannten Slicer hinzuzufügen. Damit lassen sich Pivot-Tabellen besser steuern und flexibler filtern.

```
Sub SlicerObjectEinfügen()
 Dim scc As SlicerCaches
 Dim scs As Slicers
 Dim sc As Slicer

  Call SlicerEntfernen

  Set scc = ThisWorkbook.SlicerCaches
  Set scs = scc.Add(Tabelle6.PivotTables(1), "Region").Slicers
  Set sc = scs.Add(Tabelle6, , "Region 1", "Region", 100, 300, 150, 200)

  With ActiveWorkbook.SlicerCaches(1).Slicers(1)
   .Caption = "Regionsfilter"
```



```
   .Width = 100
  End With

End Sub

Sub SlicerEntfernen()
 Dim lngSlicer As Long

 For lngSlicer = ThisWorkbook.SlicerCaches.Count To 1 Step -1
    ThisWorkbook.SlicerCaches(lngSlicer).Delete
 Next lngSlicer

End Sub
```

Listing 6.71 Einen Slicer löschen und einfügen

Bevor Sie neue Slicer einfügen, sollten Sie die alten über die Methode `Delete` entfernen.

Wenden Sie die Methode `Add` an, um der Pivot-Tabelle einen Slicer hinzuzufügen. Dabei geben Sie bekannt, auf welches Datenfeld sich der Slicer bezieht. Des Weiteren können Sie den Namen des Slicers und seine Beschriftung festlegen. Die Positionsangaben stehen für oben, links sowie für die Breite und Höhe des Slicers.

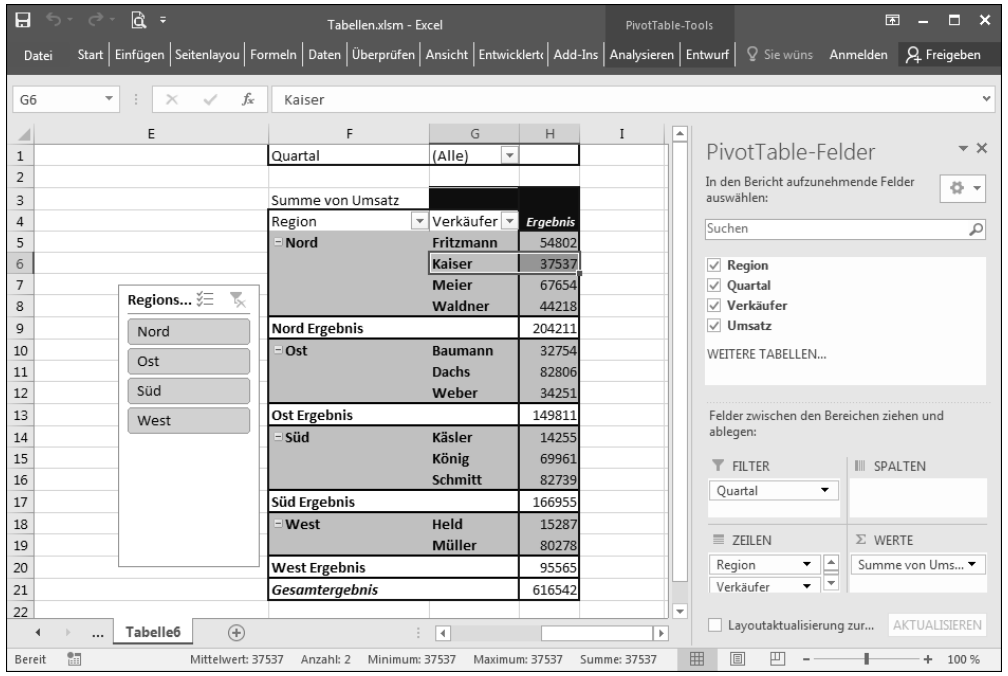


Abbildung 6.19 Der Slicer für die Region wurde eingefügt.

Mithilfe der Eigenschaft `Selected` drücken Sie einzelne Schaltflächen im Slicer. Sehen Sie sich dazu das folgende Makro an.

```
Sub DatenschnittBedienen()
  Dim scc As SlicerCaches

  Set scc = ThisWorkbook.SlicerCaches

  scc(1).SlicerItems("Nord").Selected = False
  scc(1).SlicerItems("Süd").Selected = True
  scc(1).SlicerItems("Ost").Selected = True
  scc(1).SlicerItems("West").Selected = False

End Sub
```

Listing 6.72 Einen Slicer per Makro bedienen

6.13 Diagramme programmieren

Bei der Erstellung von Diagrammen haben Sie die Auswahl aus mehreren Diagrammtypen.

Hinweis

Sie finden alle folgenden Makros in der Datei *Tabellen.xlsm* im Modul `mdl_Diagramm`.

Für den richtigen Diagrammtyp ist die Eigenschaft `ChartType` verantwortlich. In Tabelle 6.5 sehen Sie exemplarisch eine kleine Auswahl an möglichen Diagrammtypen, die über eine `xlChartType`-Konstante identifiziert werden.

Diagrammtyp	Konstante
Säulendiagramm (gruppiert)	`xlColumnClustered`
Säulendiagramm (gestapelt)	`xlColumnStacked`
Säulendiagramm (3D-Darstellung)	`xl3DColumn`
Balkendiagramm (gruppiert)	`xlBarClustered`
Balkendiagramm (gestapelt)	`xlBarStacked`
Liniendiagramm	`xlLine`
Kreisdiagramm	`xlPie`

Tabelle 6.5 Die wichtigsten Diagrammtypen

Diagrammtyp	Konstante
Punktdiagramm	`xlXYScatter`
Blasendiagramm	`xlBubble`
Flächendiagramm	`xlArea`
Ringdiagramm	`xlDoughnut`
Netzdiagramm	`xlRadar`
Oberflächendiagramm	`xlSurface`
Kursdiagramm	`xlStockHLC`

Tabelle 6.5 Die wichtigsten Diagrammtypen (Forts.)

Neben den in der Tabelle abgebildeten Diagrammtypen gibt es Zylinder-, Kegel- und Pyramidendiagramme sowie zu jedem Diagramm zahlreiche Untertypen. Eine komplette Liste der vorhandenen Diagramme in Excel bekommen Sie in der Onlinehilfe unter dem Stichwort »ChartType«.

6.13.1 Umsätze in einem Säulendiagramm darstellen

Das Säulendiagramm ist das Standarddiagramm in Excel. Sie können ein Säulendiagramm ganz schnell in Excel erstellen, indem Sie Ihren Datenbereich markieren und einfach die Taste F11 drücken. An diesem Standarddiagramm fehlen eigentlich nur noch ein Titel und eventuell ein paar Datenbeschriftungen.

Erzeugen Sie ein Säulendiagramm, dessen Titel sich vom Tabellennamen ableitet. Das Makro für diese Aufgabe sehen Sie in Listing 6.73:

```
Sub DiagrammErstellen()
  Dim chObj As ChartObject
  Dim lngZeileMax As Long

   With Tabelle7
    lngZeileMax = .Cells(.Rows.Count, 1).End(xlUp).Row
    Set chObj = .ChartObjects.Add(100, 100, 100, 100)
    With chObj.Chart
     .ChartType = xl3DColumnClustered
     .SetSourceData Source:= _
     Tabelle7.Range("Tabelle7!$A$1:$B$" & lngZeileMax)
     .HasTitle = True
     .ChartTitle.Text = Tabelle7.Range("B1").Value
```

```
    .Location Where:=xlLocationAsNewSheet
   End With
  End With
End Sub
```

Listing 6.73 Umsätze in einem Säulendiagramm ausgeben – Variante 1

Deklarieren Sie eine Objektvariable vom Typ `ChartObject`. Damit ist ein Diagrammobjekt gemeint, das sich auf einer Tabelle befindet. Danach fügen Sie über die Methode `Add` ein neues `ChartObject` der Auflistung `ChartObjects` hinzu. Dieses Auflistungsobjekt enthält bereits alle eingebetteten Diagramme der angegebenen Tabelle. Bei dieser Methode können Sie in Punkt genau angeben, wo in der Tabelle das Diagramm eingebettet und wie groß es werden soll.

