VBA mit Excel

Das umfassende Handbuch. Konzepte und Techniken der VBA-Programmierung. Das Standardwerk für Einsteiger und fortgeschrittene Anwender

Bearbeitet von
Bernd Held

2., aktualisierte und erweiterte Auflage 2015. Buch. 950 S. Hardcover
ISBN 978 3 8362 3821 2
Format (B x L): 16 x 24 cm

Leseprobe

In diesem Auszug erfahren Sie von Bernd Held alles über die praktische Anwendung der VBA-Sprachelemente. Außerdem lernen Sie, wie Sie Diagramme programmieren und können einen Blick in das vollständige Inhalts- und Stichwortverzeichnis des Buchs werfen.

Bernd Held

VBA mit Excel – Das umfassende Handbuch

950 Seiten, gebunden, 2. Auflage 2015
49,90 Euro, ISBN 978-3-8362-3821-2

www.rheinwerk-verlag.de/3891

Kapitel 3
Die Sprachelemente von Excel-VBA

Das Wesentliche, was eine Programmiersprache ausmacht, sind ihre Sprachelemente. In diesem Kapitel erfahren Sie, wie Sie mit Hilfe von Abfragen, Schleifen und anderen Anweisungen Ihre Makros flexibel gestalten.

Die Sprachelemente lassen sich nicht mit dem Makrorekorder aufzeichnen, sondern Sie müssen sie selbst erstellen. Der richtige Einsatz der Sprachelemente macht letztendlich die Kunst der Programmierung aus.

Die Beispiele

Sie finden alle Beispiele zum Download unter *http://www.rheinwerk-verlag.de/3891* in der Datei Sprachelemente.xlsm.

Fragen zum Download?

Nähere Angaben zum Download finden Sie bei Bedarf im betreffenden Hinweiskasten in der Einleitung von Kapitel 1, »Die Entwicklungsumgebung von Excel«.

3.1 Bedingungen

Mit Bedingungen können Sie in Excel bestimmte Zustände abfragen und je nach Zustand anders reagieren. Es ist aus meiner Sicht eines der wichtigsten Elemente in der Programmierung. Jede Verzweigung hat eine oder mehrere Bedingungen, dann einen Zweig, der eintritt, wenn die Bedingung zutrifft, und einen Zweig, der abgearbeitet wird, wenn die Bedingung nicht zutrifft.

Bei den Verzweigungen kann zwischen zwei Formen unterschieden werden:

Normalform:

```
IF Bedingung Then Aktion1 Else Aktion2
```

Blockform:

```
If Bedingung Then
  Aktion1a
  Aktion1b
  Aktion1c
Else
  Aktion2a
  Aktion2b
End if
```

Verwenden Sie die Blockform, können mehrere Schritte nacheinander durchgeführt werden. Dabei müssen Sie aber beachten, dass Sie die Anweisung mit einem `End If` abschließen müssen.

Bedingungen werden in der Praxis unter anderem für Prüfungen aller Art eingesetzt. Sie sind in fast jeder Schleife als »Innerei« vorhanden.

Es folgen nun einige typische Beispiele für den Einsatz von Verzweigungen.

3.2 Typische Aufgaben aus der Praxis

Lernen Sie auf den nächsten Seiten ausgewählte Praxisbeispiele kennen, die mit Bedingungen arbeiten.

3.2.1 Wert in einer Spalte suchen

Bei der folgenden Aufgabenstellung soll ein eindeutiger Wert in einer Spalte gefunden und anschließend gekennzeichnet werden. Der zu suchende Wert soll über eine `InputBox` vom Anwender eingegeben werden. Schauen Sie sich dazu das Makro aus Listing 3.1 an.

```
Sub WertInSpalteSuchen()
  Dim rngTreffer As Range
  Dim strSuchbegriff As String

  Tabelle1.Range("A:A").Interior.ColorIndex = xlColorIndexNone
  strSuchbegriff = InputBox("Suchbegriff eingeben!", "Direktsuche", 4720)

  If Len(strSuchbegriff) <> 0 Then

   Set rngTreffer = Tabelle1.Range("A:A").Find _
   (What:=strSuchbegriff, LookIn:=xlValues, LookAt:=xlWhole)
```



```
    If rngTreffer Is Nothing Then
        MsgBox "Wert nicht gefunden"
    Else
        rngTreffer.Interior.ColorIndex = 4
    End If

  End If
End Sub
```

Listing 3.1 Einen bestimmten Wert in einer Spalte finden

Deklarieren Sie im ersten Schritt des Makros aus Listing 3.1 eine Objektvariable mit dem Namen `rngTreffer` vom Typ `Range` sowie eine `String` Variable mit dem Namen `strSuchbegriff`.

Entfärben Sie Spalte A über die Eigenschaft `ColorIndex`, der Sie die Konstante `xlColorIndexNone` zuweisen.

Rufen Sie danach die Funktion `InputBox` auf. Dabei geben Sie im ersten Argument die Meldung an, die im Meldungsfenster erscheinen soll. Im zweiten Argument definieren Sie den Titel, der oberhalb des Meldungsfensters angezeigt werden soll. Beim dritten Argument können Sie eine Vorbelegung für das Textfeld im Meldungsfenster bestimmen.

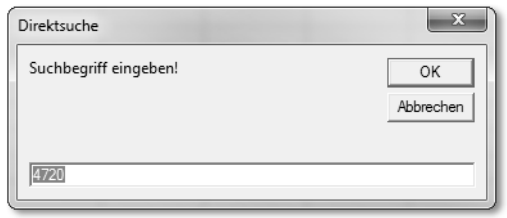


Abbildung 3.1 Über eine InputBox mit Vorbelegung eine Eingabe verlangen

Werten Sie die Eingabe des Anwenders aus. Wenn er nichts eingegeben hat, dann liefert die Funktion `Len` den Wert `0`. In diesem Fall soll nichts weiter passieren.

Im anderen Fall setzen Sie eine Direktsuche in Excel auf, indem Sie die Methode `Find` einsetzen. Diese Methode hat drei wichtige Parameter:

- `What`: Bei diesem Parameter muss der zu suchende Begriff eingegeben werden. Die Angabe dieses Parameters ist Pflicht.
- `LookIn`: Dieser Parameter bestimmt, dass beispielsweise in Werten, nicht in Formeln gesucht werden soll.
- `LookAt`: Über diesen Parameter legen Sie fest, ob Sie vollqualifiziert oder teilqualifiziert suchen möchten. Wenn Sie diesen Parameter nicht angegeben, dann wird

dafür die Konstante `xlPart` angenommen, was bedeutet, dass Excel auch Werte, die dem Suchbegriff ähnlich sind, findet. Weisen Sie diesem Parameter die Konstante `xlWhole` zu, damit die Suche wirklich nur eine eindeutige Übereinstimmung findet.

Nachdem die Suche aufgesetzt ist, muss geprüft werden, ob sie erfolgreich war. Auch zu diesem Zweck kommt die Anweisung `If` ins Spiel. War die Folge erfolglos, dann ist die Objektvariable `rngTreffer` leer. Im anderen Falle ist die Objektvariable erfolgreich gesetzt und zeigt genau auf die Fundstelle. Damit haben Sie Zugriff auf die gefundene Zelle, die Sie im Anschluss daran über die Eigenschaft `ColorIndex` einfärben.

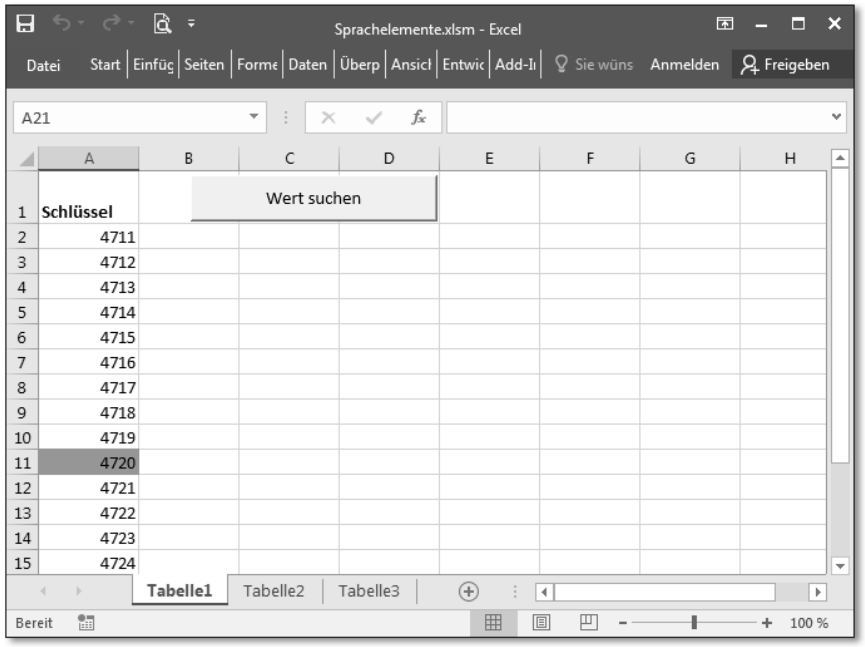


Abbildung 3.2 Der gesuchte Wert wurde gefunden und gekennzeichnet.

3.2.2 Liegt die aktive Zelle in einem vorgegebenen Bereich?

Bei dieser Fragestellung liegt in Tabelle2 der Bereich A1:D10 vor. Das Makro aus Listing 3.2 prüft, ob die aktive Zelle in diesem Bereich liegt.

```
Sub LiegtZelleImBereich()
 Dim rngBereich As Range

    Set rngBereich = Tabelle2.Range("A1:D10")

    If Intersect(ActiveCell, rngBereich) Is Nothing Then
        MsgBox "Die Zelle " & ActiveCell.Address & _
        " liegt außerhalb des Zielbereichs " & rngBereich.Address
    Else
        MsgBox "Die Zelle " & ActiveCell.Address & _
        " liegt im Zielbereich " & rngBereich.Address
    End If

End Sub
```

Listing 3.2 Prüfung, ob eine Zelle in einem Bereich liegt

Deklarieren Sie zu Beginn des Makros aus Listing 3.2 eine Objektvariable vom Typ `Range` mit dem Namen `rngBereich`. Geben Sie danach über die Anweisung `Set` bekannt, wo sich der Bereich in Tabelle2 befinden soll.

Mit Hilfe der Methode `Intersect` können Sie überprüfen, ob die aktive Zelle in dem vorgegebenen Bereich liegt. Wenn nicht, dann liefert die `If`-Bedingung als Rückgabe den Wert `Nothing`, was Sie mit einer Meldung am Bildschirm über die Funktion `MsgBox` quittieren.

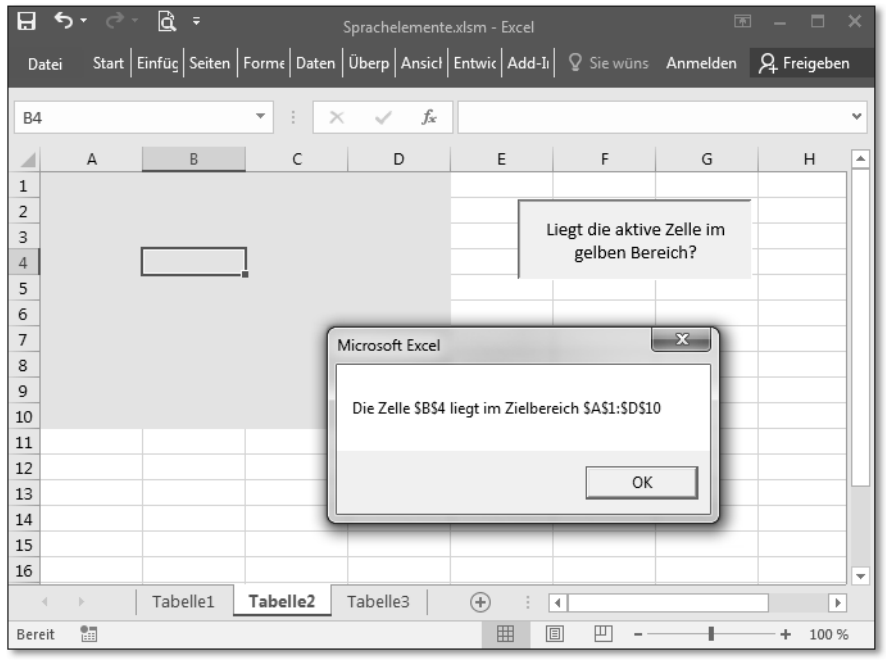


Abbildung 3.3 Die Bereichsprüfung ist in diesem Fall positiv.

3.2.3 Prüfung, ob eine bestimmte Datei verfügbar ist

Das Beispiel aus Listing 3.3 prüft, ob eine bestimmte Datei in einem vorgegebenen Verzeichnis existiert.

```
Sub IstDateiVorhanden()
 Dim strDatei As String
 Const ConstDatei = _
 "C:\Users\Bernd.Held\Desktop\Excel-VBA-Handbuch\Beispiele\Kundenliste.txt"

  strDatei = Dir(ConstDatei)
  If strDatei <> "" Then
      MsgBox "Datei vorhanden!", vbExclamation
  Else
      MsgBox "Datei " & strDatei & " nicht da!", vbCritical
  End If

End Sub
```

Listing 3.3 Prüfung, ob eine bestimmte Datei in einem vorgegebenen Verzeichnis liegt

Deklarieren Sie im ersten Schritt des Makros aus Listing 3.3 eine `String`-Variable mit dem Namen `strDatei`. Als Konstante geben Sie den Pfad und Dateinamen zu der Datei an, deren Existenz Sie prüfen möchten. Selbstverständlich müssen Sie den Pfad- und Dateinamen anpassen.

Mit Hilfe der Funktion `Dir` können Sie prüfen, ob die angefragte Datei im vorgegebenen Verzeichnis existiert. Wenn ja, dann ergibt die Prüfung über die Anweisung `If`, dass der Name der Datei in der Variablen `strDatei` steht. Kann die Datei nicht gefunden werden, dann bleibt der Inhalt der Variablen leer.

3.2.4 Spalteninhalte direkt nach der Eingabe umsortieren

Auch bei Ereignissen, die ich in Kapitel 9, »Ereignisse programmieren«, behandeln werde, können Sie mit Bedingungen arbeiten. Bei der nächsten Aufgabenstellung liegen in Tabelle3 Zahlenwerte vor. Neue Eingaben in Spalte A sollen direkt nach der Eingabe automatisch sortiert werden. Sehen Sie sich dazu einmal die Ausgangssituation in Abbildung 3.4 an.

Um die automatische Umsortierung einzubauen, stellen Sie das Ereignis `Worksheet_Change` wie folgt ein:

1. Klicken Sie in der Excel-Oberfläche mit der rechten Maustaste auf den Tabellennamen, und wählen Sie den Befehl Code anzeigen aus dem Kontextmenü. Sie landen jetzt direkt hinter der Tabelle in der Entwicklungsumgebung.
2. Stellen Sie im Codefenster auf der rechten Seite oben im ersten Dropdown den Eintrag Worksheet ein. Dadurch wird das Standardereignis `Worksheet_SelectionChange` eingestellt, das auf jedes Verschieben des Cursors reagiert. Dieses Ereignis brauchen wir nicht.

3. Wählen Sie im zweiten Dropdown oberhalb des Codefensters das Ereignis Change aus. Dadurch wird der noch leere Ereignisrahmen des Ereignisses eingestellt.
4. Komplettieren Sie das noch leere Ereignis wie in Listing 3.4 gezeigt.

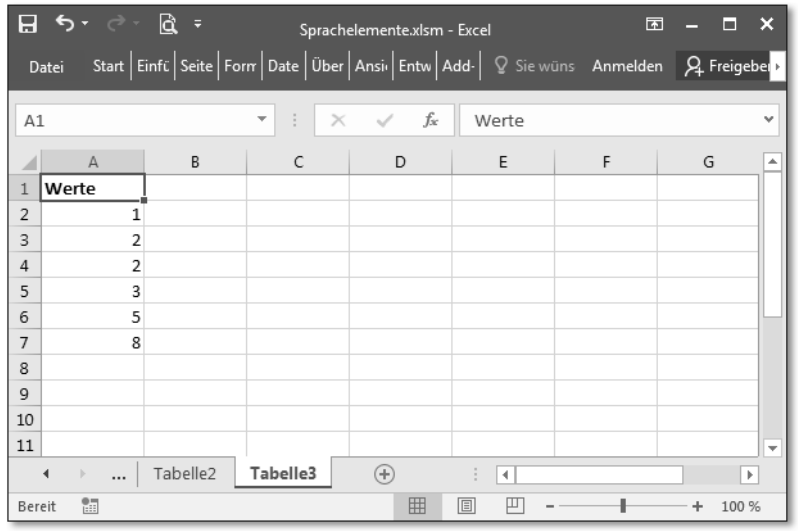


Abbildung 3.4 Neue Werte sollen am Ende der Liste erfasst und automatisch umsortiert werden.

```
Private Sub Worksheet_Change(ByVal Target As Range)
 Dim lngZeileMax As Long

    If Target.Column = 1 Then

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

      Range("A1:A" & lngZeileMax).Sort _
      Key1:=Range("A1"), Order1:=xlAscending, Header:=xlYes

    End If

End Sub
```

Listing 3.4 Eine automatische Sortierung wird angestoßen, wenn ein Eintrag in Spalte A vorgenommen wird.

Im Kopf des Ereignisses sehen Sie die Variable `Target` vom Typ `Range`. Über diese Variable weiß Excel genau, in welcher Zelle der Anwender eine Eingabe vorgenommen

hat. Falls die Eingabe in Spalte A vorgenommen wurde, liefert die Prüfung über die Eigenschaft `Column` die Spaltennummer 1. In diesem Fall wenden Sie die Methode `Sort` an, um die belegten Zellen aus Spalte A zu sortieren. Wichtig bei der Sortierung sind folgende Parameter der Methode:

- `Key1`: Über diesen Parameter legen Sie fest, nach welcher Spaltenüberschrift sortiert werden soll. Hier spricht man auch vom *Sortierkriterium*.
- `Order1`: Bei diesem Parameter legen Sie fest, ob aufsteigend oder absteigend sortiert wird. Die Sortierreihenfolge wird über die beiden Konstanten `xlAscending` bzw. `xlDescending` geregelt.
- `Header`: Dieser Parameter bestimmt, ob es in der zu sortierenden Liste eine Überschrift gibt oder nicht. Mögliche Konstanten dabei sind: `xlYes`, `xlNo` und interessanterweise `xlGuess`. Beim Gebrauch der letzten Konstante überlassen Sie es Excel, einzuschätzen, ob eine Überschrift verfügbar ist oder nicht.