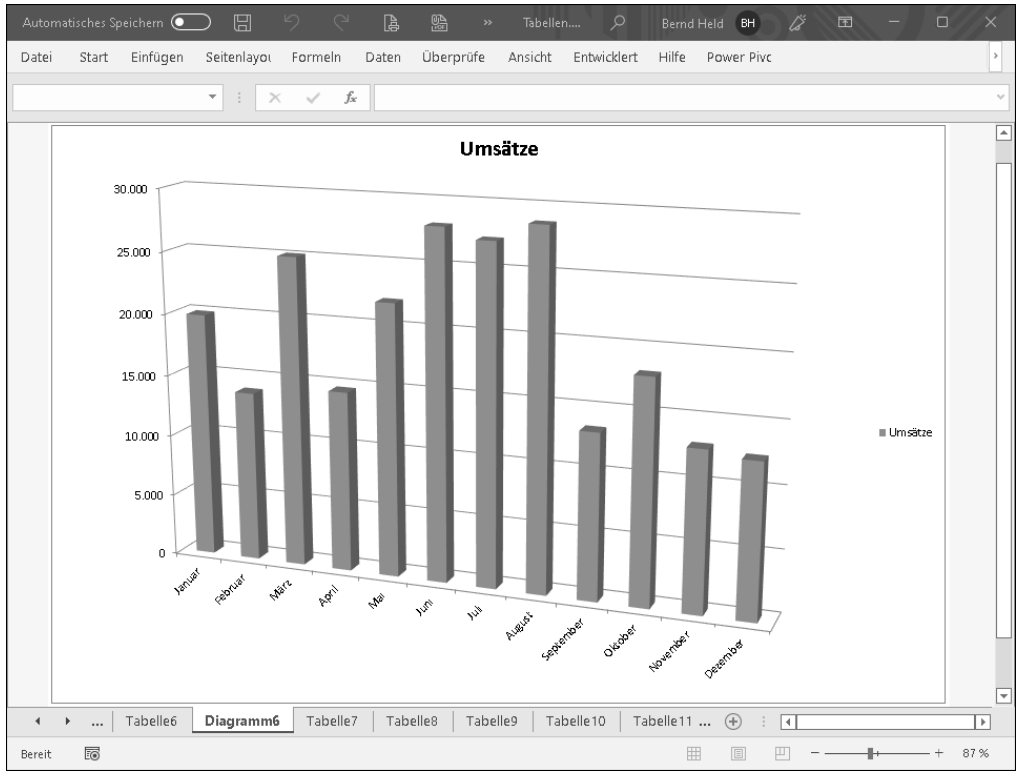


Abbildung 6.20 Das 3D-Säulendiagramm mit den monatlichen Umsätzen

Danach referenzieren Sie auf das »innere« Diagramm, das im eingebetteten Diagrammobjekt liegt. Sie bestimmen über die Eigenschaft `ChartType` die gewünschte Diagrammform und geben über die Eigenschaft `SetSourceData` bekannt, auf welchen

Bereich sich das Diagramm bezieht. Mithilfe der Eigenschaft `HasTitle` bestimmen Sie, dass das Diagramm einen Titel erhalten soll. Diesen Titel erfassen Sie mithilfe der Eigenschaft `ChartTitle`, die Sie über die Eigenschaft `Text` befüllen. Als verknüpfte Zelle wurde hier auf Zelle B1 verwiesen. Über die Eigenschaft `Location` geben Sie bekannt, wo das Diagrammobjekt eingefügt werden soll. Indem Sie dieser Eigenschaft die Konstante `xlLocationAsNewSheet` zuweisen, exportieren Sie das bis dahin noch eingebettete Diagrammobjekt in eine neue Diagrammtabelle.

Eine alternative Möglichkeit, auf die Schnelle ein neues Diagrammobjekt einzufügen, sehen Sie im Makro aus Listing 6.74.

```
Sub DiagrammObjektErstellenUndQuelleFestlegen()
  Dim wksBlatt As Worksheet
  Dim shp As Shape

  Set wksBlatt = Tabelle11
   Call DiagrammeEntfernen(wksBlatt)
  Set shp = wksBlatt.Shapes.AddChart(Left:=20, Top:=40, Width:=400,
Height:=200)

  With shp.Chart
   .SetSourceData Source:=Tabelle7.UsedRange, PlotBy:=xlColumns
   .ChartType = xlColumnClustered
   .HasLegend = False
   .ChartStyle = 203
  End With

End Sub
```

Listing 6.74 Umsätze in einem Säulendiagramm ausgeben – Variante 2

Wenden Sie die Methode `AddChart` an, um ein neues, noch leeres Diagrammobjekt einzufügen. Die Größe des Diagramms legen Sie mithilfe der Parameter `Left`, `Top`, `Width` und `Height` in Punkt fest.

Danach übergeben Sie die zugrunde liegende Datenquelle über die Methode `SetSourceData`. Die Eigenschaft `ChartType` bestimmt den Diagrammtyp. Die Legende wird über die Eigenschaft `HasLegend` ausgeblendet, indem Sie ihr den Wert `False` zuweisen. Über die Eigenschaft `ChartStyle` können Sie das Diagramm über einen Stil formatieren. Der `ChartStyle` mit der Nummer 203 ist beispielsweise eine sehr schöne Stapeldarstellung. Weitere Nummern entnehmen Sie exemplarisch Tabelle 6.6. Es gibt Hunderte von Effekten.

Nr.	Chart-Effekt
1	Schwarz
2	Blau
4	Orange
5	Grau
6	Gelb
8	Grün
9	Dunkelgrau
202	dünne blaue Säulen mit Werteanzeige vertikal über den Säulen
203	blaue Scheibchendarstellung (siehe Abbildung 6.21)
204	dicke blaue Säulen mit Schatten (Werte in Säulen)

Tabelle 6.6 Eine kleine Auswahl an »ChartStyle«-Werten

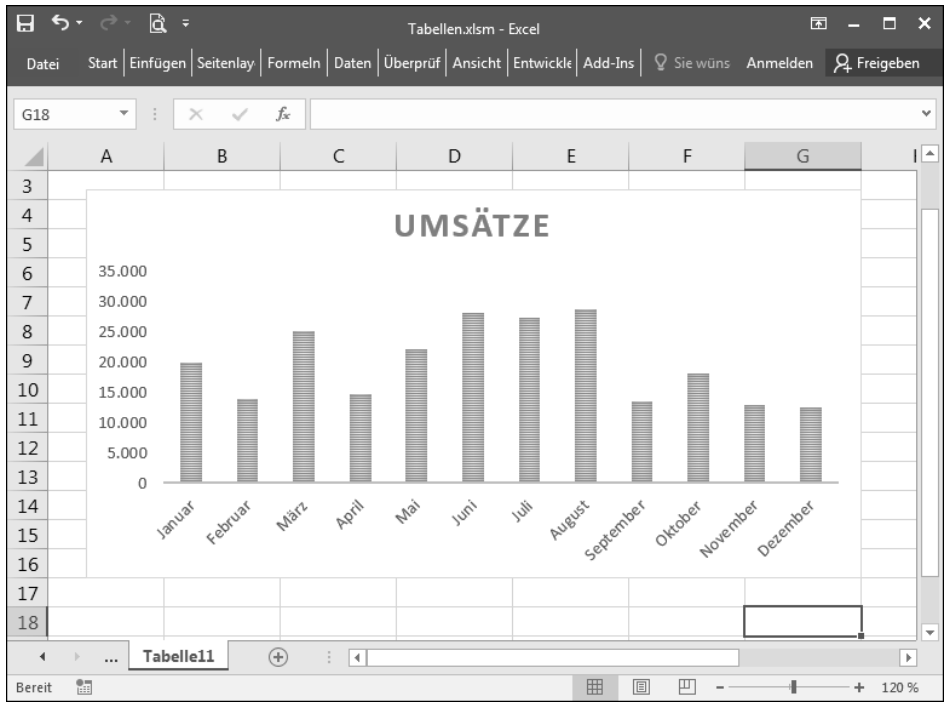


Abbildung 6.21 Noch schneller ein Diagramm einfügen über die Methode »AddChart«

Als Beispiele werden im Makro aus Listing 6.75 in Tabelle12 die Diagramm-Stile 201–249 untereinander und jeweils zwei nebeneinander eingefügt.

```
Sub Aufruf()
  Dim intz As Integer
  Dim intPos As Integer

 intPos = 1
 Call DiagrammeEntfernen(Tabelle12)
 For intz = 201 To 249 Step 2
     Call DiagrammObjektErstellenUndQuelleFestlegenV2(intz, intPos)
     intPos = intPos + 1
 Next intz
End Sub


Sub DiagrammObjektErstellenUndQuelleFestlegenV2(intz, intPos)
  Dim wksBlatt As Worksheet
  Dim shp As Shape

  Set wksBlatt = Tabelle12
  Set shp = wksBlatt.Shapes.AddChart _
  (Left:=20, Top:=intPos * 200, Width:=400, Height:=200)

  With shp.Chart
   .SetSourceData Source:=Tabelle7.UsedRange, PlotBy:=xlColumns
   .ChartType = xlColumnClustered
   .HasLegend = False
   .ChartStyle = intz
  End With

 Set shp = wksBlatt.Shapes.AddChart(Left:=420, Top:=intPos * 200,
                                    Width:=400, Height:=200)
 With shp.Chart
   .SetSourceData Source:=Tabelle7.UsedRange, PlotBy:=xlColumns
   .ChartType = xlColumnClustered
   .HasLegend = False
   .ChartStyle = intz + 1
  End With

End Sub
```

Listing 6.75 Diagrammobjekte untereinander einfügen

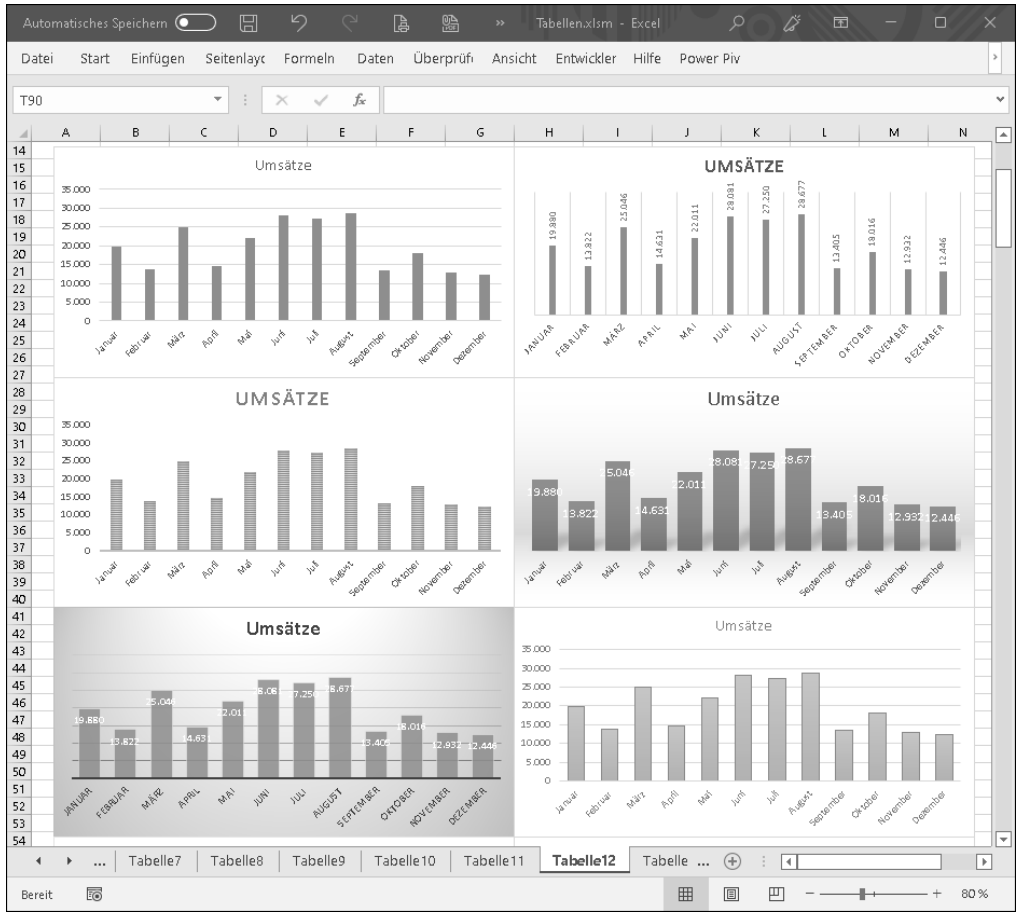


Abbildung 6.22 Die »ChartStyle«-Typen von 201 bis 249

6.13.2 Tagesumsätze im Liniendiagramm anzeigen

Um Schwankungen in den Tagesumsätzen besser zu analysieren, können Sie ein Liniendiagramm generieren. Dazu erstellen Sie dieses Mal nicht extra ein Diagrammblatt, sondern ordnen das Diagramm neben den Umsatzdaten auf demselben Tabellenblatt an.

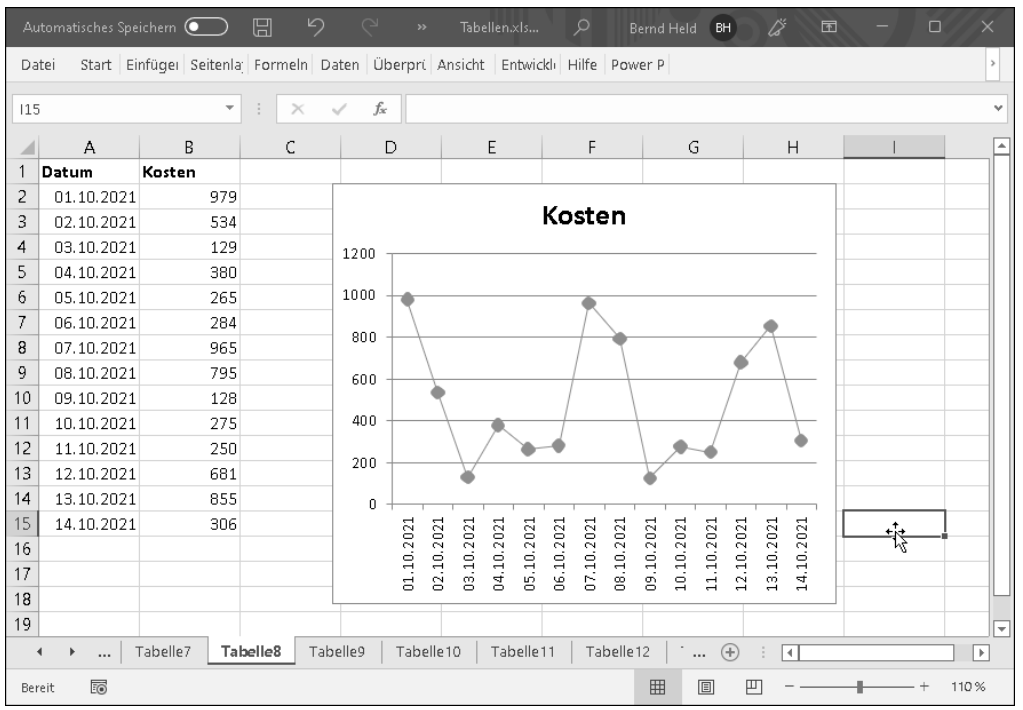
Das eingebettete Diagramm erstellen Sie mit dem Makro aus Listing 6.76:

```
Sub DiagrammeEntfernen(wksBlatt As Worksheet)
 Dim intZ As Long

 For intZ = wksBlatt.ChartObjects.Count To 1 Step -1
   wksBlatt.ChartObjects(intZ).Delete
 Next intZ
```



```
End Sub

Sub DiagrammErstellen02()
  Dim chObj As ChartObject

  Call DiagrammeEntfernen(Tabelle8)

  Set chObj = Tabelle8.ChartObjects.Add(175, 15, 300, 250)
  chObj.Name = "Umsätze Oktober 2015"
  chObj.Chart.SetSourceData Source:=Tabelle8.UsedRange

  With chObj.Chart
     .ChartType = xlLineMarkers
     .HasLegend = False
     .HasTitle = True
     .ChartTitle.Text = "Kosten"
   End With

End Sub
```

Listing 6.76 Tägliche Umsätze in einem Liniendiagramm anzeigen



	Datum	Kosten
2	01.10.2021	979
3	02.10.2021	534
4	03.10.2021	129
5	04.10.2021	380
6	05.10.2021	265
7	06.10.2021	284
8	07.10.2021	965
9	08.10.2021	795
10	09.10.2021	128
11	10.10.2021	275
12	11.10.2021	250
13	12.10.2021	681
14	13.10.2021	855
15	14.10.2021	306

Abbildung 6.23 Das tagesgenaue Liniendiagramm wurde eingefügt.

Rufen Sie im Makro `DiagrammErstellen02` aus Listing 6.76 zunächst das Makro `DiagrammEntfernen` auf. Diesem »Unterprogramm« übergeben Sie den Namen der Tabelle, auf der Sie die eingebetteten Diagramme löschen möchten. Innerhalb dieser Prozedur lassen Sie eine `For ... Next`-Schleife andersherum laufen und wenden im Innern der Schleife die Methode `Delete` an, um die Diagrammobjekte auf dieser Tabelle zu löschen.

Fügen Sie danach über die Methode `Add` ein neues, noch leeres Diagrammobjekt in die Tabelle ein. Dann geben Sie über die Eigenschaft `SetSourceData` bekannt, auf welchen Bereich in der Tabelle sich das Diagramm bezieht. Über die Eigenschaft `ChartType` und die Konstante `xlLineMarkers` legen Sie fest, dass der Diagrammtyp Liniendiagramm verwendet wird. Damit keine Legende angezeigt wird, weisen Sie der Eigenschaft `HasLegend` den Wert `False` zu. Den Titel geben Sie über die Eigenschaften `ChartTitle` und `Text` an, davor geben Sie mithilfe der Eigenschaft `HasTitle` bekannt, dass ein Titel im Diagramm angezeigt werden soll.