3.2.5 Spalten mit Wochenenden kennzeichnen

Bei der folgenden Aufgabe liegt in Tabelle4 in der ersten Zeile eine Datumsleiste vor. Die Aufgabe besteht darin, zunächst zu prüfen, ob in der jeweiligen Zelle ein Datum steht. Wenn ja, dann erfolgt eine zweite Prüfung, ob es sich bei dem Datum um ein Wochenende handelt. Sehen Sie sich dazu das Makro aus Listing 3.5 an.

```
Sub WochenendenKennzeichnen()
 Dim lngSpalte As Long
 Dim lngSpalteMax As Long

 With Tabelle4

  .Rows(1).Interior.ColorIndex = xlColorIndexNone
  lngSpalteMax = .Cells(1, .Columns.Count).End(xlToLeft).Column

   For lngSpalte = 1 To lngSpalteMax

    If IsDate(.Cells(1, lngSpalte).Value) Then

      If Weekday(.Cells(1, lngSpalte).Value, vbMonday) > 5 Then
          .Cells(1, lngSpalte).Interior.ColorIndex = 4
      End If

    End If

   Next lngSpalte
```



```
  End With

End Sub
```

Listing 3.5 Alle Wochenendspalten soll farbig hinterlegt werden.

Deklarieren Sie zu Beginn des Makros aus Listing 3.5 zwei Variablen vom Typ `Long`. Die Variable `lngSpalte` wird die Schleifensteuerung übernehmen und gleichermaßen auf die jeweils zu verarbeitende Zelle zeigen. Die Variable `lngSpalteMax` wird im Verlaufe des Makros die letzte belegte Zelle in der ersten Zeile erkunden und von dieser Zelle die Spaltennummer aufnehmen.

Entfärben Sie die erste Zeile über die Eigenschaft `ColorIndex`, der Sie die Konstante `xlColorIndexNone` zuweisen. Danach ermitteln Sie, wie gerade schon angedeutet, die letzte verwendete Zelle in der ersten Zeile, fragen mit der Eigenschaft `Column` die dazugehörige Spaltennummer ab und speichern diese Information in der Variablen `lngSpalteMax`. In der anschließenden `For ... Next`-Schleife bildet diese Variable das Schleifenabbruchkriterium. Innerhalb der Schleife prüfen Sie mit Hilfe der Anweisung `If` und der Funktion `IsDate`, ob in der jeweiligen Zelle überhaupt ein gültiges Datum steht. Wenn ja, dann erfolgt eine zweite Prüfung über die Funktion `Weekday`. Dieser Funktion übergeben Sie das jeweilige Datum. Im zweiten Argument dieser Funktion müssen Sie über die Konstante `vbMonday` angeben, dass die Woche bei uns mit dem Montag beginnt – für uns selbstverständlich, für Amerikaner fängt die eigentliche Woche mit dem Sonntag an. Wenn also die Woche mit dem Montag als erstem Tag der Woche beginnt, dann liefert uns die Funktion `Weekday` für den Montag die Zahl `1` und für den Sonntag den Wert `7` zurück. Daher fragen Sie mit der `If`-Anweisung ab, ob die Funktion `Weekday` für das jeweilige Datum einen Rückgabewert größer 5 zurückgibt. In diesem Fall handelt es sich um ein Wochenende, und die Spalte wird über die Eigenschaft `ColorIndex` eingefärbt.

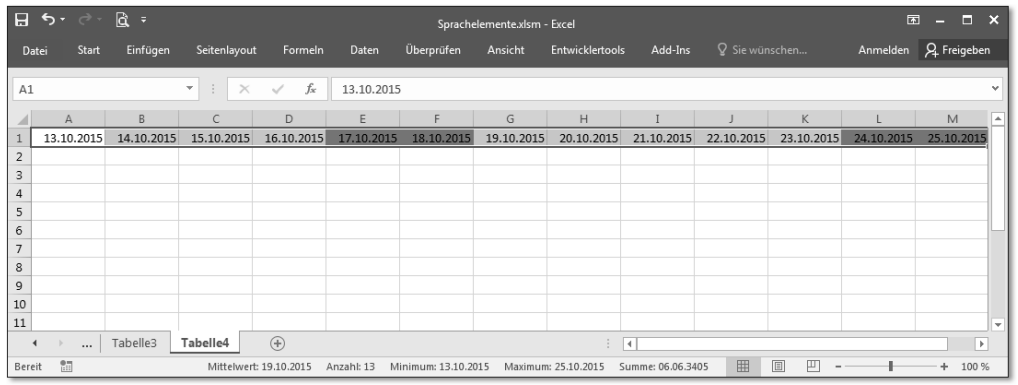


Abbildung 3.5 Alle Wochenenden wurden grün eingefärbt.

Neben der Funktion `IsDate` gibt es weitere Prüffunktionen, die Sie Tabelle 3.1 entnehmen können.

Funktion	Beschreibung
`IsEmpty`	Gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob eine Variable initialisiert wurde.
`IsArray`	Gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob eine Variable ein Datenfeld ist.
`IsDate`	Gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob ein Ausdruck in ein Datum umgewandelt werden kann.
`IsError`	Gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob ein Ausdruck ein Fehlerwert ist.
`IsNull`	Gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob ein Ausdruck keine gültigen Daten (`Null`) enthält.
`IsNumeric`	Gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob ein Ausdruck als Zahl ausgewertet werden kann.
`IsObject`	Gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob ein Bezeichner eine Objektvariable darstellt.

Tabelle 3.1 Die Prüffunktionen von Excel

3.3 Die Kurzform einer Bedingung

Neben der Verzweigung `If ... Then ... Else` gibt es eine weitere Möglichkeit, Werte zu überprüfen. Die Funktion lautet `IIf`.

Die Funktion `IIf` hat folgende Syntax:

```
IIf(expr, truepart, falsepart)
```

Mit dem Argument `expr` geben Sie den auszuwertenden Ausdruck an.

Das Argument `truepart` liefert den zurückgegebenen Wert oder Ausdruck, wenn `expr` den Wert `True` ergibt.

Das Argument `falsepart` stellt den zurückgegebenen Wert oder Ausdruck dar, wenn `expr` den Wert `False` liefert.

Diese Funktion wertet immer sowohl den Teil `truepart` als auch den Teil `falsepart` aus, auch dann, wenn nur einer von beiden Teilen zurückgegeben wird.

3.3.1 Den Doppelklick auf eine Zelle abfangen

In der folgenden Aufgabe wird in Tabelle1 der Doppelklick auf eine Zelle abgefangen. Dabei soll automatisch ein X gesetzt werden, wenn der Doppelklick auf eine leere Zelle in Spalte B durchgeführt wird. Steht in einer Zelle bereits der Buchstabe X, dann soll die Zelle wieder geleert werden.

Diese Aufgabe können Sie über ein Tabellenereignis lösen. Stellen Sie das Ereignis wie folgt ein:

1. Führen Sie im Projekt-Explorer der Entwicklungsumgebung einen Doppelklick auf den Tabellennamen Tabelle1 durch.
2. Sie gelangen dadurch direkt hinter die Tabelle. Wählen Sie im Codefenster auf der rechten Seite im ersten Dropdown oben den Eintrag `Worksheet` aus. Dadurch wird zunächst das Ereignis `Worksheet_SelectionChange` eingestellt.
3. Stellen Sie im zweiten Ereignis rechts daneben das Ereignis `Worksheet_BeforeDoubleClick` ein, und entfernen Sie danach das nicht benötigte Ereignis `Worksheet_SelectionChange`.
4. Komplettieren Sie den noch leeren Ereignisrahmen wie folgt:

```
Private Sub Worksheet_BeforeDoubleClick _
(ByVal Target As Range, Cancel As Boolean)

    If Target.Column = 2 Then
      Target.Value = IIf(Target.Value = "X", "", "X")
      Cancel = True
    End If

End Sub
```

Listing 3.6 Im Wechsel eine Zelle mit einem X versehen bzw. das X wieder löschen

Das Tabellenereignis `Worksheet_BeforeDoubleClick` hat zwei Argumente. Das Argument `Target` gibt Auskunft darüber, welche Zelle doppelt angeklickt wurde. Über das Argument `Cancel` kann die standardmäßig dem Doppelklick zugewiesene Aktion, den Wechsel in den Editiermodus der Zelle, aufgehoben werden.

Mit Hilfe der Anweisung `If` und der Eigenschaft `Column` wird abgefragt, ob der gerade durchgeführte Doppelklick auf eine Zelle in der zweiten Spalte erfolgte. Wenn ja, dann wird mit der `IIf` Anweisung geprüft, ob nicht bereits ein X in der Zelle steht. Wenn ja, dann wird die Zelle geleert, ansonsten eben mit einem X bestückt. Setzen Sie das Argument `Cancel` auf den Wert `True`, um zu verhindern, dass Excel in den Editiermodus der Zelle wechselt.

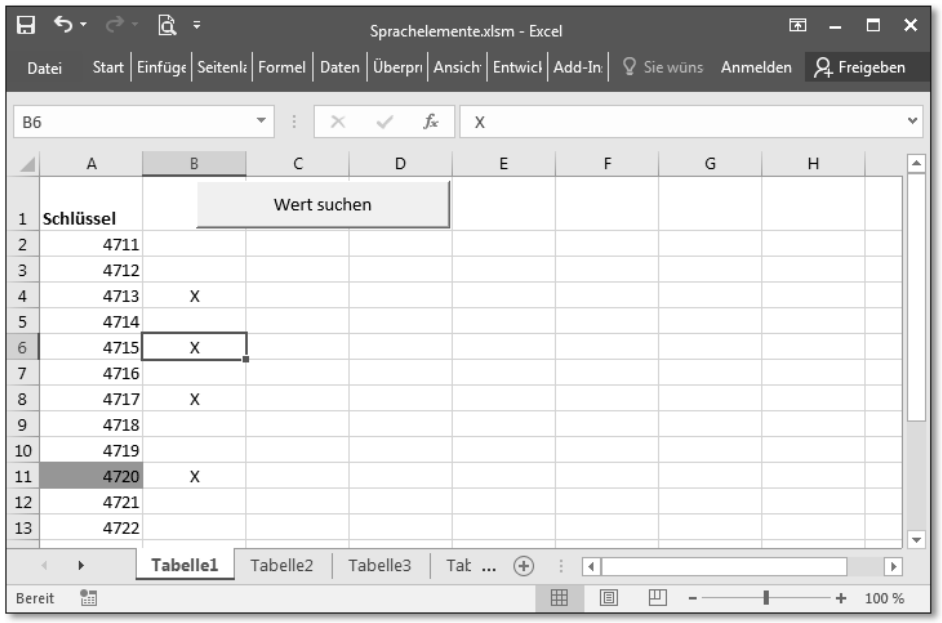


Abbildung 3.6 Ein toller Effekt – mit einem Doppelklick jeweils nach dem Füllen oder Leeren einer Zelle

3.4 Die Anweisung »Select Case« einsetzen

Mit Hilfe der Anweisung `Select Case` können Abfragen und Bedingungen leichter erfasst und übersichtlicher gelesen werden. Überhaupt bietet diese Anweisung, wenn es um einige Ausnahmen und Sonderfälle bei der Formulierung von Kriterien geht, im Vergleich zur `If`-Anweisung deutliche Vorteile.

Die Syntax dieser Anweisung lautet:

```
Select Case Ausdruck
Case Ausdrucksliste-n
  Anweisungen-n
Case Else
  elseAnweisungen
End Select
```

Unter dem Argument `Ausdruck` erfassen Sie einen beliebigen numerischen Ausdruck oder Zeichenfolgenausdruck, der ausgewertet werden soll. Im Argument `Ausdrucksliste-n` spezifizieren Sie die Abfrageoptionen näher. Dabei sind auch Vergleichsoperatoren, wie `To`, `Is` oder `Like` möglich.

Unter dem Argument `Anweisungen-n` geben Sie eine oder mehrere Anweisungen an, die ausgeführt werden sollen, wenn der abgefragte Ausdruck mit irgendeinem Element in `Ausdrucksliste-n` übereinstimmt.

Das Argument `elseAnweisungen` ist optional, insbesondere dann, wenn der abgefragte Ausdruck mit keinem Element im `Case`-Abschnitt übereinstimmen sollte.

Lernen Sie nun ganz konkret ein paar Beispiele kennen, wie Sie diese Anweisung in der Praxis einsetzen können.

3.4.1 Excel-Version abfragen

In der folgenden Aufgabe soll ermittelt werden, mit welcher Excel-Version Sie arbeiten. Für diese Aufgabe wird die Eigenschaft `Version` ausgewertet, die über einen numerischen Wert die aktuelle Excel-Installation identifiziert.

```
Sub ExcelVersionAbfragen()

 MsgBox Application.Version


  Select Case Left(Application.Version, 2)
      Case 10
        MsgBox "Excel 2002"
      Case 11
        MsgBox "Excel 2003"
      Case 12
        MsgBox "Excel 2007"
      Case 14
        MsgBox "Excel 2010"
      Case 15
        MsgBox "Excel 2013"
      Case 16
        MsgBox "Excel 2016"
      Case Else
        MsgBox "Unbekannte Version von Excel", vbInformation

   End Select

End Sub
```

Listing 3.7 Die eingesetzte Excel-Version ermitteln

Über die Funktion `Left` werden die ersten beiden Ziffern der Versionsnummer ausgewertet, die über die Eigenschaft `Version` abgefragt wurde. Innerhalb der `Select Case`-Anweisung wird die Versionsnummer überprüft. Trifft eine Bedingung zu, wird die entsprechende Meldung am Bildschirm ausgegeben, die angibt, um welche Excel-Version es sich handelt.

Info

Haben Sie es im Listing gesehen? Zwischen den beiden Versionen Excel 2007 und Excel 2010 wurde die Versionsnummer 13 übersprungen. Ob da wohl jemand abergläubisch ist?

3.4.2 Zahlenwerte prüfen

Im nächsten Beispiel werden Eingaben geprüft. Dabei soll ermittelt werden, in welchem Wertebereich die Eingabe vorgenommen wurde. Sehen Sie sich dazu das Makro aus Listing 3.8 an.

```
Sub ZahlUnWerteBereichAuswerten()
  Dim strEingabe As String

  strEingabe = InputBox("Geben Sie einen Wert ein!")

If Len(strEingabe) <> 0 Then

Select Case strEingabe

 Case 1 To 5
  MsgBox "Wert liegt zwischen 1 und 5"

 Case 6, 7, 8
  MsgBox "Wert ist entweder 6, 7 oder 8"

 Case 9 To 15
  MsgBox "Wert liegt zwischen 9 und 15"

 Case 16 To 100
  MsgBox "Wert liegt zwischen 16 und 100"

 Case Is > 100
 MsgBox "Wert liegt über 100"
```



```
  Case Else
   MsgBox "Es wurde kein gültiger Wert eingegeben!"

 End Select

End If

End Sub
```

Listing 3.8 In welchem Wertebereich liegt die eingegebene Zahl?

Wenden Sie die `Select Case`-Anweisung an, um die eingegebenen Werte zu überprüfen. In der ersten Abfrage wird kontrolliert, ob der eingegebene Wert im Bereich von 1 bis 5 liegt. Für die Prüfung eines Wertebereiches kann der Vergleichsoperator `To` eingesetzt werden. In der zweiten Abfrage werden einzelne Zahlenwerte durch Komma getrennt eingegeben. Wurde kein gültiger Zahlenwert eingegeben, kommt die Anweisung `Case Else` zum Tragen. Dieser Zweig würde beispielsweise dann angesteuert, wenn die eingegebene Zahl entweder 0 ist oder wenn es sich um eine negative Zahl handelt.

Reihenfolge der Zweige

Die Reihenfolge, in der Sie die einzelnen `Case`-Zweige anordnen, ist Ihnen frei überlassen. Wenn beispielsweise häufig Werte größer 100 eingegeben werden, dann kann der letzte Zweig aus Listing 3.8 auch als erste Bedingung eingesetzt werden. Dadurch wird die Geschwindigkeit in der Abarbeitung der Abfragen erhöht, da Zweig für Zweig von oben nach unten abgearbeitet wird. Wird eine Entsprechung gefunden, dann wird nach Erfüllung der Bedingung und Ausführen der damit definierten Aktion sofort direkt ans Ende der `Select Case`-Anweisung gesprungen.

Vergleichsoperator	Erklärung
<	kleiner als
<=	kleiner oder gleich
>	größer als
>=	größer oder gleich
=	gleich
<>	ungleich

Tabelle 3.2 Die Vergleichsoperatoren in Excel

3.4.3 Den Wochentag eines bestimmten Datums auslesen

Das Beispiel aus Listing 3.9 ermittelt, ob es sich beim aktuellen Tagesdatum um einen Werktag oder ein Wochenende handelt.

```
Sub WochentagErmitteln()

 MsgBox "Heute ist " & Date

  Select Case Weekday(Date, vbMonday)
   Case Is < 6
    MsgBox "Werktag"

   Case Else
     MsgBox "Wochenende"

  End Select

End Sub
```

Listing 3.9 Das aktuelle Datum auswerten

Sie sollten immer den am häufigsten vorkommenden Wert an den Beginn der `Select Case`-Struktur stellen. In diesem Bespiel wird der aktuelle Tag ausgewertet. Es ist wahrscheinlicher, dass der aktuelle Tag ein Werktag ist, als dass der aktuelle Tag auf ein Wochenende trifft. Da bei `Select Case` die Prüfung von Zweig zu Zweig erfolgt, ist es natürlich besser, wenn gleich der erste Vergleich erfüllt wird. In diesem Fall werden weitere Zweige nicht mehr abgearbeitet, und die Verarbeitung der Prozedur geht nach `End Select` weiter.

Die Funktion `Weekday` hat wie vorher schon beschrieben zwei Argumente. Im ersten Argument übergeben Sie der Funktion ein Datum. Dieses Datum können Sie beispielsweise über die Funktion `Date` einsteuern, die Ihnen das aktuelle Tagesdatum liefert. Im zweiten Argument teilen Sie der Funktion mit, dass die Woche mit dem Montag beginnt. Das klingt auf den ersten Blick selbstverständlich, aber bei der Standardeinstellung dieser Funktion gilt der Sonntag als erster Tag der Woche. Die Funktion `Weekday` liefert Ihnen einen Wert zwischen 1 und 7 zurück. Wenn die Woche mit dem Montag beginnt, dann ist der Tag 1 der Woche der Montag und Tag 7 eben der Sonntag.

3.4.4 Benotungen über einen Autotext durchführen

Bei der folgenden Aufgabe liegt TABELLE5 wie in Abbildung 3.7 gezeigt vor. Darin finden Sie für die einzelnen Fächer die dazugehörigen Noten.