6.13.3 Tagesgenaue Formatierung im Punktdiagramm

Fügen Sie jetzt ein Punktdiagramm ein, aus dem Sie Ihre täglichen Ausgaben ablesen können. Als Zusatzfunktion sollen alle Punkte aus der Vergangenheit mit der standardmäßig ausgewählten Farbe Blau belegt und alle in der Zukunft liegenden Planwerte mit einer anderen Farbe versehen werden. Des Weiteren soll die Überschrift etwas größer erscheinen.

Das Makro für diese Aufgabe sehen Sie in Listing 6.77:

```
Sub DiagframmErstellen03()
  Dim chObj As ChartObject
  Dim VarDat As Variant
  Dim Punkt As Point
  Dim intZ As Integer

   Call DiagrammeEntfernen(Tabelle9)

  Set chObj = Tabelle9.ChartObjects.Add(25, 60, 450, 180)
  chObj.Name = "Ausgaben"

  With chObj.Chart
    .SetSourceData Source:=Tabelle9.Range("A3").CurrentRegion
    .ChartType = xlXYScatter
    .HasLegend = False
    .HasTitle = True
    .ChartTitle.Text = "Ausgaben im Oktober"
    .ChartTitle.Font.Name = "Arial"
```



```
    .ChartTitle.Font.Size = 14
    .PlotBy = xlRows
   End With

  With chObj.Chart.SeriesCollection(1)
    VarDat = .XValues
    For Each Punkt In .Points
      Punkt.MarkerSize = 10
      intZ = intZ + 1
      If VarDat(intZ) > Tabelle9.Range("B1").Value Then
        Punkt.MarkerBackgroundColorIndex = 4
        Punkt.MarkerForegroundColorIndex = 1
        Else
        Punkt.MarkerBackgroundColorIndex = 3
        Punkt.MarkerForegroundColorIndex = 1
      End If
    Next Punkt
  End With

End Sub
```

Listing 6.77 Punktdiagramm einfügen und formatieren

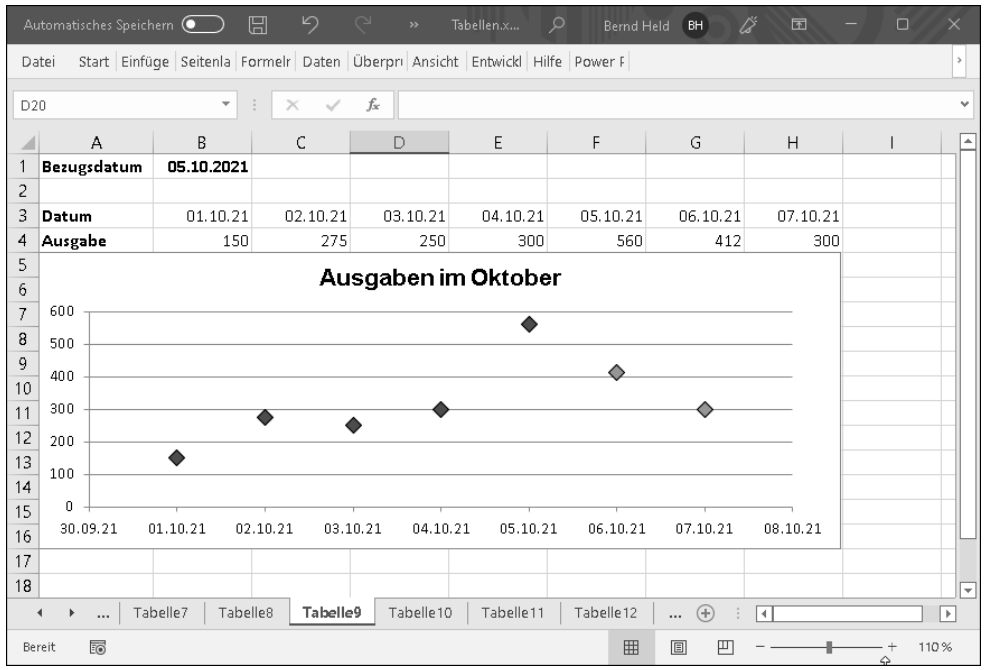


Abbildung 6.24 Punktgenaue Formatierung – Vergangenheit und Zukunft

Da ich auf die Erstellung eines Diagramms schon auf den vorigen Seiten ausgiebig eingegangen bin, sehen wir uns direkt den zweiten Teil von Listing 6.77 an. Dort wird auf das Diagramm referenziert. Danach lesen Sie die einzelnen Werte der Datenreihe in das Datenfeld `VarDat` ein. Im Anschluss daran wird eine Schleife der Form `For Each ... Next` durchlaufen, die prüft, ob die einzelnen Punkte in der Vergangenheit oder in der Zukunft liegen. Liegen die Punkte in der Zukunft, wird über die `MarkerBackgroundColorIndex`-Eigenschaft die Punkt-Innenfläche mit der Farbe Grün versehen. Die Punktumrandung erhält die Farbe Schwarz und wird durch die Eigenschaft `MarkerForegroundColorIndex` festgelegt. Liegen die Werte in der Vergangenheit, dann werden die Punkte rot gefärbt. Beide Punktsymbole werden über die Eigenschaft `MarkerSize` etwas vergrößert.

6.13.4 Diagramme als Grafiken speichern

In Excel können Sie Diagramme über einen Grafikfilter umwandeln. Das bringt Vorteile, wenn Sie Diagramme an Personen ausliefern möchten, die kein Excel installiert haben. Erstellen Sie aus den Excel-Diagrammen speicherschonende Grafikdateien im GIF-Format. Auch wenn Sie Diagramme in Ihrer Textverarbeitung verwenden und dabei keine Excel-Diagramme verknüpfen möchten, ist das Einbinden von Diagrammgrafiken die bessere Methode. Allerdings können diese umgewandelten Diagramme dann nicht mehr aktualisiert werden. Das Makro aus Listing 6.78 wandelt alle eingebetteten Diagramme einer Arbeitsmappe in GIF-Dateien um und legt diese in dem Verzeichnis ab, in dem auch die Mappe mit diesem Makro gespeichert ist.

```
Sub DiagrammeAlsGrafikSpeichern()
 Dim chObj As ChartObject
 Dim wksBlatt As Worksheet
 Dim intz As Integer

  For Each wksBlatt In ThisWorkbook.Worksheets

    For intz = 1 To wksBlatt.ChartObjects.Count
      Set chObj = wksBlatt.ChartObjects(intz)
      chObj.Chart.Export Filename:=ThisWorkbook.Path & "\" & _
      wksBlatt.ChartObjects(intz).Name & ".gif", FilterName:="GIF"
    Next intz

   Next wksBlatt

End Sub
```

Listing 6.78 Alle eingebetteten Diagramme einer Arbeitsmappe werden in Grafikdateien exportiert.

Das Makro aus Listing 6.78 besteht aus zwei Schleifen: Die äußere Schleife arbeitet alle Tabellenblätter der aktiven Arbeitsmappe ab, die innere Schleife exportiert alle eingebetteten Diagramme auf dem jeweiligen Tabellenblatt. Über die Methode `Export` exportieren Sie die Diagramme in ein Grafikformat. Dabei verwendet die Methode `Export` die Parameter `FileName` und `FilterName`. Im Parameter `FileName` geben Sie den Pfad sowie den Namen der Grafikdatei an. Dazu wenden Sie die Eigenschaft `Name` auf das `Chart`-Objekt an. Im Parameter `FilterName` geben Sie den Namen des Grafikfilters an, der verwendet werden soll.