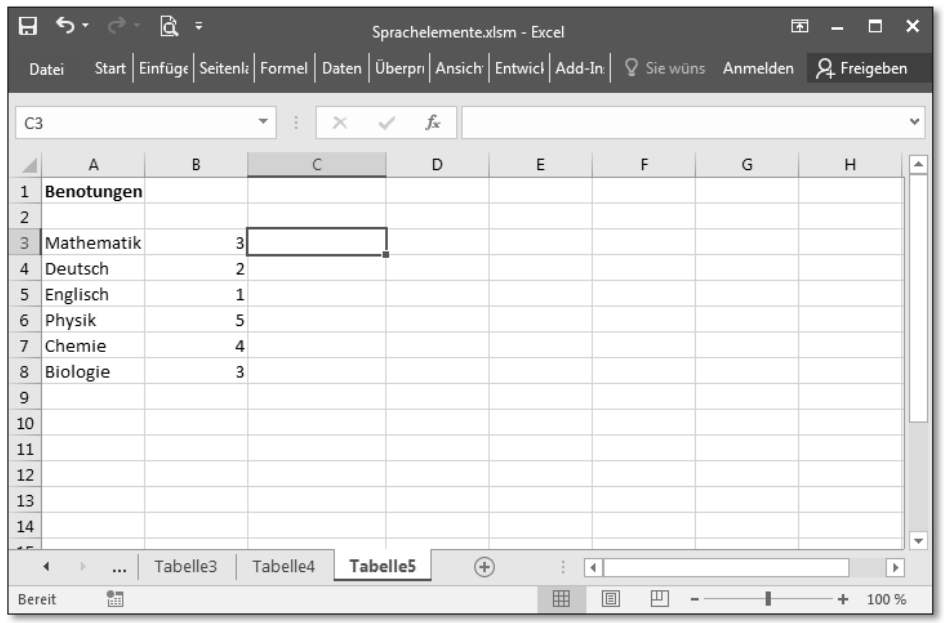


Abbildung 3.7 Fächer mit dazugehörigen Noten

In Spalte C soll jetzt eine Übersetzung der Noten in Texte stattfinden. Schreiben Sie dazu eine eigene benutzerdefinierte Funktion. Sehen Sie sich die Funktion aus Listing 3.10 an.

```
Function Benotung(rngZelle As Range)

  Select Case rngZelle.Value
      Case Is = 1: Benotung = "Sehr gut"
      Case Is = 2: Benotung = "Gut"
      Case Is = 3: Benotung = "Befriedigend"
      Case Is = 4: Benotung = "Ausreichend"
      Case Is = 5: Benotung = "Mangelhaft"
      Case Is = 6: Benotung = "Ungenügend"
      Case Else: Benotung = "keine gültige Zensur"
  End Select

End Function
```

Listing 3.10 Die Noten in Text aufschlüsseln

Wie Sie in Listing 3.10 sehen, wird die `Select Case`-Anweisung auch derart eingesetzt, dass über den Doppelpunkt ein Zeilenumbruch erspart werden kann. Dies ist aber nur dann sinnvoll, wenn es wirklich nur eine Aktion ist, die durchgeführt werden soll, wenn ein Zweig angesteuert wird.

Der Aufruf dieser Funktion kann direkt in der Zelle erfolgen. Markieren Sie dazu den Zellenbereich C3:C8, geben Sie die Formel `=Benotung(B3)` ein, und schließen Sie sie über die Tastenkombination Strg + ↵ ab.

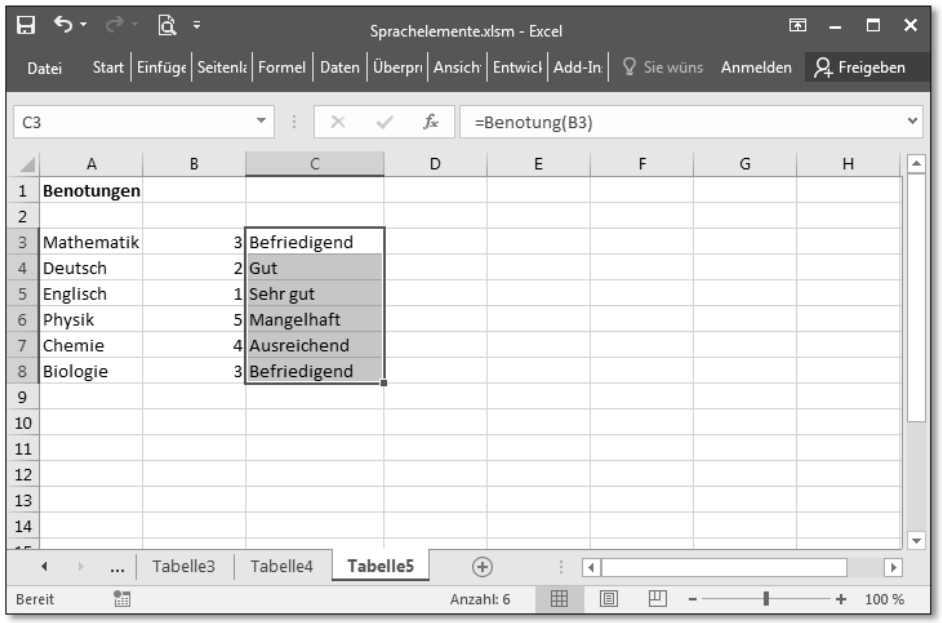


Abbildung 3.8 Ein Zeugnis mit Noten in Zahlen und in Textform

Info

Lernen Sie mehr über Funktionen in Kapitel 8, »Eigene Funktionen, reguläre Ausdrücke und API-Funktionen«.

3.5 Die »Enum«-Anweisung

Über die `Enum`-Anweisung generieren Sie eine Aufzählungsliste. `Enum` hat folgende Syntax:

```
[Public | Private] Enum Name
Elementname [= Konstantenausdruck]
Elementname [= Konstantenausdruck]
End Enum
```

Die Syntax der `Enum`-Anweisung besteht aus folgenden Bestandteilen:

- `Public`: optional. Legt fest, dass der `Enum`-Typ im gesamten Projekt sichtbar sein soll. `Enum`-Typen sind standardmäßig `Public`.
- `Private`: optional. Legt fest, dass der `Enum`-Typ nur innerhalb des Moduls sichtbar sein soll, in dem er angezeigt wird.
- `Name`: erforderlich. Der Name des `Enum`-Typs. `Name` muss ein zulässiger Visual-Basic-Kennzeichner sein und wird beim Deklarieren von Variablen oder Parametern vom `Enum`-Typ als Typ angegeben.
- `Elementname`: erforderlich. Ein zulässiger Visual-Basic-Kennzeichner, der den Namen angibt, der für ein konstituierendes Element des `Enum`-Typs verwendet werden soll.
- `Konstantenausdruck`: optional. Repräsentiert den Wert des Elements.

3.5.1 Umsatz klassifizieren mit »Enum«

Im Beispiel aus Listing 3.11 wird eine Umsatzgruppe definiert, die aus drei »Untervariablen« besteht. Diese können Sie nach der Deklaration in einer Prozedur direkt ansprechen und ausgeben.

```
Public Enum MeinUmsatz
    Klein = 0
    Mittel = 1
    Groß = 2
End Enum

Dim Umsatz As MeinUmsatz

Sub VariableFüllenUndAusgeben()

 Select Case Tabelle6.Range("A1").Value
  Case Is >= 5000
    Umsatz = Groß
  Case 1000 To 5000
    Umsatz = Mittel
  Case Is < 1000
    Umsatz = Klein
  End Select

  MsgBox "Der Umsatz gehört in die Klasse " & Umsatz

End Sub
```

Listing 3.11 Die »Enum«-Anweisung für die Klassifizierung von Umsätzen einsetzen

Deklarieren Sie zu Beginn außerhalb des Makros aus Listing 3.10 eine Aufzählung, indem Sie Texte mit Zahlen gleichsetzen und in die `Enum`-Struktur packen.

Danach deklarieren Sie eine Variable, die genau auf diese Struktur verweist. Im Makro selbst werten Sie Zelle A1 in Tabelle6 über die `Select Case`-Anweisung aus. Je nach Wert füllen Sie die Variable `Umsatz`. Beim Editieren werden Ihnen die drei möglichen Aufzählungen elegant in einem Dropdown angeboten. Geben Sie am Ende die ermittelte Zuordnung über die Funktion `MsgBox` am Bildschirm aus.

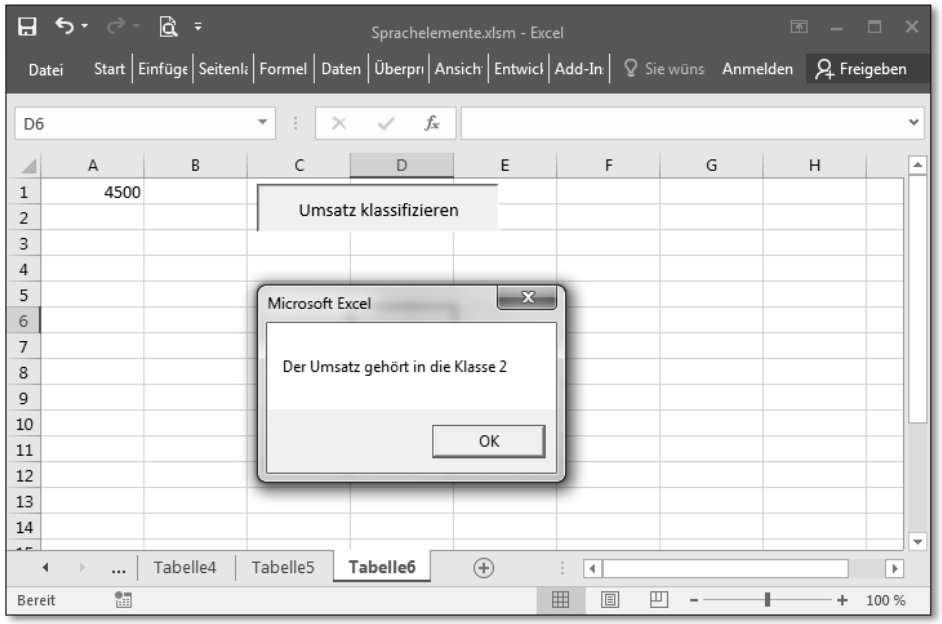


Abbildung 3.9 Der erfasste Umsatz wurde in die dazugehörige Gruppe eingegliedert.

3.6 Schleifen erstellen und verstehen

Eine Schleife ist ein Vorgang, der immer wieder gleich abläuft und so oft wiederholt wird, bis er beendet wird.

In Excel werden Schleifen dazu verwendet, Abläufe mehrmals hintereinander durchzuführen. Die Schleifen werden so lange durchlaufen, bis eine oder mehrere Bedingungen zutreffen, die dann einen Abbruch der Schleife bewirken. Je nach verwendeter Schleife findet die Abbruchprüfung am Anfang oder am Ende der Schleife statt.

Eine typische Schleife in Excel wäre das zeilenweise Abarbeiten einer Tabelle von oben nach unten. Für jede Zeile werden in der Schleife bestimmte Prüfungen durchgeführt. Eine typische Prüfung wäre beispielsweise der Vergleich einer Zelle mit einem Vorgabewert. Die Schleife sorgt dafür, dass jede Zeile gleichermaßen verarbeitet wird, bis die Verarbeitung an der letzten gefüllten Zelle der Tabelle angekommen ist. Hier erfährt die Schleife dann einen Abbruch. Manche Schleifen können diesen »normalen« Abbruch auch innerhalb der Schleife quasi zwischendurch erfahren, wenn die Verarbeitung beispielsweise auf ein Problem stößt, bei dem es besser ist, die Schleife vorzeitig zu beenden.

In Excel gibt es eine ganze Anzahl verschiedener Schleifentypen, die Sie auf den nächsten Seiten kennenlernen werden.

3.6.1 Die »For ... Next«-Schleife

Sie können die Schleife `For ... Next` verwenden, um einen Block von Anweisungen eine unbestimmte Anzahl von Wiederholungen auszuführen. `For ... Next`-Schleifen verwenden eine Zählervariable, deren Wert mit jedem Schleifendurchlauf erhöht oder verringert wird. Sie müssen daher nicht daran denken, den Zähler selbst hoch- oder herunterzusetzen.

Die Syntax dieser Schleife lautet:

```
For Zähler = Anfang To Ende [Step Schritt]
 [Anweisungen]
 [Exit For]
 [Anweisungen]
Next [Zähler]
```

Das Argument `Zähler` ist erforderlich und besteht aus einer numerischen Variablen, die als Schleifenzähler dient.

Das Argument `Anfang` repräsentiert den Startwert von `Zähler`.

Mit dem Argument `Ende` wird der Endwert des Zählers festgelegt. Das Argument `Schritt` ist optional. Hier können Sie den Wert bestimmen, um den `Zähler` bei jedem Schleifendurchlauf verändert wird. Falls kein Wert angegeben wird, ist die Voreinstellung 1.

Unter `Anweisungen` stehen eine oder mehrere Anweisungen zwischen `For` und `Next`, die so oft wie angegeben ausgeführt werden.

Innerhalb einer Schleife können Sie eine beliebige Anzahl von `Exit For`-Anweisungen setzen, die eine alternative Möglichkeit darstellen, die Schleife vorzeitig zu verlassen.

Lernen Sie jetzt einige typische Aufgabenstellungen für diese Schleife kennen.

Die Farbpalette von Excel auslesen

Im Beispiel aus Listing 3.12 werden in Tabelle7 alle verfügbaren Farben, die Excel für den Hintergrund einer Zelle zur Verfügung stellt, dargestellt.

```
Sub FarbenErmitteln()
  Dim wksBlatt As Worksheet
  Dim intZ As Integer

  Set wksBlatt = Tabelle7

  For intZ = 1 To 56

    With wksBlatt
      .Cells(intZ, 1).Value = intZ
      .Cells(intZ, 2).Interior.ColorIndex = intZ
    End With

  Next intZ

End Sub
```

Listing 3.12 Die 56 Standardfarben von Excel in einer Tabelle ausgeben

Zunächst wird eine Zählvariable vom Typ `Integer` deklariert. Danach deklarieren Sie die Objektvariable `wksBlatt` mit dem Datentyp `Worksheet`. Geben Sie dann über die Anweisung `Set` bekannt, dass Sie mit TABELLE7 arbeiten möchten. Setzen Sie eine Schleife auf, die genau 56 Mal durchlaufen wird, da es nur 56 Standardfarben für den Hintergrund einer Zelle gibt. Wir haben hier also eine vorher festgelegte Wiederholungszahl für die Schleife. Innerhalb der Schleife können Sie über die `With`- Anweisung die Schreibarbeit ein wenig reduzieren, da die Anweisungen für die aktive Tabelle gelten.

Über die `Cells`-Auflistung werden die Zellen gezielt angesteuert und von oben nach unten gefüllt. Diese Auflistung hat genau zwei Argumente: Das erste Argument gibt die Zeilennummer an, das zweite enthält die Spaltennummer. In diesem Beispiel werden die Informationen zeilenweise in die Spalten A (= `1`) und B (= `2`) geschrieben. Die erste Spalte zeigt die eindeutige Farbnummer. Bei Spalte B wird genau diese Farbnummer genutzt, um den Hintergrund der Zelle (`Interior`) über die Eigenschaft `ColorIndex` zu füllen.

Hinweis

Wir haben im Beispiel aus Listing 3.12 die Zählvariable doppelt verwendet. Zum einen wird dadurch die Zeilennummer verwaltet (von Zeile 1 bis Zeile 56), zum anderen werden die verfügbaren Farben (von 1 bis 56) den Zellen zugewiesen.

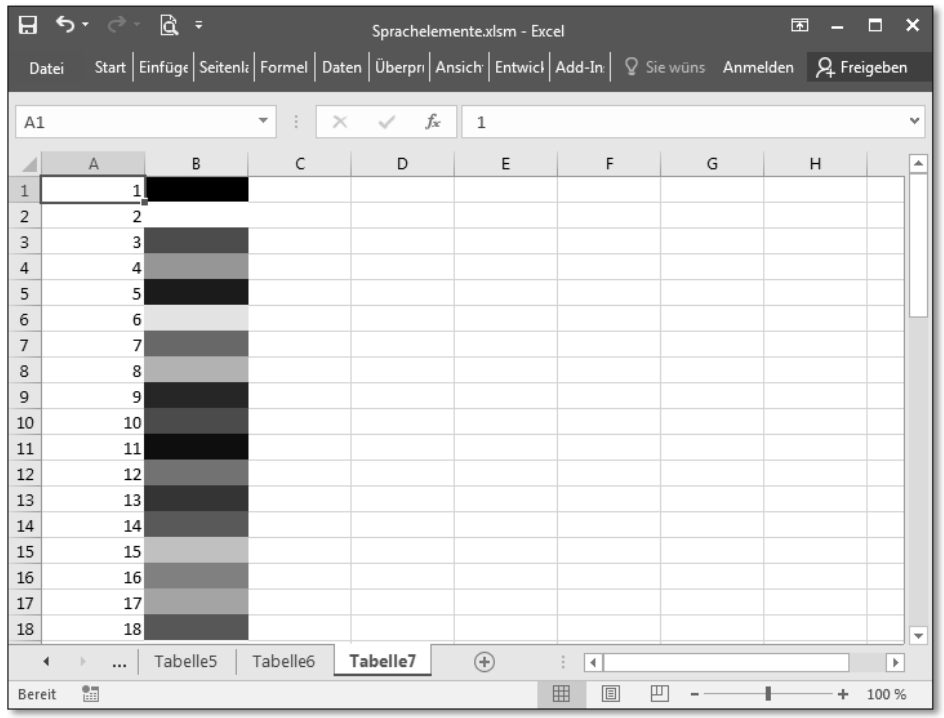


Abbildung 3.10 Die 56 Standardfarben von Excel auf einen Blick

Die Schablone für eine Verarbeitung von Zeilen von oben nach unten

Die allermeisten Aufgaben in Excel befassen sich direkt mit Tabellen, die Zeile für Zeile von oben nach unten abgearbeitet werden. Daher möchte ich Ihnen die Mutter aller Schablonen für eine Schleife vorstellen, über die Sie die meisten Aufgaben lösen können. Diese Schablone ist in 5 Schritte eingeteilt, und das Schöne daran ist, dass die ersten 4 Schritte immer gleich sind, egal, was auch immer Sie mit der Tabelle anstellen. Das bedeutet, dass Sie sich zukünftig voll und ganz auf Schritt 5, die eigentliche Aufgabe, konzentrieren können.