Bei Makro aus Listing 6.79 werden alle in Tabelle12 eingebetteten Diagrammobjekte im Unterverzeichnis *Diagramme* abgelegt. Dabei wird zuvor geprüft, ob das Verzeichnis überhaupt existiert. Wenn ja, dann wird es erst einmal geleert und anschließend neu befüllt.

```
Sub DiagrammExportAktuellesBlatt()
  Dim chtObj As ChartObject
  Dim strPfad As String

  strPfad = ThisWorkbook.Path & "\Diagramme"
  If Dir(strPfad, vbDirectory) = "" Then
    MkDir strPfad
   Else
    Kill strPfad & "\*.jpg"
  End If

  For Each chtObj In Tabelle12.ChartObjects

      chtObj.Chart.Export strPfad & "\" & chtObj.Name & ".jpg"

  Next chtObj

End Sub
```

Listing 6.79 Alle Diagrammobjekte einer Tabelle exportieren

Mithilfe der Funktion `Dir` prüfen Sie, ob es ein bestimmtes Verzeichnis überhaupt gibt. Wenn ja, dann gibt diese Funktion den Namen des gerade geprüften Verzeichnisses zurück. Wenn nicht, dann wird eine leere Zeichenfolge zurückgeliefert. In diesem Fall führen Sie die Anweisung `MkDir` aus, um das Verzeichnis anzulegen. Ist das Verzeichnis bereits angelegt, dann wenden Sie die Anweisung `Kill` an, um alle *.jpg*-Dateien in diesem Verzeichnis zu entfernen. Durchlaufen Sie im Anschluss daran mithilfe einer `For Each ... Next`-Schleife alle Diagrammobjekte auf der Tabelle. Im Innern der Schleife kommt die Methode `Export` zum Einsatz, um die Diagramme als Grafiken abzulegen.

6.13.5 Gewinn und Verlust in einem Säulendiagramm präsentieren

In der folgenden Aufgabe werden in einem Säulendiagramm Monatsergebnisse dargestellt. Damit auf einen Blick sichtbar wird, wenn ein Ergebnis in einem Monat unter den Wert 0 fällt – also ein Verlust eingefahren wird –, sollen solche Datenpunkte mit der Hintergrundfarbe Rot belegt werden. Alle anderen Ergebnisse werden grün gefärbt. Das Makro für diese Aufgabe sehen Sie in Listing 6.80:

```
Sub SäulenVerschiedenfarbig()
  Dim chObj As ChartObject
  Dim SerReihe As Series
  Dim Punkt As Point
  Dim intZ As Integer
  Dim VarDat As Variant

  Call DiagrammeEntfernen(Tabelle10)

  Set chObj = Tabelle10.ChartObjects.Add(1, 55, 780, 250)
  chObj.Name = "Umsätze Oktober 2015"
  chObj.Chart.SetSourceData Source:=Tabelle10.Range("A3").CurrentRegion

  With chObj.Chart
     .ChartType = xlColumnClustered
     .HasLegend = False
     .HasTitle = True
     .ChartTitle.Text = "Ergebnis 2015"
     .Axes(xlCategory).TickLabelPosition = xlLow
   End With
    Set SerReihe = _
    Tabelle10.ChartObjects(1).Chart.SeriesCollection(1)

    With SerReihe
      VarDat = .Values
      For Each Punkt In .Points
        intZ = intZ + 1
        If VarDat(intZ) < 0 Then
          Punkt.Interior.ColorIndex = 3
        Else
          Punkt.Interior.ColorIndex = 4
        End If
      Next Punkt
    End With
End Sub
```

Listing 6.80 Verschiedenfarbige Säulen, je nach Ergebnissituation

Definieren Sie zuerst eine Objektvariable vom Typ `Series`. Anschließend lesen Sie alle Y-Werte in ein Datenfeld ein und durchlaufen danach eine Schleife des Typs `For Each ... Next`, die die einzelnen Datenpunkte überprüft. Die Werte entnehmen Sie dem Datenfeld `VarDat`. Weist ein Datenpunkt einen Wert < 0 auf, so wird der Innenbereich der Säule mit der Farbe Rot formatiert. Bei allen positiven Werten wird ein grüner Farbton gewählt.

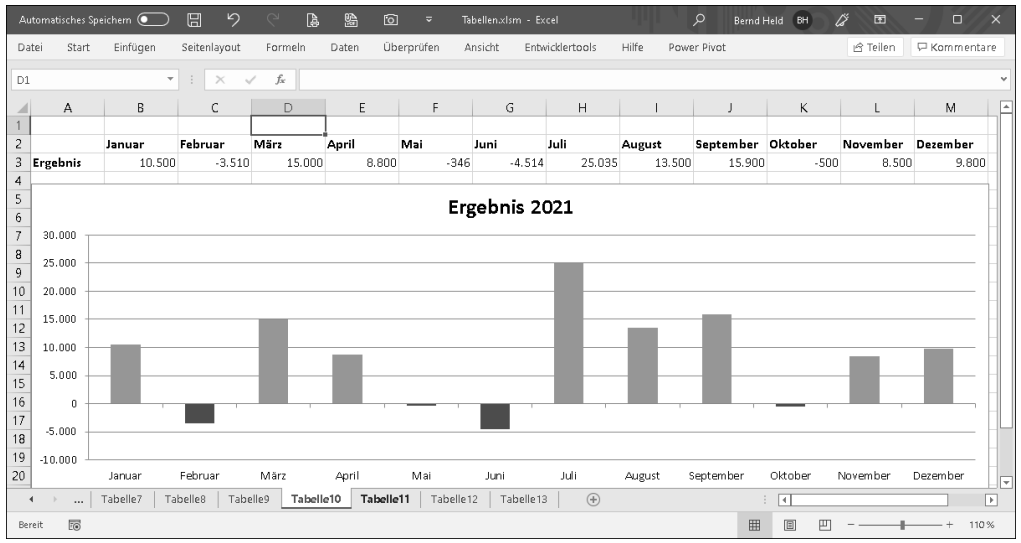


Abbildung 6.25 Negative Ergebnisse werden rot formatiert.

6.13.6 Linienstärke unabhängig von den Markierungssymbolen formatieren

Seit der Excel-Version 2007 ist in Liniendiagrammen die Problematik gegeben, dass, wenn Sie die Stärke einer Linie über VBA bestimmen, dies zugleich für die Stärke der Linie um das Markierungssymbol gilt. Mit dem Makro aus Listing 6.81 überreden Sie Excel dazu, diese beiden Linien getrennt voneinander ansprechen und formatieren zu können.

```
Sub LiniendiagrammLinienStärkeUnabhängigVonMarkierungsrahmen()
  Dim chtObj As ChartObject
  Dim Punkt As Point
  Dim intZ As Integer

  Set chtObj = Tabelle8.ChartObjects(1)
    chtObj.Chart.SeriesCollection(1).Format.Line.Weight = xlHairline

    For intZ = 1 To chtObj.Chart.SeriesCollection(1).Points.Count
      Set Punkt = chtObj.Chart.SeriesCollection(1).Points(intZ)
```

```
    Punkt.Format.Line.Weight = xlThin
  Next intZ

  chtObj.Chart.SeriesCollection(1).Border.Weight = xlThin

End Sub
```

Listing 6.81 Markierungssymbole unabhängig von der Linienstärke formatieren

Zunächst einmal: Den Effekt, dass Excel beim Vergrößern der Markierungssymbole automatisch auch die Linienstärke verdickt, können Sie nicht verhindern, wohl aber die Linienstärke durch eine Nachformatierung wieder zurücksetzen. Legen Sie zunächst die Linienstärke für die komplette Linie über die Eigenschaft `Weight` mit der Konstanten `xlHairline` fest. Danach formatieren Sie das Markierungssymbol in einer Schleife Punkt für Punkt über das Objekt `Point` und die Eigenschaft `Weight` über die Konstante `xlThick`. Dadurch wird leider auch die Linie größer formatiert. Daher weisen Sie dem Objekt `Border` im Nachgang der Eigenschaft `Weight` die Konstante `xlThin` zu, um die Linie wieder zu verschmälern.

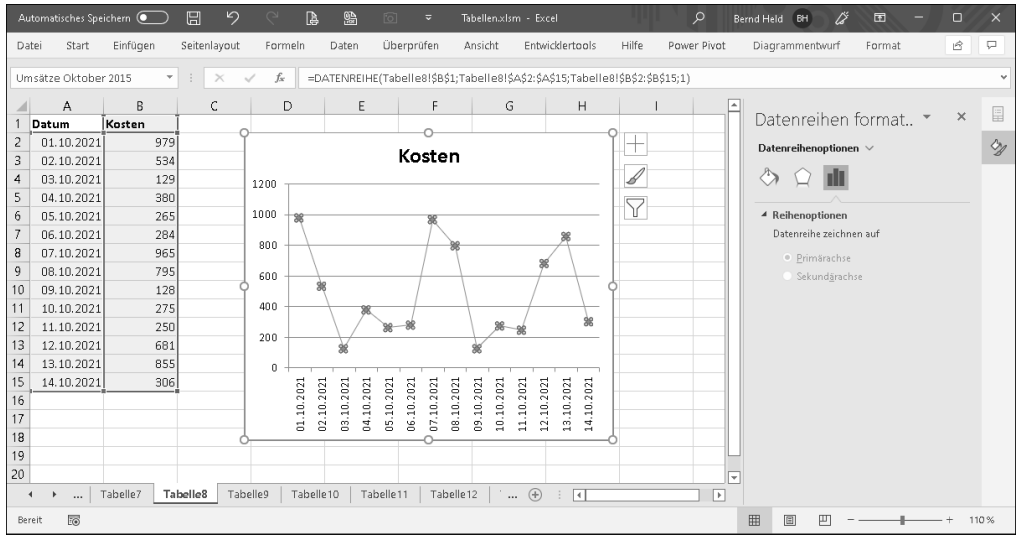


Abbildung 6.26 Dicke Markierungssymbole, dünne Linien – so sieht es nach etwas aus.