Im Makro aus Listing 3.13 sehen Sie die Schablone für diese Schleife.

```
Sub Schablone_VonObenNachUnten()
'Verwendung: Zeilenverarbeitung
'Schritt 1: Deklaration von Variablen
 Dim lngZeile As Long 'Repräsentiert den Zähler f. die Schleife und die Zeile
 Dim lngZeileMax As Long 'Repräsentiert die letzte belegte Zeile

  'Schritt 2: Festlegen der Verarbeitungstabelle
  With Tabelle8
```

```
    'Schritt 3: Ermitteln der letzten verwendeten Zeile
    lngZeileMax = .UsedRange.Rows.Count

    'Schritt 4: Aufsetzen der Schleife
      For lngZeile = 2 To lngZeileMax

       'Schritt 5: Eigentliche Aufgabe

      Next lngZeile

  End With

End Sub
```

Listing 3.13 Die Schablone für eine zeilenweise Verarbeitung einer Tabelle von oben nach unten

In Schritt 1 deklarieren Sie zwei Variablen vom Typ `Long` mit Hilfe der Anweisung `Dim`. Danach steht Ihnen ein reservierter Platz mit dem Namen `Zeile` im Arbeitsspeicher zur Verfügung. Beide Variablen haben jetzt den Wert 0, das heißt, die Variablen müssen nicht gesondert initialisiert werden. Über die Variable `Zeile` steuern Sie später die Schleife und zeigen direkt auf die zu verarbeitende Zeile. In der Variablen `lngZeileMax` speichern Sie nachher die Zeilennummer der zuletzt benutzten Zeile in der Tabelle.

In Schritt 2 legen Sie die zu verarbeitende Tabelle über die Anweisung `With` fest. Immer wenn Sie danach auf die so festgelegte Tabelle zugreifen möchten, reicht es, wenn Sie anstatt des vollen Namens der Tabelle einen Punkt setzen. Excel ergänzt dann den bei `With` definierten Tabellennamen zur Laufzeit des Makros.

In Schritt 3 ermitteln Sie die Zeilennummer der letzten belegten Zeile der Tabelle. Dazu müssen Sie wissen, dass jede gefüllte Tabelle einen benutzten Bereich hat. Dieser Bereich kann über die Eigenschaft `UsedRange` abgefragt werden. Bei der Ermittlung dieses Bereiches sucht Excel nach der letzten gefüllten Zelle der Tabelle nach rechts und nach unten. Nehmen wir einmal an, es stünde jeweils ein Wert in Zelle A1 und D10. Dann wäre der benutzte Bereich A1:D10. Sie können das ruhig einmal ausprobieren und dann in das Direktfenster der Entwicklungsumgebung folgende Zeile eingeben, um den benutzten Bereich der aktiven Tabelle abzufragen:

```
?activesheet.usedrange.address
```

Der benutzte Bereich einer Tabelle umfasst eine bestimmte Anzahl von Zeilen, die Sie über den Befehl `Rows.Count` abfragen und in der Variablen `lngZeileMax` »parken«. Sie wissen also jetzt, wie lange Sie die Schleife anschließend nach unten »jagen« müssen.

In Schritt 4 wird die Schleife aufgesetzt. In der Regel beginnt diese in Zeile 2, wenn angenommen wird, dass die Überschrift nicht verarbeitet werden soll. Die Schleife

fängt also bei der zweiten Zeile an und arbeitet sich Zeile für Zeile nach unten durch, bis die letzte Zeile (=`lngZeileMax`) erreicht wird. Über die Anweisung `Next lngZeile` wird die Variable `lngZeile` automatisch um den Wert 1 erhöht. Damit zeigen Sie bei jedem Schleifendurchlauf jeweils auf die nächste Zeile.

In Schritt 5 findet die eigentliche Aufgabe statt. Hier können beispielsweise Prüfungen erfolgen, die eine Zelle je nach dem enthaltenen Wert formatieren.

Die Schablone für eine Verarbeitung von Zeilen von unten nach oben

Eine `For ... Next`-Schleife können Sie in einer Tabelle auch von unten nach oben Zeile für Zeile durchlaufen lassen. Sie fragen sich vielleicht jetzt: »Warum sollte ich das denn überhaupt wollen?«

Bei dieser umgekehrten Laufrichtung handelt es sich um eine Notwendigkeit beim Löschen von Zeilen aus einer Tabelle. Nur wenn Sie bei diesem Vorhaben unten anfangen und oben aufhören, sind Sie erfolgreich.

Die Begründung dafür liegt am Verhalten von Excel, denn wenn Sie eine Zeile aus einer Tabelle löschen, rutschen die darunterliegenden Zeilen eine Zeile nach oben. Das würde beim standardmäßig üblichen Abarbeiten der Tabelle von oben nach unten zu einer Veränderung des Zählers führen. Excel würde quasi jeweils eine Zeile überspringen und nicht sauber Zeilen löschen, wenn gleich mehrere zu löschende Zeilen direkt untereinander lägen.

Aber auch bei anderen Aufgabenstellungen kann eine Abarbeitung einer Tabelle vom Ende zum Beginn der Tabelle sicherlich reizvoll sein.

Sehen Sie sich jetzt einmal den Makrorahmen aus Listing 3.14 an, der eine Tabelle von unten nach oben abarbeitet.

```
Sub Schablone_VonUntenNachOben()
  'In der Regel beim Löschen von Zeilen in Verwendung
  'Schritt 1: Deklaration von Variablen
  Dim lngZeile As Long
  Dim lngZeileMax As Long

 'Schritt 2: Festlegen der Verarbeitungstabelle
  With Tabelle8

    'Schritt 3: Ermitteln der letzten verwendeten Zeile
    lngZeileMax = .UsedRange.Rows.Count

    'Schritt 4: Aufsetzen der Schleife
    For lngZeile = lngZeileMax To 2 Step -1
```

```
    'Schritt 5: Löschkriterium festlegen

   Next lngZeile

  End With

End Sub
```

Listing 3.14 Die Schablone für eine zeilenweise Verarbeitung einer Tabelle von unten nach oben

In Schritt 1 deklarieren wir zwei Variablen vom Typ `Long` mit Hilfe der Anweisung `Dim`. Danach steht uns ein reservierter Platz mit dem Namen `Zeile` im Arbeitsspeicher zur Verfügung.

In Schritt 2 legen wir die zu verarbeitende Tabelle über die Anweisung `With` fest.

In Schritt 3 ermitteln wir die Zeilennummer der letzten belegten Zeile der Tabelle.

In Schritt 4 wird die Schleife aufgesetzt. Dabei beginnen wir bei der letzten Zeile der Tabelle und enden vor der Überschrift der Tabelle. Damit wir von unten nach oben kommen, müssen wir die Schrittweite auf den Wert `-1` setzen. Damit wird bei jedem Schleifendurchlauf der Wert 1 von der Variablen `Zeile` abgezogen.

In Schritt 5 legen wir das Löschkriterium fest. Als Löschkriterien könnten wir beispielsweise doppelte Werte, leere Zellen oder Zellen, die bestimmte Inhalte aufweisen, heranziehen.

Die Schablone für eine Verarbeitung von Spalten von links nach rechts

Die beiden Laufrichtungen einer Schleife von oben nach unten und von unten nach oben sind jetzt abgehandelt. Wenn es um die Verarbeitung von Spalten geht, dann kommt der Schleifenrahmen aus Listing 3.15 zum Einsatz.

```
Sub Schablone_VonLinksNachRechts()
  'Spaltenverarbeitung
  'Schritt 1: Deklaration von Variablen
  Dim lngSpalte As Long   'Repräsentiert die zu verarbeitende Spalte
  Dim lngSpalteMax As Long 'Repräsentiert die letzte gefüllte Spalte
'Schritt 2: Festlegen der Verarbeitungstabelle
  With Tabelle12

   'Schritt 3: Ermitteln der letzten gefüllten Spalte
   lngSpalteMax = .UsedRange.Columns.Count

   'Schritt 4: Aufsetzen der Schleife
```



```
  For lngSpalte = 1 To lngSpalteMax

   'Schritt 5: Eigentliche Aufgabe

  Next lngSpalte

 End With

End Sub
```

Listing 3.15 Die Schablone für eine spaltenweise Verarbeitung einer Tabelle von links nach rechts

In Schritt 1 deklarieren Sie zwei Variablen vom Typ `Long` mit Hilfe der Anweisung `Dim`. Danach steht Ihnen ein reservierter Platz mit dem Namen `lngSpalte` im Arbeitsspeicher zur Verfügung. Beide Variablen haben jetzt den Wert 0. Über die Variable `lngSpalte` steuern Sie später die Schleife und zeigen direkt auf die zu verarbeitende Spalte. In der Variablen `lngSpalteMax` speichern Sie nachher die Spaltennummer der letzten benutzten Spalte in der Tabelle.

In Schritt 2 legen Sie die zu verarbeitende Tabelle über die Anweisung `With` fest.

In Schritt 3 ermitteln Sie die Spaltennummer der letzten belegten Spalte der Tabelle. Dazu müssen Sie wissen, dass jede gefüllte Tabelle einen benutzten Bereich hat. Dieser Bereich kann über die Eigenschaft `UsedRange` abgefragt werden. Dieser benutzte Bereich besteht aus einer bestimmten Anzahl von Spalten, die Sie über die Funktion `Count` zählen und in der Variablen `lngSpalteMax` zwischenspeichern können.

In Schritt 4 setzen Sie die Schleife auf, die bei der ersten Spalte beginnt und sich dann nach rechts Spalte für Spalte durcharbeitet, bis die letzte Spalte (= `lngSpalteMax`) erreicht ist.

In Schritt 5 formulieren Sie die eigentliche Aufgabe der Schleife. Diese Aufgabe ist in der Schablone noch ausgespart.

Die Schablone für eine Verarbeitung von Spalten von rechts nach links

Diese Schablone aus Listing 3.16 wird dann eingesetzt, wenn es darum geht, bestimmte Spalten aus einer Tabelle zu entfernen. Was schon für die Löschung von Zeilen galt, trifft auch auf die Löschung von Spalten 1 : 1 zu.

```
Sub Schablone_VonRechtsNachLinks()
  'bei der Löschung von Spalten in Verwendung
  'Schritt 1: Deklaration von Variablen
  Dim lngSpalte As Long  'Repräsentiert die verarbeitende Spalte
  Dim lngSpalteMax As Long 'Repräsentiert die letzte gefüllte Spalte
'Schritt 2: Festlegen der Verarbeitungstabelle
```

```
  With Tabelle13
   'Schritt 3: Ermitteln der letzten gefüllten Spalte
   lngSpalteMax = .UsedRange.Columns.Count

   'Schritt 4: Aufsetzen der Schleife
   For lngSpalte = lngSpalteMax To 1 Step -1

    'Schritt 5: Eigentliche Aufgabe

   Next lngSpalte

  End With
End Sub
```

Listing 3.16 Die Schablone für eine spaltenweise Verarbeitung einer Tabelle von rechts nach links (Löschen von Spalten)

Da die Schritte 1 bis 3 dieselben wie beim Rahmen für die Verarbeitung einer Tabelle von links nach rechts sind, steigen wir in Schritt 4 ein. Der Schleifenzähler wird auf die letzte Spalte der Tabelle gesetzt, und die Schleife arbeitet sich Spalte für Spalte von rechts nach links durch.

Praxis, Praxis, Praxis

Auf den folgenden Seiten werden Sie den Einsatz dieser vier vorgestellten Schablonen anhand einiger praktischer Beispiele sehen.

Werte oberhalb eines Referenzwertes aufspüren

Bei der folgenden Aufgabenstellung werden die Werte in Spalte B von TABELLE9 ausgewertet. Dabei sollen diese Werte immer mit dem Vorgabewert aus Zelle E1 verglichen werden und je nach Wert entsprechend eingefärbt werden.

Verwenden Sie für diese Aufgabe die vorher erstellte Schablone *Schablone_VonObenNachUnten*, und passen Sie sie an. Das könnte dann so wie in Listing 3.17 gezeigt aussehen.

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

   With Tabelle9

    lngZeileMax = .UsedRange.Rows.Count
```



```
    For lngZeile = 2 To lngZeileMax

     If .Range("B" & lngZeile).Value >= .Range("E1").Value Then
       .Range("B" & lngZeile).Interior.ColorIndex = 4
      Else
       .Range("B" & lngZeile).Interior.ColorIndex = xlColorIndexNone
     End If

    Next lngZeile

  End With

End Sub
```

Listing 3.17 Werte oberhalb des Referenzwertes werden automatisch eingefärbt.

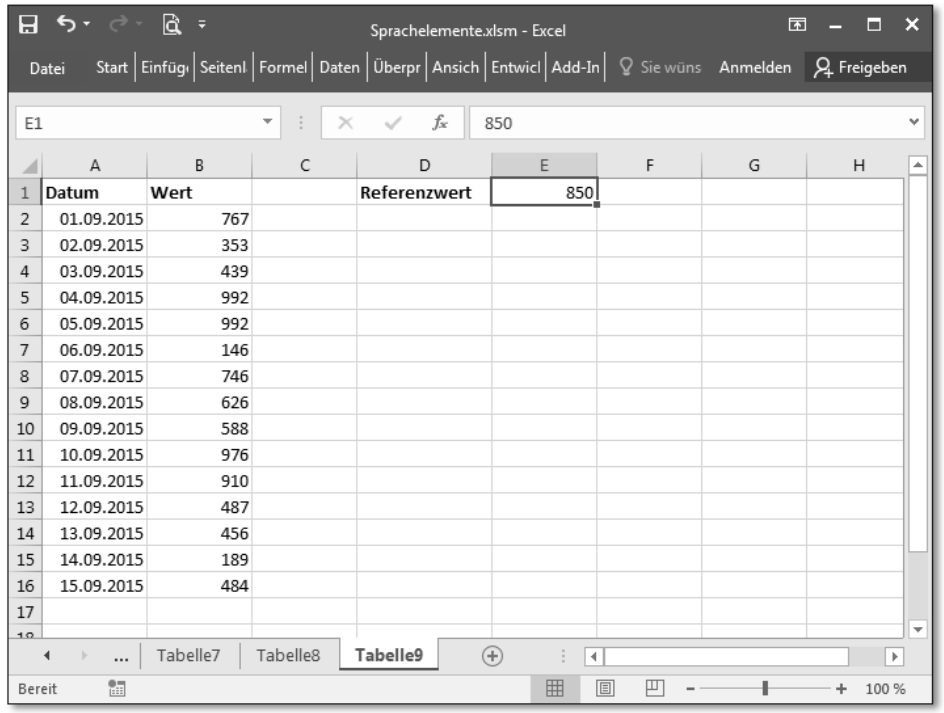
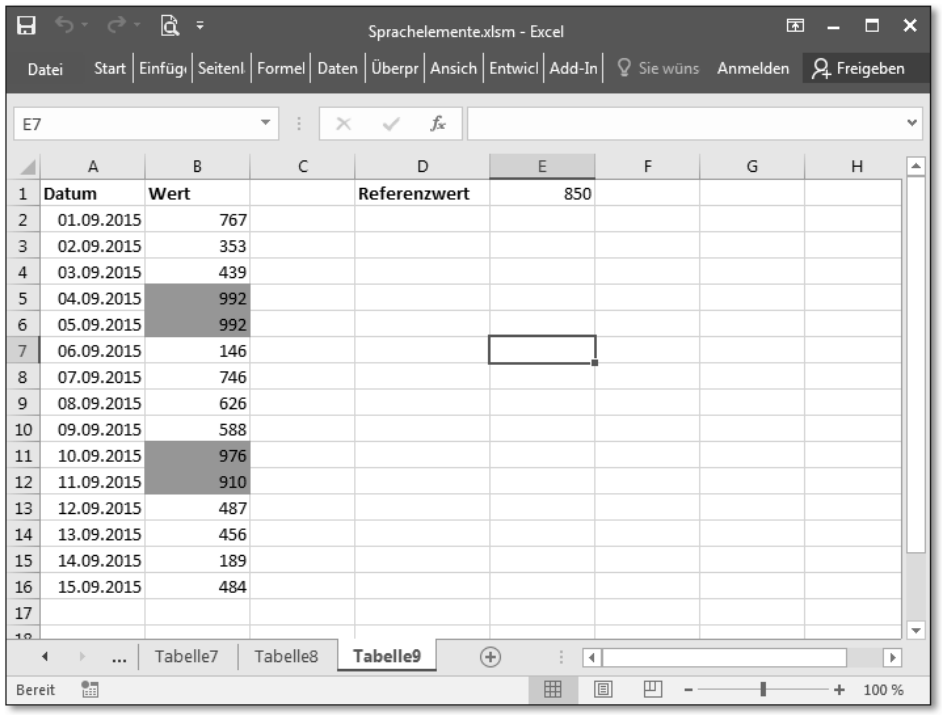


Abbildung 3.11 Die Ausgangssituation – welche Werte liegen über dem Referenzwert aus Zelle E1?

Da ich die Schritte 1 bis 4 bereits beim Rahmen in Listing 3.13 beschrieben habe, können wir gleich mitten in die Schleife aus Listing 3.17 gehen und die dort verwendeten Befehle besprechen.

Mit einer `If`-Anweisung prüfen Sie den Wert einer jeden Zelle aus Spalte B, die Sie ja über die Schleife von oben nach unten durchlaufen. Ist dieser Wert größer als oder gleich dem in Zelle E1, dann färben Sie den Hintergrund der Zelle, indem Sie dem Objekt `Interior` über die Eigenschaft `ColorIndex` den Wert `4` zuweisen.



	A	B	C	D	E	F	G	H
1	Datum	Wert		Referenzwert	850			
2	01.09.2015	767						
3	02.09.2015	353						
4	03.09.2015	439						
5	04.09.2015	992						
6	05.09.2015	992						
7	06.09.2015	146						
8	07.09.2015	746						
9	08.09.2015	626						
10	09.09.2015	588						
11	10.09.2015	976						
12	11.09.2015	910						
13	12.09.2015	487						
14	13.09.2015	456						
15	14.09.2015	189						
16	15.09.2015	484						
17								

Abbildung 3.12 Alle Werte oberhalb des Referenzwertes wurden gekennzeichnet.

Duplikate in einer Liste kennzeichnen

Beim folgenden Beispiel liegt eine Liste mit einigen doppelten Werten vor. Ihre Aufgabe besteht nun darin, diese doppelten Werte aufzuspüren und zu kennzeichnen. Sehen Sie sich dazu einmal Abbildung 3.13 an.

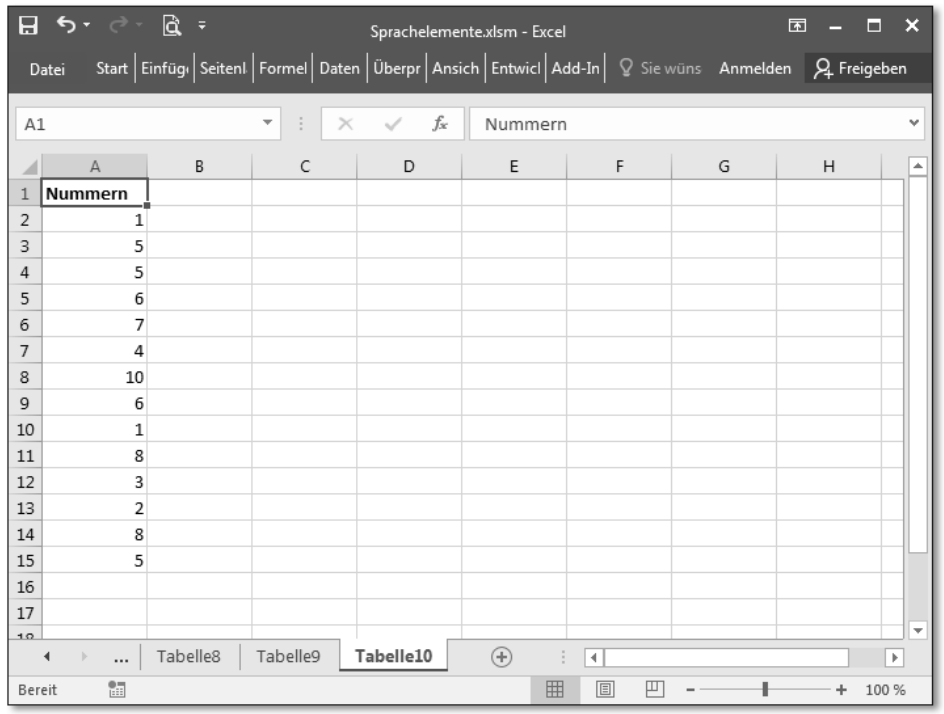
Auf den ersten Blick ist es gar nicht so einfach, die doppelten Werte zu erkennen. Für ein Makro ist das kein Problem. Sehen Sie sich dazu das Makro aus Listing 3.18 an.

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

  With Tabelle10
   lngZeileMax = .UsedRange.Rows.Count

   For lngZeile = 2 To lngZeileMax
```



```
    If Application.WorksheetFunction.CountIf(.Range("A:A"), _
      .Range("A" & lngZeile).Value) > 1 Then
      .Range("A" & lngZeile).Interior.ColorIndex = 4
    Else
      .Range("A" & lngZeile).Interior.ColorIndex = xlColorIndexNone
    End If

   Next lngZeile

  End With

End Sub
```