6.13.7 Sparklines automatisch erstellen

Beim letzten Beispiel in diesem Kapitel erstellen wir ein sogenanntes *Sparkline* in TABELLE13. Ein Sparkline ist ein Zellendiagramm, das ab der Version Excel 2010 zur Verfügung steht. Schauen Sie sich zunächst einmal die Ausgangssituation in Abbildung 6.27 an.

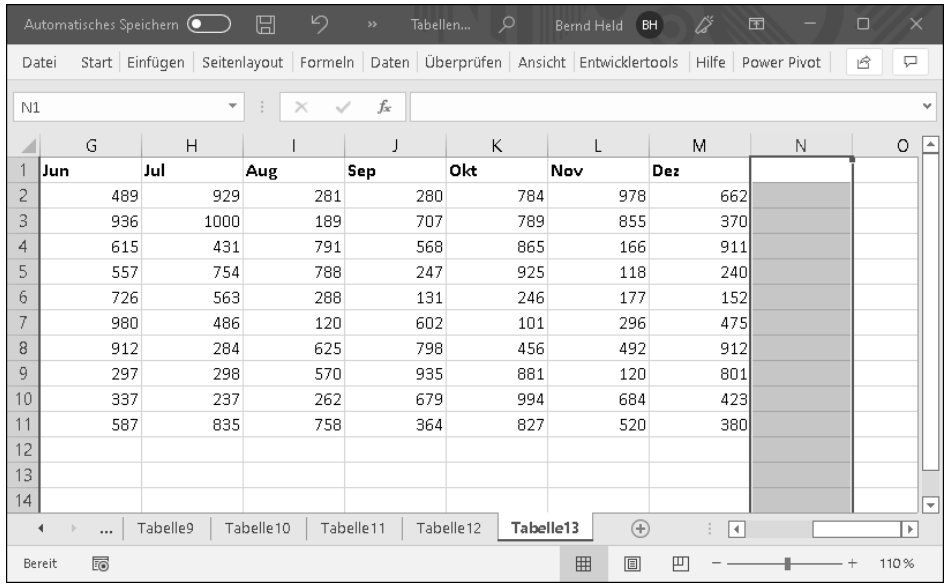


Abbildung 6.27 Eine Tabelle mit monatlichen Umsätzen in T€

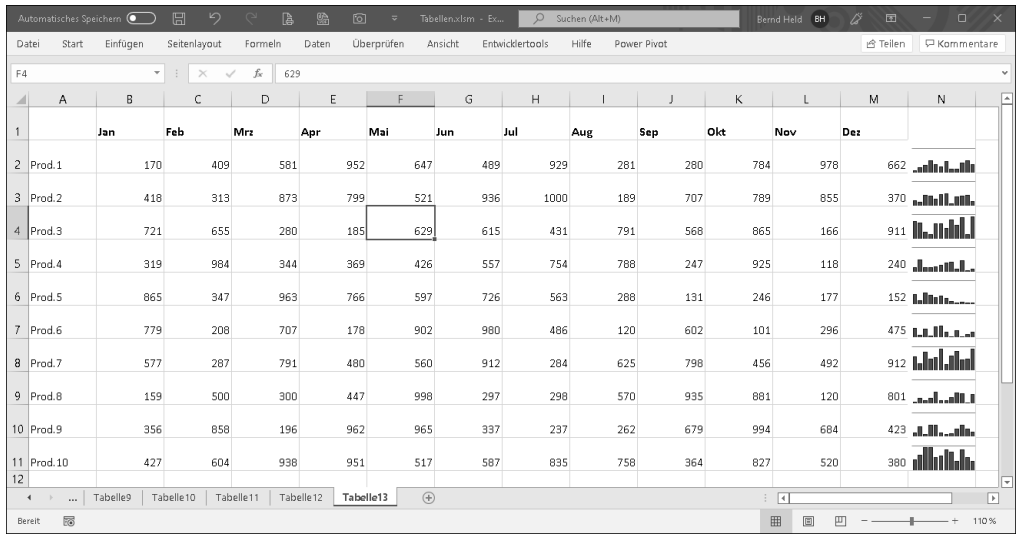
Das Makro aus Listing 6.82 erstellt auf Basis der Monatswerte in Spalte O für jedes Produkt eine Zellengrafik, aus der man die monatliche Entwicklung leichter ersehen kann.

```
Sub SparklinesErstellen()
  Dim rngBereich As Range
  Dim intz As Integer
  Dim varZeile As Variant

  With Tabelle13

    Set rngBereich = .Range("B2:M11")
    For intz = .ChartObjects.Count To 1 Step -1
        .ChartObjects(intz).Delete
    Next intz
  End With

  For Each varZeile In rngBereich.Rows
    Tabelle13.Shapes.AddChart.Select
    With ActiveChart
     .ChartType = xlColumnClustered
     .SetSourceData Source:=varZeile
     .Axes(xlCategory, xlPrimary).HasTitle = False
     .Axes(xlValue, xlPrimary).HasTitle = False
```

```
      .HasAxis(xlCategory, xlPrimary) = False
      .HasAxis(xlValue, xlPrimary) = False
      .SeriesCollection(1).Interior.ColorIndex = 3
      .SeriesCollection(1).Border.ColorIndex = 5
     .Parent.Top = Tabelle13.Cells(varZeile.Row, 14).Top
     .Parent.Left = Tabelle13.Cells(varZeile.Row, 14).Left
     .Parent.Height = Tabelle13.Cells(varZeile.Row, 14).Height
     .Parent.Width = Tabelle13.Cells(varZeile.Row, 14).Width
     .Parent.Border.ColorIndex = xlNone
     .PlotArea.Top = 0
     .PlotArea.Left = 0
     .PlotArea.Height = .Parent.Height
     .PlotArea.Width = .Parent.Width
     .ChartGroups(1).GapWidth = 50
    End With
  Next varZeile
End Sub
```

Listing 6.82 Ein Sparkline per Makro einfügen und nachformatieren

Zunächst wird der Datenbereich, auf den sich die Sparklines beziehen, bekannt gegeben. In einer Vorschleife werden danach alle eventuell bereits in der Tabelle befindlichen Zellendiagramme entfernt. Danach wird die Methode `AddChart` angewendet, um die Zellengrafik einzufügen. Die folgenden Eigenschaften bestimmen den Diagrammtyp sowie das Aussehen des Zellendiagramms.



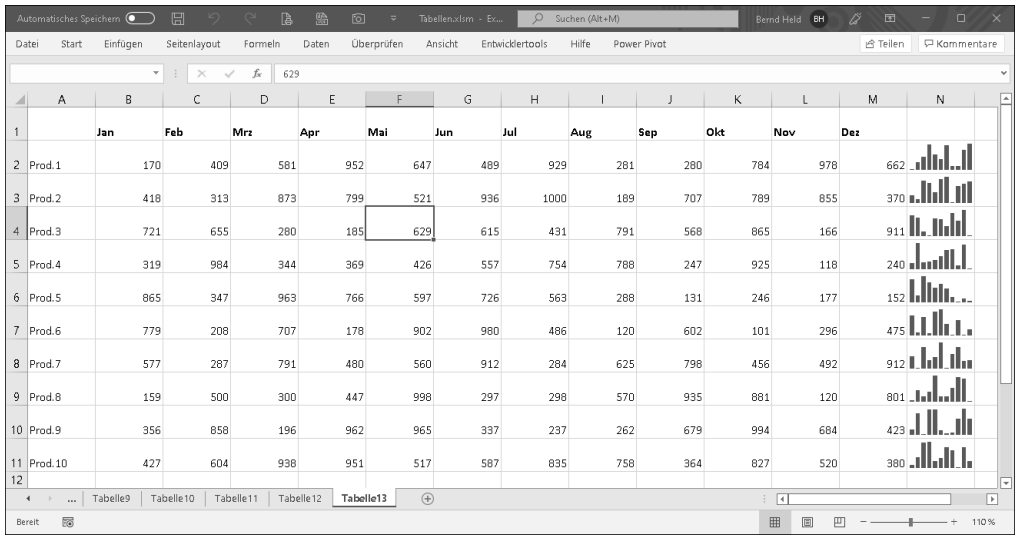
	Jan	Feb	Mrz	Apr	Mai	Jun	Jul	Aug	Sep	Okt	Nov	Dez
Prod.1	170	409	581	952	647	489	929	281	280	784	978	662
Prod.2	418	313	873	799	521	936	1000	189	707	789	855	370
Prod.3	721	655	280	185	629	615	431	791	568	865	166	911
Prod.4	319	984	344	369	426	557	754	788	247	925	118	240
Prod.5	865	347	963	766	597	726	563	288	131	246	177	152
Prod.6	779	208	707	178	902	980	486	120	602	101	296	475
Prod.7	577	287	791	480	560	912	284	625	798	456	492	912
Prod.8	159	500	300	447	998	297	298	570	935	881	120	801
Prod.9	356	858	196	962	965	337	237	262	679	994	684	423
Prod.10	427	604	938	951	517	587	835	758	364	827	520	380

Abbildung 6.28 Ein Sparkline wurde eingefügt.