Listing 3.18 Doppelte Werte aufspüren und kenntlich machen



	A
1	Nummern
2	1
3	5
4	5
5	6
6	7
7	4
8	10
9	6
10	1
11	8
12	3
13	2
14	8
15	5
16	
17	

Abbildung 3.13 Hier kommen einige Nummern doppelt vor – welche?

Im Inneren der Schleife wird der Zugriff auf alle Tabellenfunktionen von Excel über die Eigenschaft `Worksheetfunction` angezapft. In dieser Auflistung finden Sie unter anderem die Tabellenfunktion ZÄHLENWENN (englisch `countIf`), die Sie einsetzen können, um die Duplikate zu ermitteln. Der Rest ist Formsache.

Denken Sie daran, im `Else`-Zweig der Abfrage die Farbe wieder zurückzusetzen. Es könnte ja sein, dass Sie einen ehemals doppelten in einen einmaligen Wert korrigieren. Dann muss aus der ehemals grünen Zelle schließlich wieder eine farblose Zelle werden. Die Zuweisung »keine Farbe« wird über die Konstante `xlColorIndexNone` erreicht, die der Eigenschaft `ColorIndex` zugewiesen wird.

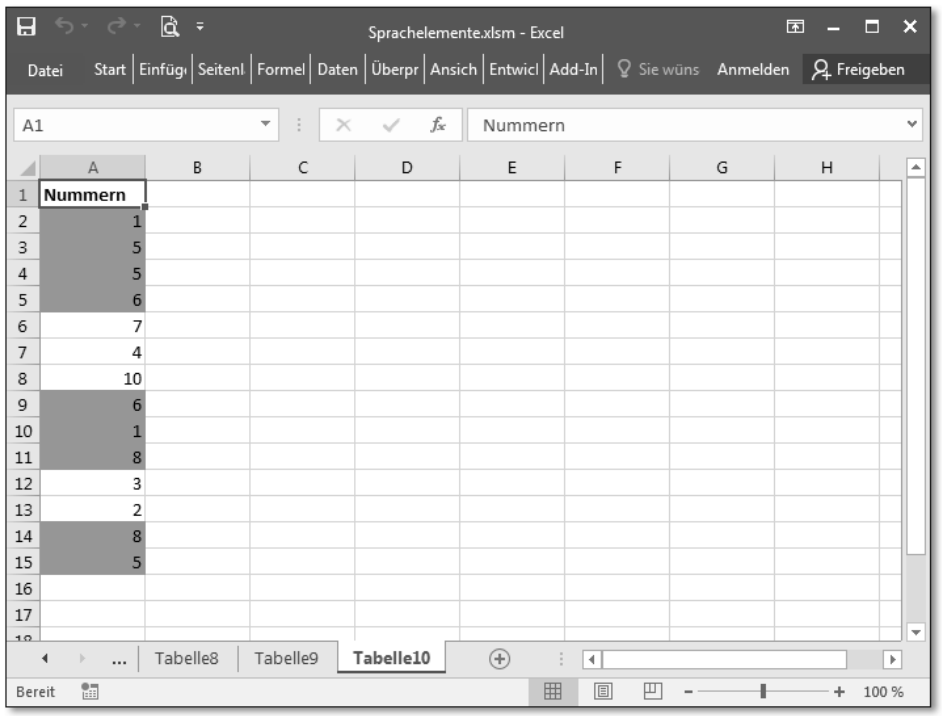


Abbildung 3.14 Alle doppelten Nummern wurden eingefärbt.

Leere Zeilen entfernen

Bei der folgenden Aufgabe liegen in TABELLE11 Daten vor. Einige Zeilen sind dabei leer, andere teilweise gefüllt.

Zur Lösung der Aufgabe können Sie die Schablone *Schablone_VonUntenNachOben* verwenden und etwas anpassen. So geschehen im Makro aus Listing 3.19.

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

  With Tabelle11
      lngZeileMax = .UsedRange.Rows.Count
      For lngZeile = lngZeileMax To 1 Step -1
```



```
          If Application.WorksheetFunction.CountA(.Rows(lngZeile)) = 0 Then
              .Rows(lngZeile).Delete
              lngZz = lngZz + 1
          End If
      Next lngZeile
  End With

  MsgBox lngZz & " Zeilen wurden gelöscht!", vbInformation
End Sub
```

Listing 3.19 Leere Zeilen entfernen

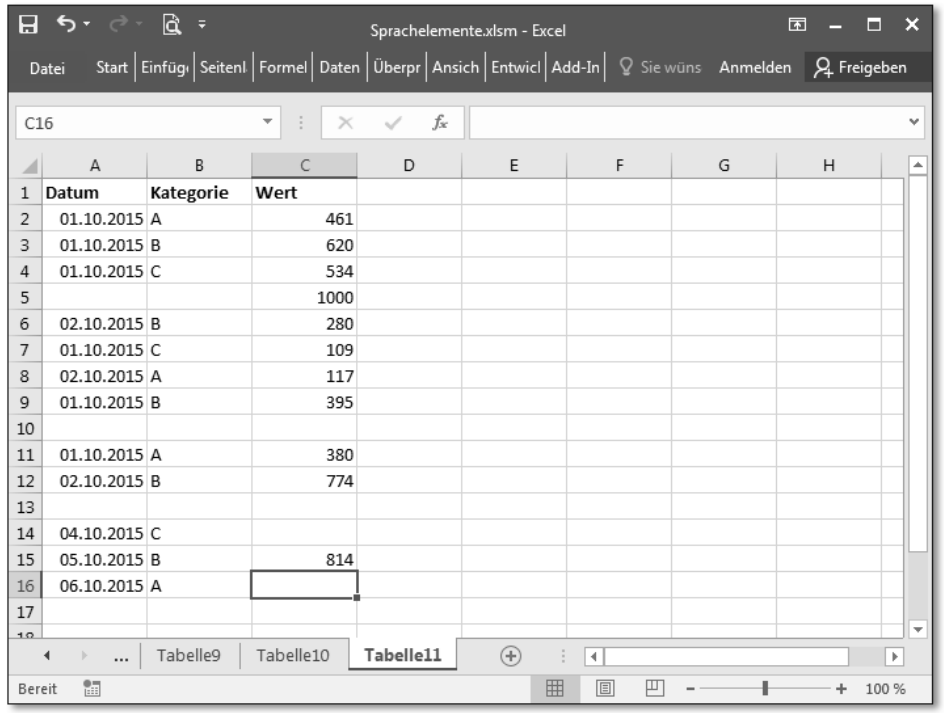
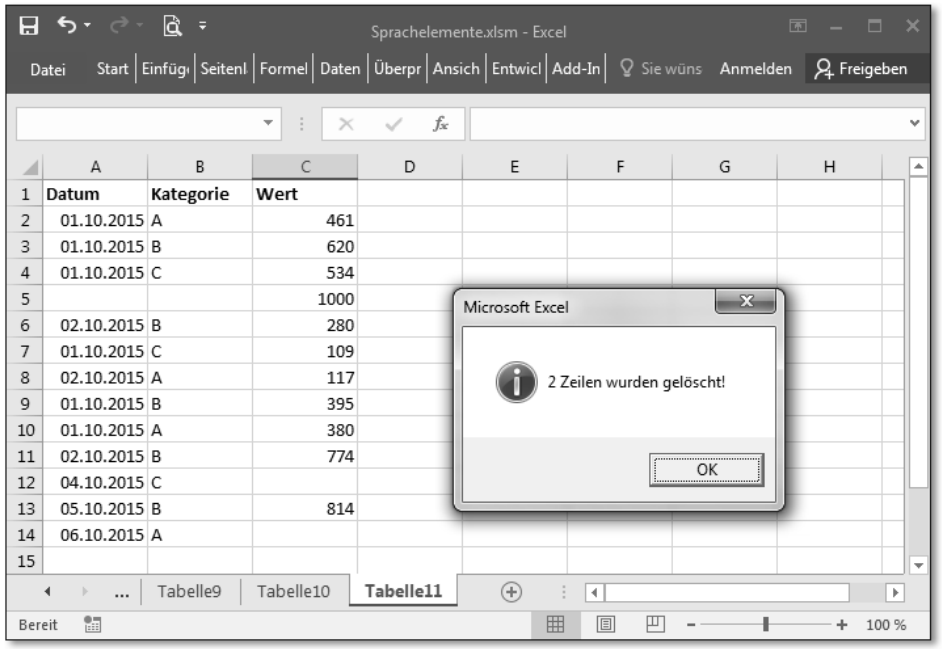


Abbildung 3.15 Nur die wirklich leeren Zeilen sollen entfernt werden.

Die Schleife muss beim Löschen von Zeilen in einer Tabelle von unten nach oben laufen. In der Schleife selbst wenden Sie die Tabellenfunktion `CountA` (deutsch ANZAHL2) an, um zu ermitteln, ob in der kompletten Zeile überhaupt irgendein Eintrag (Wert oder Text) steht. Wenn nicht, dann kann die komplette Zeile über die Methode `Delete` entfernt werden. Bei jeder Löschung erhöhen Sie die Zählvariable `lngZz` um den Wert 1.



	A	B	C
1	Datum	Kategorie	Wert
2	01.10.2015	A	461
3	01.10.2015	B	620
4	01.10.2015	C	534
5			1000
6	02.10.2015	B	280
7	01.10.2015	C	109
8	02.10.2015	A	117
9	01.10.2015	B	395
10	01.10.2015	A	380
11	02.10.2015	B	774
12	04.10.2015	C	
13	05.10.2015	B	814
14	06.10.2015	A	
15			

Abbildung 3.16 Die leeren Zeilen wurden entfernt und dabei nebenher noch gezählt.

Wochenenden und Werktage kennzeichnen

Bei der Aufgabe aus Listing 3.20 sollen in Tabelle12 die Wochenenden sowie die normalen Werktage gekennzeichnet werden. Dabei erhalten Wochenenden die Farbe Grün, die Werktage hingegen werden mit der Hintergrundfarbe Gelb formatiert. Für diese Aufgabe können Sie die *Schablone_VonLinksNachRechts* verwenden und etwas anpassen.

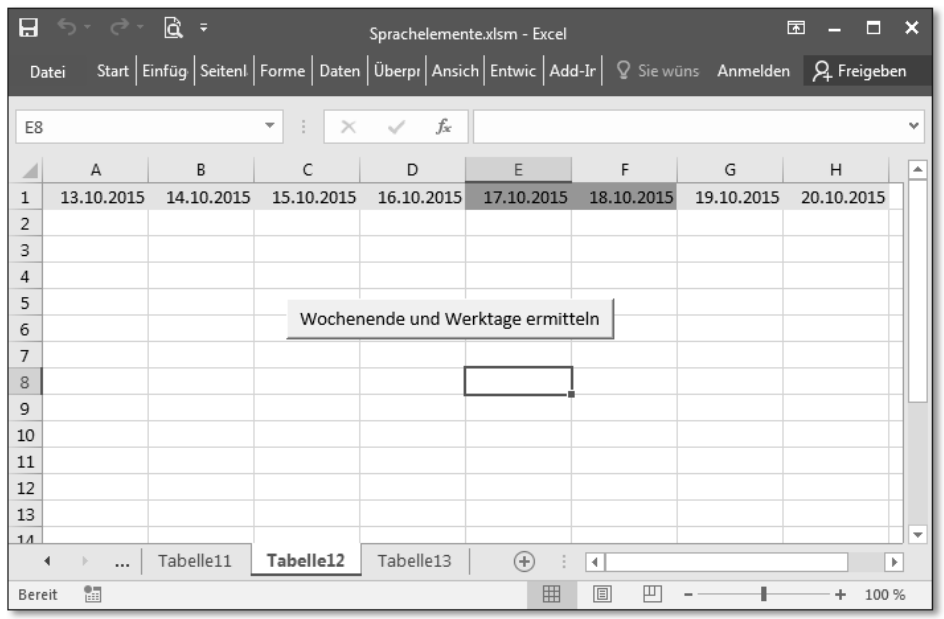
```
Sub WochenendenKennzeichnenSpalten()
  Dim lngSpalte As Long
  Dim lngSpalteMax As Long

  With Tabelle12
   lngSpalteMax = .UsedRange.Columns.Count

   For lngSpalte = 1 To lngSpalteMax

     If Weekday(.Cells(1, lngSpalte).Value, vbMonday) > 5 Then
       .Cells(1, lngSpalte).Interior.ColorIndex = 4
       Else
       .Cells(1, lngSpalte).Interior.ColorIndex = 6
     End If
```



```
   Next lngSpalte

  End With

End Sub
```

Listing 3.20 Wochenende und Werktage in Spalten kennzeichnen

In der Schleife selbst kommt die Funktion `Weekday` zum Einsatz, die ich in diesem Kapitel schon besprochen habe. Daher gehe ich an dieser Stelle nicht mehr darauf ein.



	A	B	C	D	E	F	G	H
1	13.10.2015	14.10.2015	15.10.2015	16.10.2015	17.10.2015	18.10.2015	19.10.2015	20.10.2015

Abbildung 3.17 Alle Tage in der Datumsleiste wurden gekennzeichnet.

Bestimmte Spalten aus einer Tabelle entfernen

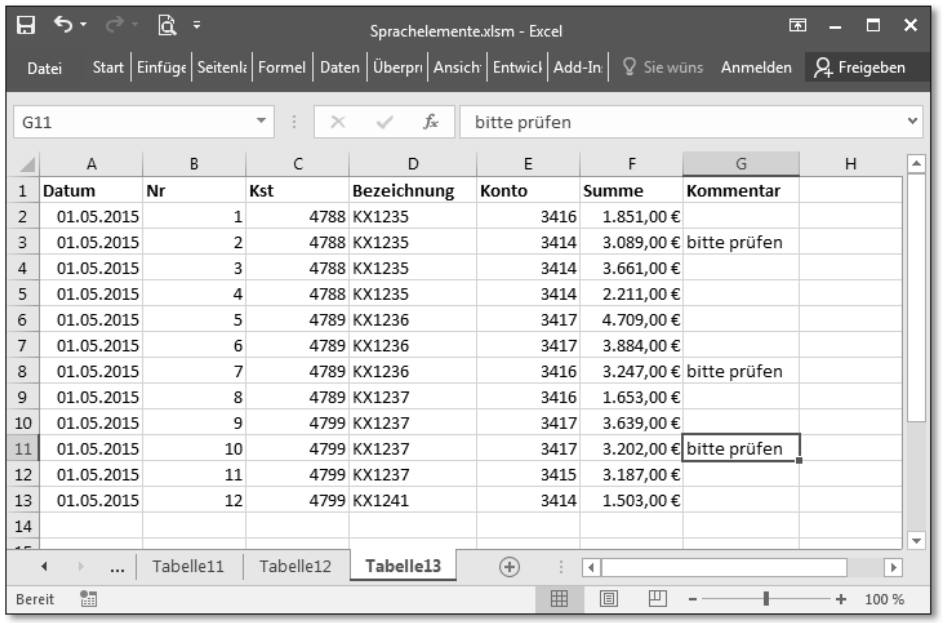
Bei der folgenden Aufgabenstellung liegen in Tabelle13 einige Daten vor. Einige Spalten in dieser Tabelle werden nicht mehr benötigt und können gelöscht werden.

Für diese Aufgabe können Sie die *Schablone_VonRechtsNachLinks* heranziehen und anpassen. Sehen Sie sich dazu die Umsetzung in Listing 3.21 an:

```
Sub BestimmteSpaltenLöschen()
  Dim lngSpalte As Long
  Dim lngSpalteMax As Long

  With Tabelle13
      lngSpalteMax = .UsedRange.Columns.Count
```

```
        For lngSpalte = lngSpalteMax To 1 Step -1

            Select Case UCase(.Cells(1, lngSpalte).Value)

              Case "SUMME", "Nr"
                  .Columns(lngSpalte).Delete
              Case Else
                  'keine Aktion
            End Select