Etwas kürzer und ohne Nachformatierung können Sie auch das Makro aus Listing 6.83 einsetzen.

```
Sub SparklineEinfügenLinie()
 Dim intz As Integer

  With Tabelle13

    For intz = .ChartObjects.Count To 1 Step -1
           .ChartObjects(intz).Delete
    Next intz

    .Range("$N$2:$N$11").SparklineGroups.Add Type:=xlSparkColumn, _
     SourceData:="Tabelle13!B2:L11"

  End With

End Sub
```

Listing 6.83 Ein Standard-Sparkline per Makro einfügen



	Jan	Feb	Mrz	Apr	Mai	Jun	Jul	Aug	Sep	Okt	Nov	Dez
Prod.1	170	409	581	952	647	489	929	281	280	784	978	662
Prod.2	418	313	873	799	521	936	1000	189	707	789	855	370
Prod.3	721	655	280	185	629	615	431	791	568	865	166	911
Prod.4	319	984	344	369	426	557	754	788	247	925	118	240
Prod.5	865	347	963	766	597	726	563	288	131	246	177	152
Prod.6	779	208	707	178	902	980	486	120	602	101	296	475
Prod.7	577	287	791	480	560	912	284	625	798	456	492	912
Prod.8	159	500	300	447	998	297	298	570	935	881	120	801
Prod.9	356	858	196	962	965	337	237	262	679	994	684	423
Prod.10	427	604	938	951	517	587	835	758	364	827	520	380

Abbildung 6.29 Das Standard-Sparkline wurde eingefügt.

Des Weiteren haben Sie die Möglichkeit, beispielsweise den größten Wert in einem Sparkline automatisch zu kennzeichnen.

```
Sub SparlineHoechstpunktFormat()

  With Tabelle13
        .Range("$N$2:$N$11").SparklineGroups.Item(1).Points.Highpoint.Visible
= True
  End With

End Sub
```

Listing 6.84 Ein Standard-Sparkline per Makro einfügen

Über die Eigenschaften `HighPoint` und `Visible` können Sie den Höchstwert im Sparkline automatisch kennzeichnen.

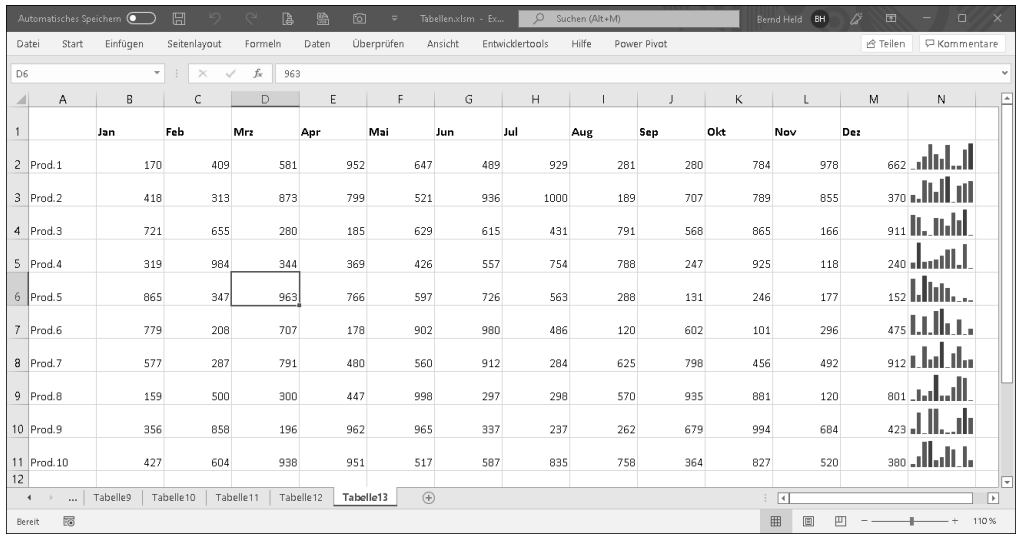


Abbildung 6.30 Den Höchstwert pro Produkt kennzeichnen

6.14 Tabellen blitzschnell vergleichen und Unterschiede dokumentieren

Auf den folgenden Seiten werde ich Ihnen zwei schnelle Varianten zeigen, über die Sie zwei Tabellen miteinander vergleichen können. Dabei sollen die »abweichenden« Zellen in einer Liste dokumentiert und alle Abweichungen automatisch eingefärbt werden.

Als Ausgangssituation liegen zwei Tabellen mit jeweils 10.000 Zeilen und 10 Spalten vor.

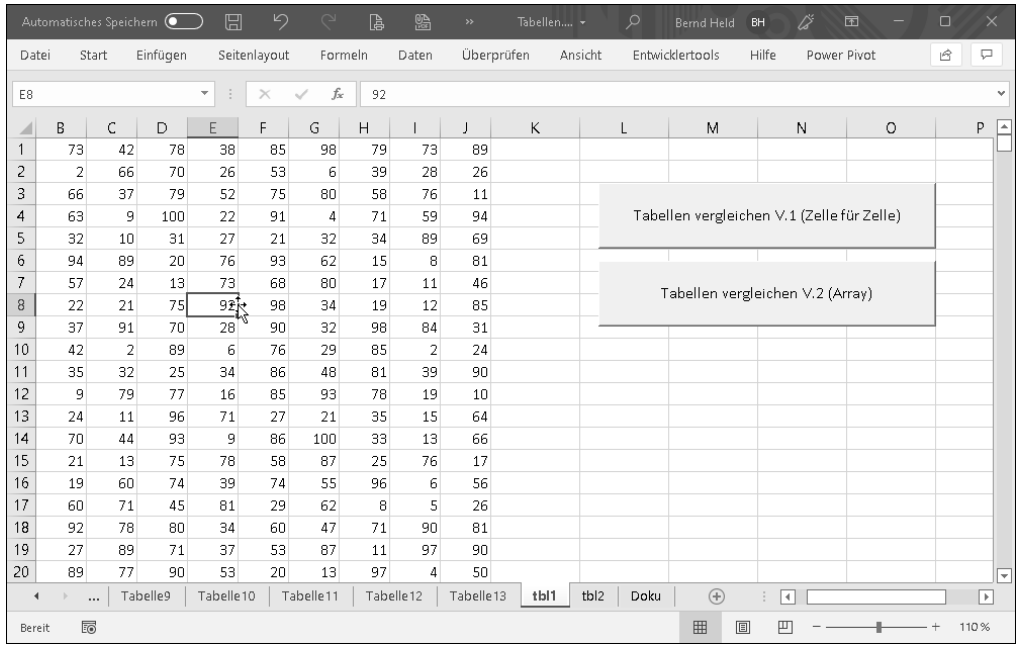


Abbildung 6.31 Die Tabellen tbl1 und tbl2 sollen verglichen werden.