        Next lngSpalte

    End With

End Sub
```

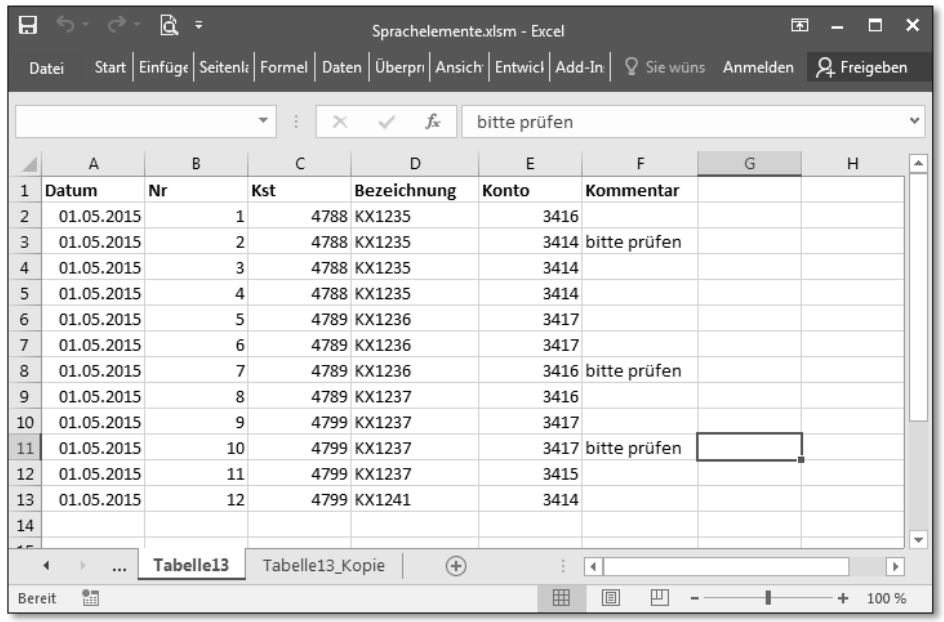
Listing 3.21 Bestimmte Spalten ersatzlos entfernen



	A	B	C	D	E	F	G	H
1	Datum	Nr	Kst	Bezeichnung	Konto	Summe	Kommentar	
2	01.05.2015	1	4788	KX1235	3416	1.851,00 €		
3	01.05.2015	2	4788	KX1235	3414	3.089,00 €	bitte prüfen	
4	01.05.2015	3	4788	KX1235	3414	3.661,00 €		
5	01.05.2015	4	4788	KX1235	3414	2.211,00 €		
6	01.05.2015	5	4789	KX1236	3417	4.709,00 €		
7	01.05.2015	6	4789	KX1236	3417	3.884,00 €		
8	01.05.2015	7	4789	KX1236	3416	3.247,00 €	bitte prüfen	
9	01.05.2015	8	4789	KX1237	3416	1.653,00 €		
10	01.05.2015	9	4799	KX1237	3417	3.639,00 €		
11	01.05.2015	10	4799	KX1237	3417	3.202,00 €	bitte prüfen	
12	01.05.2015	11	4799	KX1237	3415	3.187,00 €		
13	01.05.2015	12	4799	KX1241	3414	1.503,00 €		
14								

Abbildung 3.18 Die Spalten »Nr« und »Summe« sollen entfernt werden.

In der rückwärts laufenden Schleife wenden Sie die `Select Case`-Anweisung an, um gleich mehrere Spaltentitel abzuhandeln. Mit Hilfe der Funktion `UCase` können Sie Excel dazu bewegen, nicht zwischen Groß- und Kleinschreibung zu unterscheiden. Löschen Sie nicht benötigte Spalten, indem Sie die Methode `Delete` auf die entsprechende Spalte anwenden.



	A	B	C	D	E	F	G	H
1	Datum	Nr	Kst	Bezeichnung	Konto	Kommentar		
2	01.05.2015	1	4788	KX1235	3416			
3	01.05.2015	2	4788	KX1235	3414	bitte prüfen		
4	01.05.2015	3	4788	KX1235	3414			
5	01.05.2015	4	4788	KX1235	3414			
6	01.05.2015	5	4789	KX1236	3417			
7	01.05.2015	6	4789	KX1236	3417			
8	01.05.2015	7	4789	KX1236	3416	bitte prüfen		
9	01.05.2015	8	4789	KX1237	3416			
10	01.05.2015	9	4799	KX1237	3417			
11	01.05.2015	10	4799	KX1237	3417	bitte prüfen		
12	01.05.2015	11	4799	KX1237	3415			
13	01.05.2015	12	4799	KX1241	3414			
14								

Abbildung 3.19 Die nicht relevanten Spalten wurden entfernt.

3.6.2 Die »For Each ... Next«-Schleife

Bei der `For Each ... Next`-Schleife handelt es sich um die schnellste Schleife, die Excel zur Verfügung hat. Diese Schleife wird mit Objekten wie Zellen, Tabellen und Mappen eingesetzt. Alles, was Sie dafür brauchen, sind das Objekt an sich und die dazugehörige Objektvariable. Diese Schleife gehört zu den leicht verständlichen Schleifen, wenn Sie bei der Deklaration der Variablen sinnvolle Namen definieren. Die Schleife `For Each ... Next` wiederholt eine Gruppe von Anweisungen für jedes Element in einem Datenfeld oder einer Auflistung.

Die Syntax dieser Schleife lautet:

```
For Each Element In Gruppe
  Anweisungen
  Exit For
Next Element
```

Das Argument `Element` stellt die Variable zum Durchlauf durch die Elemente der Auflistung oder des Datenfeldes dar. Bei Auflistungen sind für `Element` nur eine Variable vom Typ `Variant`, eine allgemeine Objektvariable oder eine beliebige spezielle Objektvariable zulässig. Bei Datenfeldern ist für `Element` nur eine Variable vom Typ `Variant` zulässig. Das nächste Argument, `Gruppe`, steht für den Namen einer Objektauflistung oder eines Datenfeldes. Das letzte Argument, `Anweisungen`, ist optional und führt eine oder mehrere Anweisungen durch, die für jedes Element in der Gruppe ausgeführt werden sollen.

Alle leeren Zellen einer Markierung ansprechen

Beim folgenden Beispiel sollen alle Zellen, die vorher markiert wurden, durchsucht und leere Zellen mit der Hintergrundfarbe Gelb formatiert werden. Dazu können Sie das Makro aus Listing 3.22 einsetzen.

```
Sub AlleLeerenZellenInAuswahlIdentifizieren()
 Dim rngZelle As Range

  If TypeName(Selection) <> "Range" Then Exit Sub

  For Each rngZelle In Selection.SpecialCells(xlCellTypeBlanks)
          rngZelle.Interior.ColorIndex = 6
  Next rngZelle

End Sub
```

Listing 3.22 Leere Zellen in einer Auswahl von Zellen einfärben

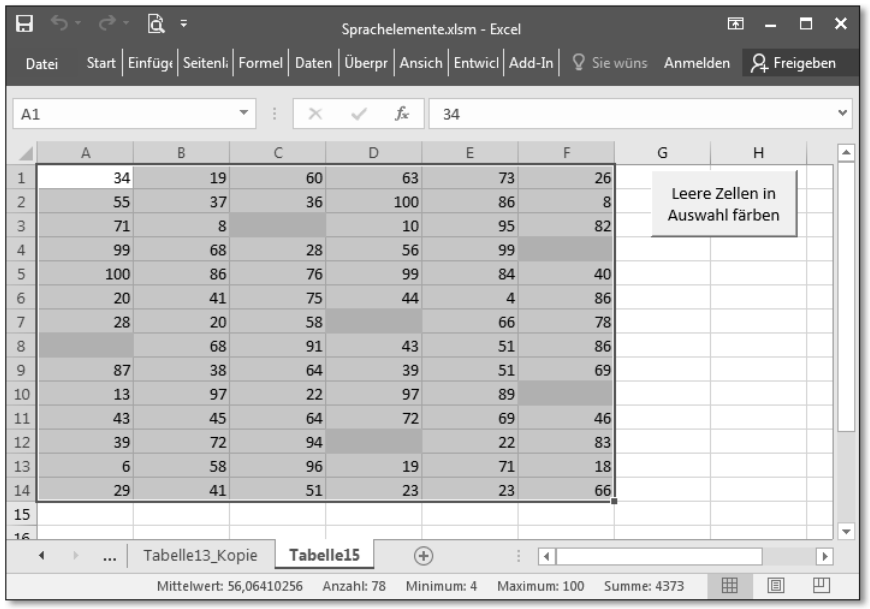


Abbildung 3.20 Alle leeren Zellen in der Auswahl wurden eingefärbt.

Mit Hilfe der Funktion `TypeName` stellen Sie sicher, dass überhaupt eine Zellenmarkierung vorliegt. In diesem Fall liefert die Funktion als Rückgabe den Text `Range`. Diese Sicherheitsmaßnahme ist dann wichtig, wenn Sie beispielsweise eingebettete Diagrammobjekte in der Tabelle haben und aus Versehen anstatt der Zellen eben ein Diagramm markieren. In einem solchen Fall würde das Makro abstürzen.

In einer anschließenden Schleife vom Typ `For Each ... Next` werden alle Zellen der Markierung, wenn Sie denn leer sind, verarbeitet. Um zu ermitteln, ob die Zellen leer sind, können Sie die Methode `SpecialCells` mit der Konstanten `xlCellTypeBlanks` verwenden. Innerhalb der Schleife färben Sie den Innenraum der jeweiligen leeren Zelle, indem Sie dem Objekt `Interior` über die Eigenschaft `ColorIndex` die Farbnummer `6` (= Gelb) zuweisen.

Alle Zellen mit dem gleichen Inhalt markieren

Ganz nützlich ist auch das Makro aus Listing 3.23. Dabei wird der Inhalt der aktiven Zelle genommen und kontrolliert, ob sich im benutzten Bereich der Tabelle weitere Zellen mit dem gleichen Inhalt befinden. Wenn ja, dann werden diese dynamisch markiert.

```
Sub AlleZellenMitWertMarkieren()
  Dim rngZelle As Range
  Dim rngBereich As Range
  Dim lngZ As Long

  For Each rngZelle In Tabelle15.UsedRange
   If rngZelle.Value = ActiveCell.Value Then
    If lngZ = 0 Then
     Set rngBereich = rngZelle
     lngZ = 1
    Else
     Set rngBereich = Union(rngBereich, rngZelle)
    End If
   End If
  Next rngZelle

  rngBereich.Select

End Sub
```

Listing 3.23 Ausgehend von der aktiven Zelle weitere Zellen mit gleichem Inhalt markieren

In einer Schleife der Form `For Each ... Next` werden alle Zellen im benutzten Bereich von TABELLE15 abgearbeitet. Innerhalb der Schleife findet zunächst ein Vergleich der aktiven Zelle mit der jeweils durch die Schleife angesprochenen Zelle statt. Tritt eine Übereinstimmung das erste Mal auf, dann wird der Objektvariablen `rngBereich` die gerade verarbeitete Zelle über die Anweisung `Set` zugewiesen. Danach wird der Zähler `lngZ` auf den Wert 1 gesetzt. Bei weiteren Übereinstimmungen wird jeweils die dazukommende Zelle dem Bereich `rngBereich` über die Methode `Union` hinzugefügt. Am Ende des Makros wird die Methode `Select` verwendet, um die vorher ermittelnden Zellen zu markieren.

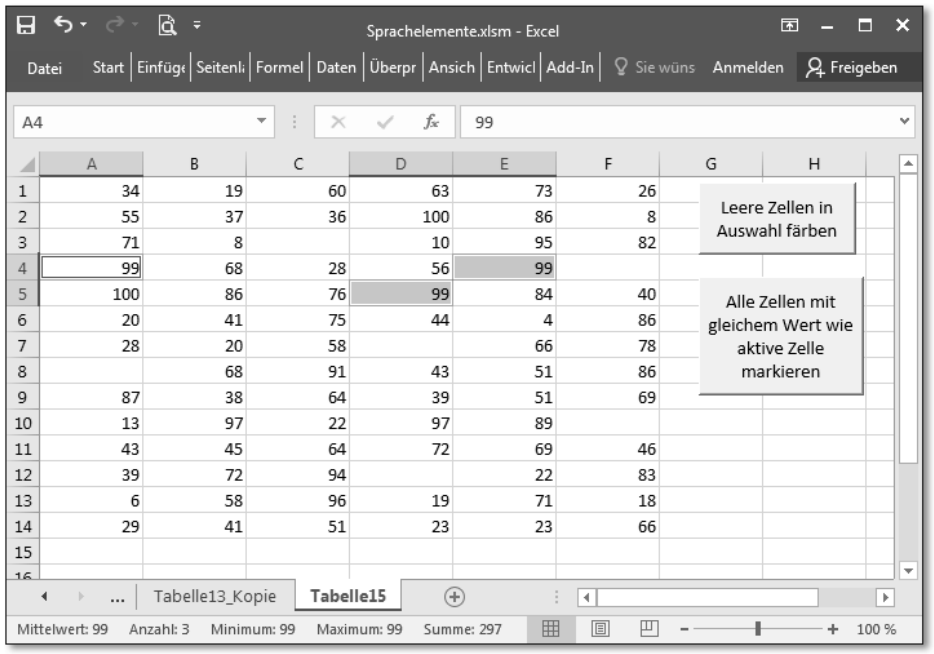


Abbildung 3.21 Alle Zellen mit dem Inhalt der aktiven Zelle werden markiert.

Die bedingte Formatierung per Makro einstellen

Bei der folgenden Aufgabenstellung sollen in TABELLE16 zwei Spalten miteinander verglichen werden. Dabei soll die bedingte Formatierung von Excel in Form von Symbolsätzen zum Einsatz kommen. Sehen Sie sich zunächst die Ausgangssituation aus Abbildung 3.22 an. Stellen Sie die bedingte Formatierung für Spalte B über das Makro aus Listing 3.24 ein:

```
Sub BedingteFormatierungPerMakroEinfügen()
  Dim rngZelle As Range
  Dim lngZeileMax As Long

  With Tabelle16
   lngZeileMax = .Cells(.Rows.Count, 1).End(xlUp).Row

      With .Range("B1:B" & lngZeileMax)

        .FormatConditions.Delete

        For Each rngZelle In .Cells

          rngZelle.FormatConditions.AddIconSetCondition
          With rngZelle.FormatConditions(1)
```



```
            .IconSet = ActiveWorkbook.IconSets(xl3Arrows)
            With .IconCriteria(2)
                .Type = xlConditionValueFormula
                .Value = "=" & rngZelle(1, 0).Address
                .Operator = xlGreaterEqual
            End With
            With .IconCriteria(3)
                .Type = xlConditionValueFormula
                .Value = "=" & rngZelle(1, 0).Address
                .Operator = xlGreater
            End With
          End With

        Next rngZelle

    End With

  End With

End Sub
```

Listing 3.24 Für den schnellen Überblick einen Symbolsatz per Makro einfügen

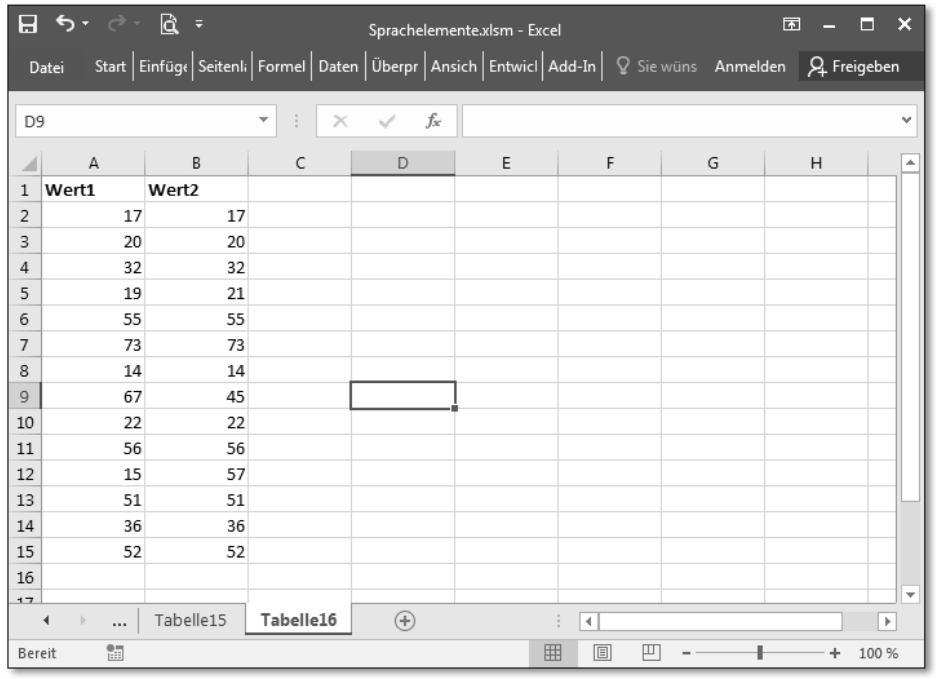


Abbildung 3.22 Wo sind die Unterschiede? Welcher Wert ist kleiner oder größer?

Deklarieren Sie zu Beginn des Makros aus Listing 3.24 die Objektvariable `rngZelle` als `Range` und die Variable `lngZeileMax` mit dem Datentyp `Long`. Danach ermitteln Sie, wie viele Zeilen in Spalte B gefüllt sind. Setzen Sie dazu die Eigenschaft `End` ein, über die Sie mittels der Konstanten `xlUp` die Richtung festlegen. Sie schauen dabei aus der letzten Zeile von Spalte A nach oben und fragen von der letzten gefüllten Zelle in Spalte A die Zeilennummer über die Eigenschaft `Row` ab.

Geben Sie jetzt über die Anweisung `With` den Bereich derjenigen Zellen in Spalte B an, die Sie mit einer bedingten Formatierung ausstatten möchten. Danach löschen Sie alle eventuell bereits eingestellten bedingten Formate, indem Sie das Auflistungsobjekt `FormatConditions` komplett über die Methode `Delete` entfernen. Über die Methode `AddIconSetCondition` legen Sie fest, dass Sie mit einem Symbolsatz arbeiten möchten.

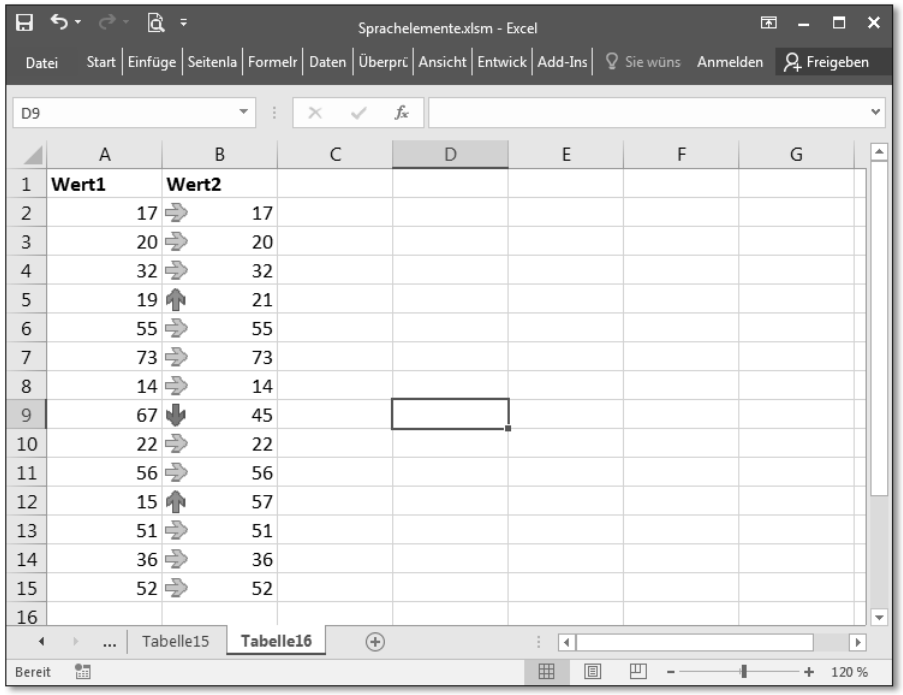


Abbildung 3.23 Veränderungen aller Art werden über Symbole veranschaulicht.

Mit einem Symbolsatz können Sie Daten kennzeichnen und in drei bis fünf Kategorien einteilen, die durch einen Schwellenwert getrennt werden. Jedes Symbol stellt einen Wertebereich dar. Im Symbolsatz »3 Pfeile« (`xl3arrows`) stellt der rote Pfeil nach unten beispielsweise niedrigere Werte dar, der gelbe Pfeil zur Seite gleiche Werte und der grüne Pfeil nach oben größere Werte. Die Einteilung findet über die Eigenschaft `IconCriteria` statt. In der Eigenschaft `Type` geben Sie bekannt, dass Sie zum Vergleichen der Spalte auf eine Formel zurückgreifen möchten. Diese Formel geben Sie danach über die Eigenschaft `Value` bekannt. Mit Hilfe der Eigenschaft `Operator` definieren Sie, wie Excel beim Vergleich vorgehen soll. Dabei werden in diesem Beispiel die Konstanten `xlGreaterEqaul` und `xlGreater` verwendet.

Einen Bereich in einer Tabelle rahmen

Bei der folgenden Aufgabenstellung soll in Tabelle17 ein bestimmter Bereich eingerahmt werden. Zusätzlich soll ein dickerer Rahmen um den kompletten Bereich gezogen werden. Sehen Sie sich die Umsetzung in Listing 3.25 an.

```
Sub BereichRahmen()
  Dim rngZelle As Range
  Dim rngBereich As Range

  Set rngBereich = Tabelle17.Range("B2:D10")

  For Each rngZelle In rngBereich

    With rngZelle.Borders(xlEdgeLeft)
          .LineStyle = xlContinuous
          .Weight = xlThin
          .ColorIndex = xlAutomatic
       End With
       With rngZelle.Borders(xlEdgeTop)
           .LineStyle = xlContinuous
           .Weight = xlThin
           .ColorIndex = xlAutomatic
       End With
       With rngZelle.Borders(xlEdgeBottom)
           .LineStyle = xlContinuous
           .Weight = xlThin
           .ColorIndex = xlAutomatic
       End With
       With rngZelle.Borders(xlEdgeRight)
           .LineStyle = xlContinuous
           .Weight = xlThin
           .ColorIndex = xlAutomatic
       End With

  Next rngZelle

  rngBereich.BorderAround Weight:=xlThick, ColorIndex:=xlAutomatic

End Sub
```

Listing 3.25 Einen bestimmten Bereich in einer Tabelle mit einem Rahmen versehen

Deklarieren Sie im ersten Schritt des Makros aus Listing 3.25 zwei Objektvariablen vom Typ `Range`. Über die Anweisung `Set` geben Sie an, in welcher Tabelle sich der zu verarbeitende Bereich befindet, und gleichzeitig legen Sie fest, wo er genau liegt.

In einer Schleife des Typs `For Each ... Next` wird Zelle für Zelle abgearbeitet. Innerhalb der Schleife wird auf das Objekt `Borders` zugegriffen. Dabei wird über die Konstanten `xlEdgeLeft`, `xlEdgeTop`, `xlEdgeBottom` und `xlEdgeRight` festgelegt, welcher Rahmen angesprochen wird.

Mit Hilfe der Eigenschaft `LineStyle` legen Sie die Art der Linie fest. Über die Konstante `xlContinuous` definieren Sie eine normale durchgezogene Linie.

Über die Eigenschaft `Weight` legen Sie die Dicke der Linie fest. Zur Verfügung stehen dabei die selbstsprechenden Konstanten `xlHairline`, `xlThick`, `xlThin` und `xlMedium` zur Verfügung.

Mit Hilfe der Eigenschaft `ColorIndex` bestimmen Sie die Farbe des Rahmens. Die Konstante `xlAutomatic` färbt dabei die Linie schwarz (= Standard).

Mit Hilfe der Methode `BorderAround` legen Sie einen Gesamtrahmen um den Bereich. Dabei können Sie im Parameter `Weight` die Dicke der Linie festlegen. Hier stehen dieselben Konstanten wie bei der gleichnamigen Eigenschaft zur Verfügung. Über den Parameter `ColorIndex` legen Sie die Farbe des Gesamtrahmens fest.

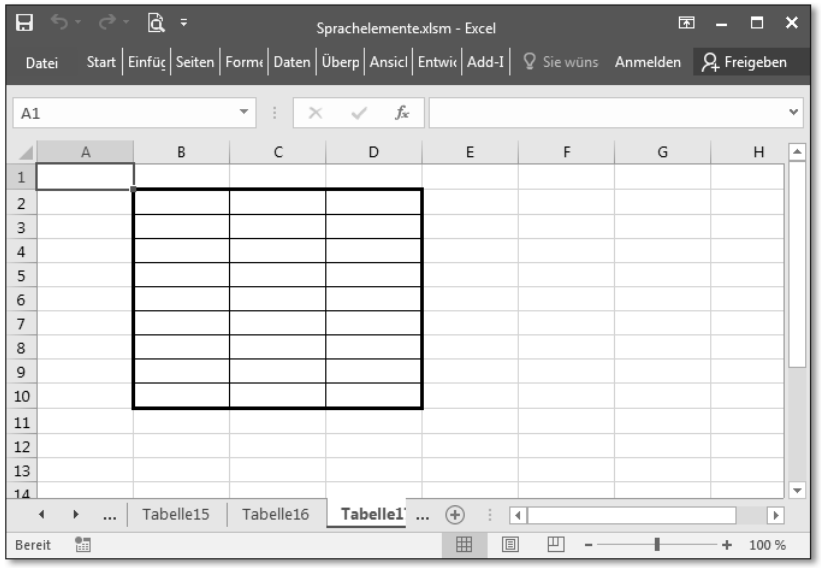


Abbildung 3.24 Der Bereich B2:D10 wurde eingerahmt.

Alle Tabellen einer Arbeitsmappe als separate Mappen abspeichern

Bei der folgenden Aufgabenstellung sollen alle Tabellen der Arbeitsmappe kopiert und als eigenständige Dateien gespeichert werden. Diese Arbeit möchten Sie sicher

nicht händisch machen, oder? Zur Umsetzung dieser Aufgabenstellung können Sie das Makro aus Listing 3.26 einsetzen.


```
Sub AlleTabellenExportieren()
 Dim wksBlatt As Worksheet

 Application.DisplayAlerts = False

 For Each wksBlatt In ThisWorkbook.Worksheets

   Application.StatusBar = "Tabelle " & wksBlatt.Name & " wird exportiert"
   wksBlatt.Copy
   ActiveWorkbook.SaveAs Filename:="C:\Windows\Temp\" & _
                                    wksBlatt.Name & ".xlsx"
   ActiveWorkbook.Close

 Next wksBlatt

 Application.DisplayAlerts = True
 Application.StatusBar = False

End Sub
```

Listing 3.26 Alle Tabellen als separate Mappen speichern

Deklarieren Sie im ersten Schritt des Makros aus Listing 3.26 eine Objektvariable vom Typ `Worksheet` mit dem Namen `wksBlatt`. Schalten Sie über die Eigenschaft `DisplayAlerts` Excel-Warnmeldungen ab, indem Sie dieser Eigenschaft den Wert `False` zuweisen.

In einer Schleife der Art `For Each ... Next` arbeiten Sie alle Tabellen der Arbeitsmappe nacheinander ab. Innerhalb der Schleife dokumentieren Sie den Exportvorgang, indem Sie die Statusleiste von Excel beschreiben. Dazu weisen Sie der Eigenschaft `StatusBar` einen begleitenden Text zu.

Mit Hilfe der Methode `Copy` kopieren Sie die einzelnen Tabellen aus der Arbeitsmappe. Dadurch werden diese Tabellen jeweils zur aktiven Arbeitsmappe, die Sie über die Methode `SaveAs` im Ordner *Temp* von *Windows* speichern können. Schließen Sie danach die aktive Arbeitsmappe über die Methode `Close`.

Vergessen Sie nicht, der Eigenschaft `DisplayAlerts` den Wert `True` zuzuweisen, um zukünftig Warnmeldungen von Excel wieder anzuzeigen. Geben Sie die Steuerung der Statusleiste wieder an Excel zurück, indem Sie die Eigenschaft `StatusBar` auf den Wert `False` setzen.

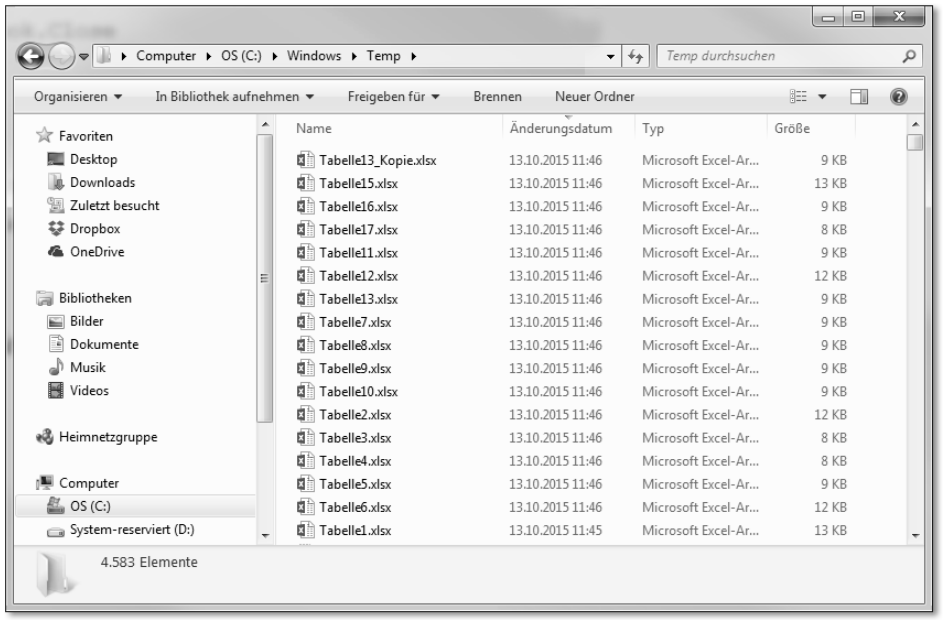


Abbildung 3.25 Die neuen Mappen liegen im Zielverzeichnis vor.

Alle Kommentare einer Tabelle nachformatieren

Bei der folgenden Aufgabenstellung liegen in TABELLE18 einige Zellenkommentare vor, die umformatiert werden sollen.

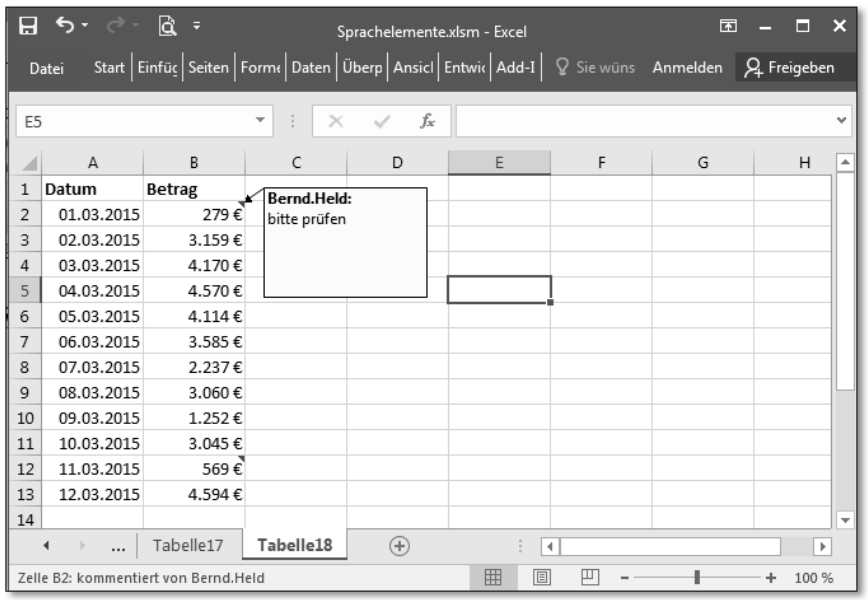


Abbildung 3.26 Die Kommentare der Tabelle sollen anders formatiert werden.

Starten Sie das Makro aus Listing 3.27, um alle Kommentare in TABELLE18 umzuformatieren.

```
Sub KommentareFormatieren()
 Dim cmtNotiz As Comment

  For Each cmtNotiz In Tabelle18.Comments
      With cmtNotiz.Shape.TextFrame.Characters.Font
          .Name = "Arial"
          .Size = 14
          .Italic = True
          .Underline = True
          .Bold = True
      End With
  Next cmtNotiz

End Sub
```

Listing 3.27 Alle Notizen in einer Tabelle umformatieren

Deklarieren Sie zu Beginn des Makros aus Listing 3.27 eine Objektvariable vom Typ `Comment` mit dem Namen `cmtNotiz`. In einer anschließenden Schleife des Typs `For Each ... Next` durchlaufen Sie alle Kommentare aus TABELLE18, die automatisch im Auflistungsobjekt `Comments` verzeichnet sind.

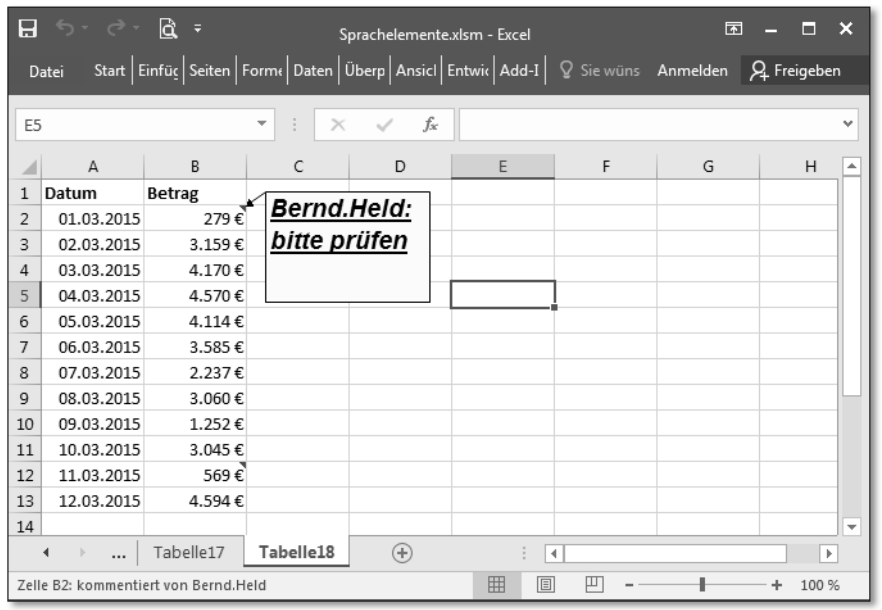


Abbildung 3.27 Die Kommentare sind nun besser lesbar.

Innerhalb der Schleife greifen Sie auf den Kommentar zu, indem Sie über das `Shape`-Objekt gehen, da der Kommentar ja quasi in einem Rechteck liegt. Innerhalb dieser Form gibt es ein Textfeld, das Sie über die Eigenschaft `TextFrame` ansteuern können. Innerhalb dieses Textfelds können Sie über die Auflistung `Characters` auf alle erfassten Zeichen zugreifen.

Über das Objekt `Font` können Sie danach ganz gezielt die Schriftformatierung des Kommentars beeinflussen. Dazu stehen Ihnen einige Eigenschaften zur Verfügung. Über die Eigenschaft `Name` stellen Sie die gewünschte Schriftart ein. Die Eigenschaft `Size` legt fest, wie groß die Schrift ist. Über die Eigenschaft `Italic` formatieren Sie den Kommentartext kursiv. Mit Hilfe der Eigenschaft `Underline` unterstreichen lassen Sie den Kommentartext automatisch unterstreichen. Über die Eigenschaft `Bold` definieren Sie den Fettdruck.

3.6.3 Die Schleife »Do Until ... Loop«

Die Schleife des Typs `Do Until ... Loop` wiederholt einen Block mit Anweisungen, solange eine Bedingung den Wert `True` erhält. Die Bedingung wird jeweils am Ende der Schleife geprüft. Als Abbruchbedingung lassen sich unterschiedliche Bedingungen abfragen.

So können Sie z. B. eine Abbruchbedingung festlegen, wenn ein bestimmter Wert erreicht ist oder eine Zelle einen bestimmten Text aufweist. Beispielsweise könnten Sie eine solche Schleife so oft wiederholen, wie sich die Zellenformatierung der Zellen nicht ändert.

Die Syntax dieser Schleife sieht wie folgt aus:

```
Do Until Bedingung
  Anweisungen
  Exit Do
Loop
```

Die `Bedingung` stellt einen numerischen Ausdruck oder Zeichenfolgenausdruck dar, der entweder erfüllt (`True`) oder nicht erfüllt (`False`) ist. Liefert die Bedingung den Wert `0`, so wird die Bedingung als `False` interpretiert. Hinter den `Anweisungen` verbergen sich eine oder mehrere Anweisungen, die wiederholt werden, solange oder bis `Bedingung` durch `True` erfüllt ist.

Eine CSV-Datei mit Umsätzen einlesen

Beim folgenden Beispiel liegt eine CSV-Datei wie in Abbildung 3.28 gezeigt vor. Diese Textdatei soll in Tabelle19 eingelesen werden.

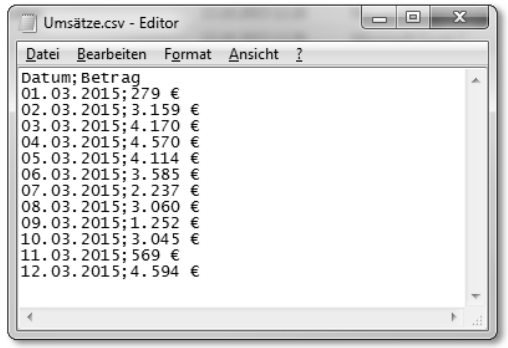


Abbildung 3.28 Die Daten aus dieser CSV-Datei sollen importiert werden.

Um die CSV-Datei in Tabelle19 zu übertragen, starten Sie das Makro aus Listing 3.28.