Beim ersten Ansatz wird eine `For each ... Next`-Schleife eingesetzt, die die erste Tabelle Zelle für Zelle abarbeitet. Innerhalb der Schleife wird die aktuelle Zelle mit der Zelle der anderen Tabelle (gleiche Koordinate) verglichen. Wird eine Abweichung erkannt, wird die Zelle rot eingefärbt und die Koordinate in der Tabelle TBL_DOKU dokumentiert.

```
Sub TabellenVergleichenV1()
  Dim rngBereich As Range
  Dim rngZelle As Range
  Dim lngZeile As Long

  Debug.Print "Start V.1: " & Now

  tbl_Doku.UsedRange.Clear
  lngZeile = 1

  Set rngBereich = tbl_1.Range("A1:J10000")
  rngBereich.Interior.ColorIndex = xlColorIndexNone
```

```
  For Each rngZelle In rngBereich

    If rngZelle.Value <> tbl_2.Range(rngZelle.Address).Value Then
      rngZelle.Interior.ColorIndex = 3
      tbl_Doku.Cells(lngZeile, 1).Value = rngZelle.Address
      lngZeile = lngZeile + 1
    End If

  Next rngZelle

  Debug.Print "Ende V.1: " & Now

End Sub
```

Listing 6.85 Tabellen vergleichen – Zelle für Zelle

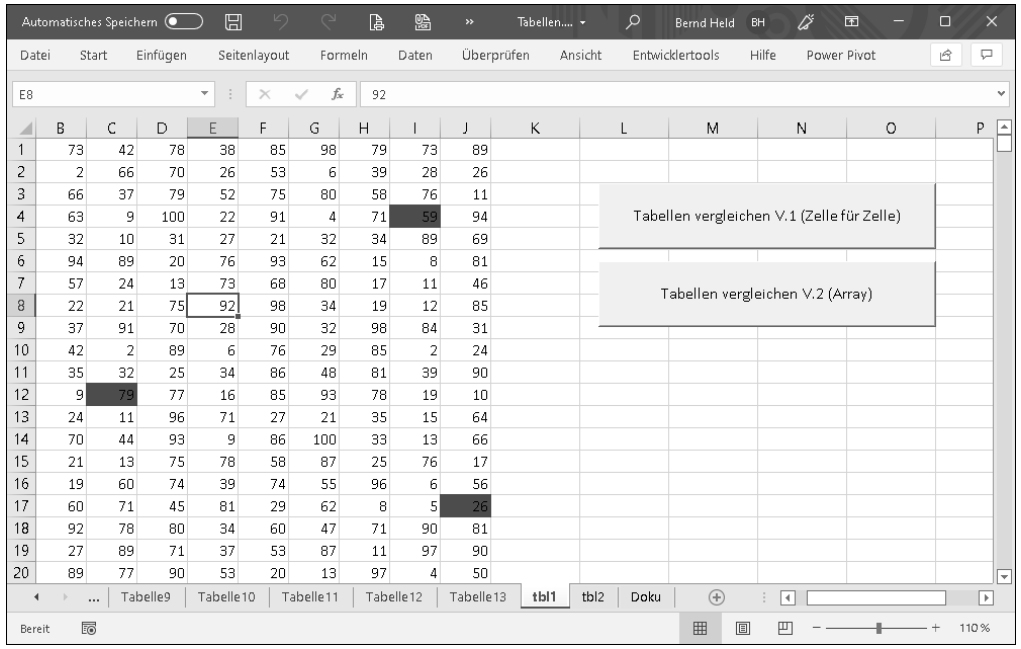


Abbildung 6.32 Die Unterschiede wurden farbig gekennzeichnet.

Die Laufzeit dieser ersten Variante beträgt bei 100.000 Zellen ca. eine Sekunde. Die einzelnen Abweichungen werden zusätzlich in der Tabelle TBL_DOKU dokumentiert.

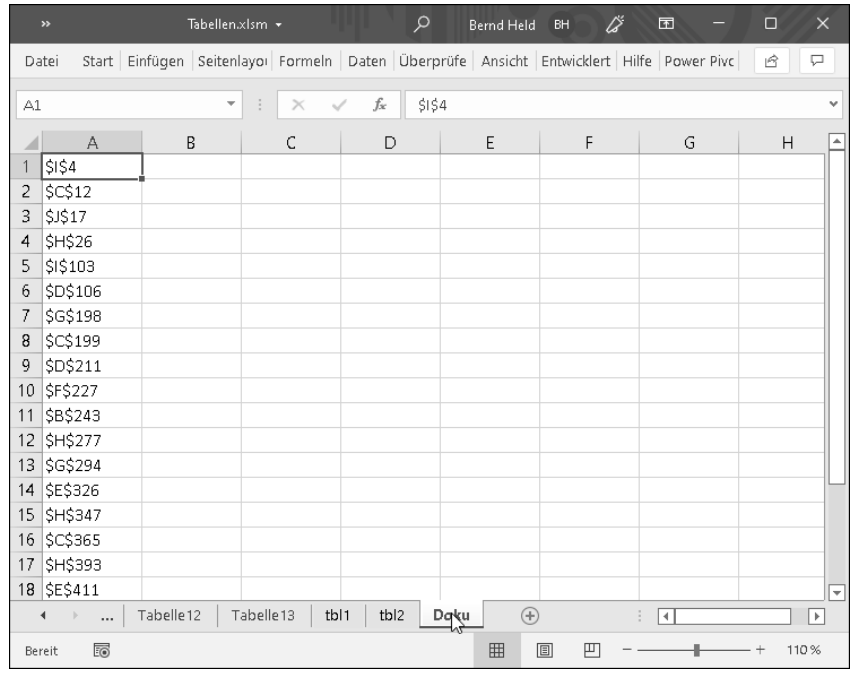


Abbildung 6.33 Alle abweichenden Zellen werden in einer Tabelle festgehalten.

```
Sub TabellenVergleichenV2()
  Dim VarDat As Variant
  Dim VarVGL As Variant
  Dim VarDok()
  Dim strBereich As String
  Dim lngZeile As Long, lngZeileMax As Long
  Dim lngSpalte As Long, lngSpalteMax As Long, lngZ As Long

  Debug.Print "Start V.2: " & Now
  tbl_Doku.UsedRange.Clear

  strBereich = "A1:J10000"

  VarDat = tbl_1.Range(strBereich)
  VarVGL = tbl_2.Range(strBereich)
  lngZeileMax = UBound(VarDat, 1)
  lngSpalteMax = UBound(VarDat, 2)

  tbl_1.Range(strBereich).Interior.ColorIndex = xlColorIndexNone
```

```
  For lngZeile = 1 To lngZeileMax

    For lngSpalte = 1 To lngSpalteMax

      If VarDat(lngZeile, lngSpalte) <> _
       VarVGL(lngZeile, lngSpalte) Then
        tbl_1.Cells(lngZeile, lngSpalte).Interior.ColorIndex = 6
        ReDim Preserve VarDok(lngZ)
        VarDok(lngZ) = tbl_1.Cells(lngZeile, lngSpalte).Address
        lngZ = lngZ + 1
      End If

    Next lngSpalte

  Next lngZeile

  tbl_Doku.Range("A1").Resize(lngZ, 1).Value = Application.Transpose(VarDok)

  Debug.Print "Ende V.2: " & Now

End Sub
```

Listing 6.86 Tabellen vergleichen – Array-Lösung

Bei der zweiten Variante wird der Vergleich mithilfe von zwei Datenfeldern durchgeführt. Dabei werden beide Tabellen in die Arrays `VarDat` und `VarVGL` gepackt, und die komplette Verarbeitung findet im Arbeitsspeicher statt. Wird eine Abweichung festgestellt, wird ein dritter Array mit dem Namen `VarDok` jeweils mit der Anweisung `Redim Preserve` um ein Feld erweitert und die Adresse der Zelle (die die gleiche Adresse ist wie in der Tabelle selbst) geschrieben. Die Formatierung erfolgt bereits während der Schleife, sodass das Formatieren von Zellen in diesem Fall nicht ins Gewicht fällt.

Am Ende des Makros wird dann der Dokumentations-Array `VarDok` beginnend ab der Zelle A1 in die Tabelle TBL_DOKU geschrieben. Zu diesem Zweck wird der dazu benötigte Platz über die Methode `Resize` bereitgestellt und der Array mithilfe der Funktion `Transpose` gedreht.

Die Laufzeit dieser Variante beträgt bei 100.000 Zellen deutlich weniger als eine Sekunde.

Als kleines Feature wäre es jetzt doch optimal, wenn man über einen Doppelklick auf die jeweilige Zelle der Abweichungskoordinaten direkt in die Tabelle an die entsprechende Position befördert würde. Kein Problem. Das Ereignis `Worksheet_BeforeDoubleClick` direkt hinter der Tabelle TBL_DOKU führt genau diese Aufgabe aus.


```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
                                        Cancel As Boolean)
  If Target.Column = 1 Then

    If Target.Value <> "" Then
      Application.Goto Reference:=Tabelle1.Range(Target.Value), _
                       scroll:=True
      Cancel = True
    End If

  End If
End Sub
```

Listing 6.87 Per Doppelklick direkt zur Abweichung gelangen

Zunächst muss geprüft werden, ob der Doppelklick in Spalte A auf eine gefüllte Zelle durchgeführt worden ist. Wenn ja, sorgt die Methode `Goto` dafür, dass genau an diese Zelle in der Tabelle1 gesprungen wird. Der Parameter `Scroll` sorgt dafür, dass die angesprungene Zelle links oben im Fenster angeordnet wird.

Das Video dazu finden Sie unter: *www.youtube.com/watch?v=kD5l5QBbin8*