```
Sub TextdateiEinlesen()
  Dim objFSO As Object
  Dim ts As Object
  Dim strSatz As String
  Dim intZ As Integer

  With Tabelle19
    .Rows.Delete
    Set FSO = CreateObject("Scripting.FileSystemObject")

    'hier gegebenfalls den Pfad anpassen
    Set ts = FSO.OpenTextFile(ThisWorkbook.Path & "\Umsätze.csv")

    intZ = 1

      Do Until ts.AtEndOfStream
          strSatz = ts.ReadLine
          Cells(intZ, 1).Value = strSatz
          intZ = intZ + 1
      Loop

    ts.Close

    .Columns("A:A").TextToColumns Destination:=.Range("A1"), _
        DataType:=xlDelimited, semicolon:=True
  End With

End Sub
```

Listing 3.28 Eine CSV-Datei in einer Excel-Tabelle einlesen (Zeile für Zeile)

Für das Importieren von Textdaten können Sie auf eine Windows-eigene Bibliothek zurückgreifen, die alle Methoden und Eigenschaften anbietet, die Sie für die Bearbeitung von Textdateien benötigen. Diese Bibliothek mit dem Namen SCRIPTING RUNTIME binden Sie nach dem Prinzip des *Late Bindings* ein. Das bedeutet, dass Sie erst während des Makrolaufs einen Verweis auf diese Bibliothek mittels der Funktion `CreateObject` vornehmen. Danach haben Sie Zugriff auf alle Befehle, die in dieser Bibliothek enthalten sind. Unter anderem ist das die Methode `OpenTextFile`, über die Sie eine Textdatei öffnen.

Danach setzen Sie eine `Do Until`-Schleife auf, die Zeile für Zeile aus der Textdatei liest und in TABELLE19 schaufelt. Innerhalb der Schleife wenden Sie die Methode `ReadLine` an, um eben zeilenweise die Textdatei auszulesen. Dabei speichern Sie Satz für Satz temporär in der `String`-Variablen `strSatz`. Den Inhalt der Variablen übertragen Sie dann zunächst in die erste Spalte der Tabelle. Die Auftrennung des Strings nehmen Sie später vor.

Nach dem Schleifenaustritt wenden Sie die Methode `Close` an, um die Textdatei zu schließen. Danach kommt die Methode `TextToColumns` zum Einsatz, die auf Basis des Trennzeichens (in diesem Fall ist das ein Semikolon) die Daten trennt.

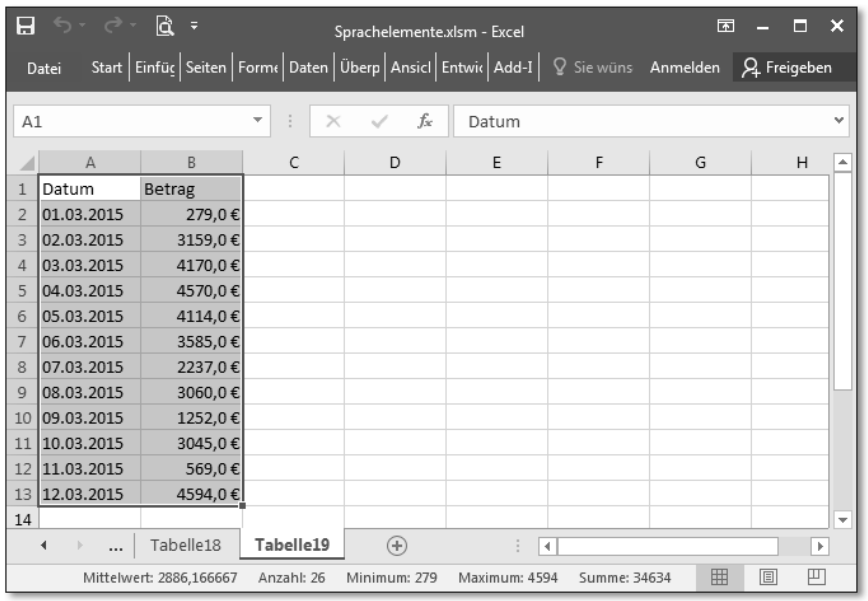


Abbildung 3.29 Alle Datensätze aus der CSV-Datei wurden importiert.

Arbeitsmappe nach Untätigkeit automatisch schließen

Für viele ein großes Ärgernis: Der Kollege »Dauerspaziergänger« arbeitet an einer Datei und geht zwischendurch einen Kaffee trinken. Er hat die Datei aber leider nicht geschlossen, und Sie kommen nicht an sie heran. Abhilfe schafft ein kleines Skript,

das Sie beispielsweise beim Öffnen der Arbeitsmappe über das Ereignis `Workbook_Open` starten.

Schauen Sie sich die Umsetzung des Vorhabens in Listing 3.29 an. Wir geben dem Kollegen mal 120 Sekunden, bis die Datei automatisch schließt und dabei gespeichert wird.

```
Sub Zeitschaltuhr()
  Dim dteBeginn As Date
  Dim dtePause As Date

  dteBeginn = Timer
  dtePause = 120

  Do Until Timer > dteBeginn + dtePause
      DoEvents
  Loop

  ThisWorkbook.Close savechanges:=True

End Sub
```

Listing 3.29 Eine Mappe nach zwei Minuten Untätigkeit einfach mal schließen

Deklarieren Sie zu Beginn des Makros aus Listing 3.29 zwei Datumsvariablen, um die Zeit messen zu können. Über die Funktion `Timer` holen Sie sich sekundenscharf die aktuelle Uhrzeit und speichern diese in der Variablen `dteBeginn`. Legen Sie in der Variablen `dtePause` die Anzahl der Sekunden fest, die Sie dem Kollegen auf Wanderschaft gewähren möchten.

In einer `Do Until`-Schleife wird jetzt bereits im Kopf der Schleife geprüft, ob die Zeit abgelaufen ist. In der Schleife selbst wird die Funktion `DoEvents` eingesetzt. Diese Funktion sorgt dafür, dass die Kontrolle wieder zurück an das Betriebssystem gegeben wird.

Bei Schleifenaustritt, also wenn die Zeit abgelaufen ist, wenden Sie die Methode `Close` an, um die Arbeitsmappe zu schließen. Über den Parameter `SaveChanges` können Sie dabei selbst entscheiden, ob Sie die Mappe noch einmal speichern möchten oder nicht.

3.6.4 Die Schleife »Do While ... Loop«

Die Schleife des Typs `Do While ... Loop` wiederholt einen Block mit Anweisungen, solange eine Bedingung den Wert `True` enthält. Die Prüfung der angegebenen Bedingung erfolgt immer zu Beginn der Schleife. Als Abbruchbedingung können Sie z. B.

eine Abbruchbedingung festlegen, wenn ein bestimmter Wert erreicht ist oder eine Zelle einen bestimmten Text aufweist.

Die Syntax dieser Schleife sieht wie folgt aus:

```
Do While Bedingung
  Anweisungen
  Exit Do
Loop
```

Die `Bedingung` stellt einen numerischen Ausdruck oder Zeichenfolgenausdruck dar, der entweder erfüllt (`True`) oder nicht erfüllt (`False`) wird. Liefert die Bedingung den Wert `0`, so wird die Bedingung als `False` interpretiert. Hinter den `Anweisungen` verbergen sich eine oder mehrere Anweisungen, die wiederholt werden, solange oder bis die Bedingung `True` erfüllt ist. Innerhalb einer Schleife der Art `Do While ... Loop` können Sie eine beliebige Anzahl von `Exit Do`-Anweisungen an beliebiger Stelle als Alternative zum Verlassen einer `Do ... Loop`-Anweisung verwenden.

Anzahl von Dateien eines Verzeichnisses ermitteln

Beim Beispiel aus Listing 3.30 sollen alle Dateien aus einem bestimmten Verzeichnis gezählt werden. Dazu setzen wir die Funktion `Dir` ein, die DOS-Anwender aus »alten Tagen« sicherlich noch kennen werden.

```
Sub DateienZaehlen()
  Dim strOrdnerName As String
  Dim strName As String
  Dim intz As Integer

  strOrdnerName = "C:\Windows\Temp\"
  strName = Dir(strOrdnerName & "*.*")

    Do While strName <> ""
        strName = Dir
        intz = intz + 1
    Loop

  MsgBox "Anzahl der Dateien: " & intz

End Sub
```

Listing 3.30 Alle Dateien in einem Verzeichnis zählen

Geben Sie zu Beginn des Makros den Namen des Ordners an, in dem die darin enthaltenen Dateien gezählt werden sollen. Übergeben Sie diesen Ordner sowie den »Dateifilter« an die Funktion `Dir`.

Nun kommt die `Do While`-Schleife zum Einsatz. Innerhalb der Schleife wird wiederum die Funktion `Dir` angewendet, um die nächste Datei zu ermitteln. Erhöhen Sie dann jeweils den Zähler `intz` um den Wert 1. Die Ende-Bedingung für die Schleife wird dann erreicht, wenn keine weitere Datei mehr gefunden wird. In diesem Fall wird eine leere Zeichenfolge zurückgegeben.

Alle Dateien aus einem Verzeichnis auslesen

Im Beispiel aus Listing 3.31 werden die Namen aller Dateien aus einem bestimmten Verzeichnis in Tabelle20 übertragen.

```
Sub DateienAusOrdnerInTabelleAusgeben()
  Dim strPfad As String
  Dim strDatei As String
  Dim lngZeile As Long

  With Tabelle20
  .Rows.Delete
  lngZeile = 1
  strPfad = ThisWorkbook.Path

  strDatei = Dir(strPfad & "\*.xls*")

   Do While strDatei <> ""
     .Cells(lngZeile, 1).Value = strPfad & "\" & strDatei
     lngZeile = lngZeile + 1
     strDatei = Dir
   Loop

   .Columns(1).AutoFit

  End With

End Sub
```

Listing 3.31 Alle Dateien aus einem Verzeichnis auslesen

Deklarieren Sie zu Beginn des Makros aus Listing 3.31 zwei `String`-Variablen. In der Variablen `strPfad` wird der Ordner angegeben, der durchsucht werden soll. Über die Variable `strDatei` wird später der Dateifilter gebildet.

Ermitteln Sie den Pfad des auszulesenden Ordners, indem Sie über die Eigenschaft `Path` den Pfad der Mappe ableiten, in der die Makros gespeichert sind. Das bedeutet, dass Excel genau in diesem Pfad nachsehen soll, in dem auch die Datei *Sprachelemente.xlsm* steht.

Über die Funktion `Dir` lesen Sie zunächst einmal die erste Datei im Verzeichnis aus. Anschließend setzen Sie die `Do While`-Schleife auf, die so lange über die Funktion `Dir` nach weiteren Dateien sucht, bis diese Funktion eine leere Zeichenfolge zurückliefert. Das ist in etwa so, als wenn Sie im Gebirge jodeln, und es kommt kein Echo zurück. Wenn also die Funktion `Dir` eine leere Zeichenfolge zurückliefert, dann werden keine weiteren Dateien mehr gefunden. Demnach erfolgt ein Schleifenaustritt. Passen Sie am Ende die Breite von Spalte A automatisch über die Methode `AutoFit` an.

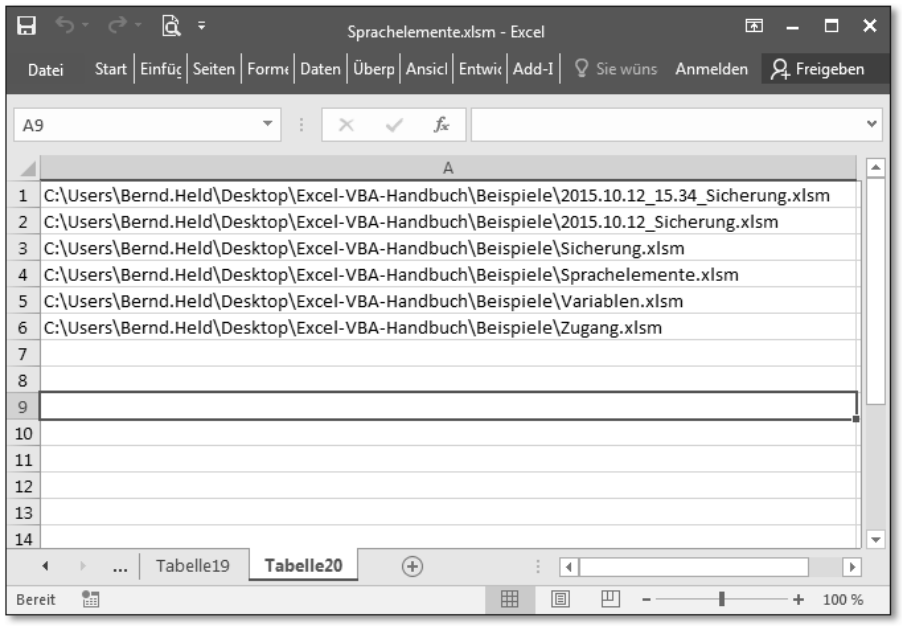


Abbildung 3.30 Eine Auflistung aller Dateien aus einem bestimmten Verzeichnis

3.7 Sonstige Sprachelemente

Neben Bedingungen und Schleifen gibt es noch ein anderes gängiges Sprachelement, das wir bereits hin und wieder in vorangegangenen Beispielen eingesetzt haben.

3.7.1 Die Anweisung »With«

Die Anweisung `With` wird eingesetzt, um sich viel Schreibarbeit zu sparen und mehr Übersichtlichkeit in den Quellcode zu bringen. Dabei wird das Objekt, auf das mehrere Befehle angewendet werden sollen, einmalig definiert, ohne dieses Objekt für jeden Befehl zu wiederholen.

Die Syntax dieser Anweisung lautet:

```
With Objekt

    Anweisungen

End With
```

Im Argument `Objekt` geben Sie den Namen des Objekts an, das Sie ansprechen möchten. Im Argument `Anweisungen` erfassen Sie ein oder mehrere Anweisungen, die für das Objekt ausgeführt werden sollen.

Alle verwendeten Zellen einer Spalte formatieren

Beim der folgenden Aufgabenstellung sollen alle verwendeten Zellen von Spalte A in Tabelle21 formatiert werden, und zwar neben dem Zahlenformat auch die Schriftfarbe, den Schriftschnitt sowie die Hintergrundfarbe. Damit Sie so wenig Schreibarbeit wie möglich haben, setzen Sie die `With`-Anweisung wie in Listing 3.32 gezeigt ein:

```
Sub BestimmteZelleAnprechenUndFormatieren()
  Dim lngZeileMax As Long

 lngZeileMax = Tabelle21.UsedRange.Rows.Count

 With Tabelle21.Range("A2:A" & lngZeileMax)

   'Datumsformat
   .NumberFormat = "DD.MM.YYYY"

   'Hintergrundfarbe festlegen (Grün)
   .Interior.ColorIndex = 4

   'Schriftschnitt Fett
   .Font.Bold = True

   'Schriftfarbe bestimmen (Weiss)
   .Font.ColorIndex = 2

 End With

End Sub
```

Listing 3.32 Alle verwendeten Zellen von Spalte A formatieren

Ermitteln Sie zu Beginn des Makros aus Listing 3.32, wie viele Zeilen in Spalte A belegt sind. Speichern Sie diese Information in der Variablen `lngZeileMax`. Danach geben Sie an, mit welcher Tabelle und mit welchem Bereich Sie arbeiten möchten. Setzen Sie dazu die Anweisung `With` ein. Immer wenn Sie danach auf diese »Zusammenfassung« zugreifen möchten, genügt es, wenn Sie als erstes Zeichen der Zeile einen Punkt erfassen. Excel weiß dann, welches Objekt Sie meinen.

Über die Eigenschaft `NumberFormat` können Sie die Formatierung des Datums festlegen. Die Formatierung erfolgt über Buchstabenkürzel (`DD` = zweistellige Tagesangabe, `MM` = zweistellige Monatsangabe, `YYYY` = vierstellige Jahresangabe).

Über die Eigenschaft `ColorIndex`, die Sie auf das Objekt `Interior` und das Objekt `Font` anwenden, färben Sie die verwendeten Zellen ein.

Über das Objekt `Font` haben Sie Zugriff auf alle Eigenschaften, die für die Formatierung der Schrift verantwortlich sind. Über die Eigenschaft `Bold` formatieren Sie einen Fettdruck.

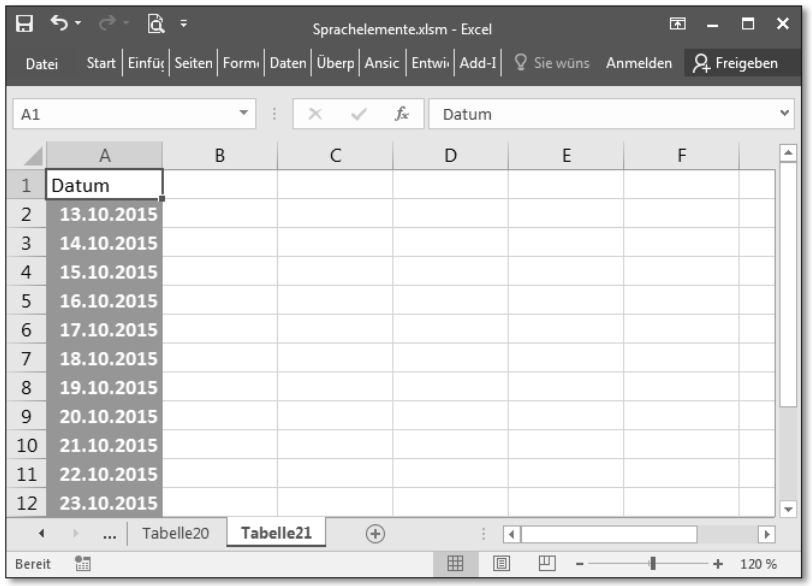


Abbildung 3.31 Das Datumsformat sowie die restlichen Formatierungen wurden eingestellt.

Eine eigene Gültigkeitsliste erstellen

Bei der folgenden Aufgabenstellung soll in Tabelle22 im Bereich A1:A10 eine Gültigkeitsliste in Form eines Zellen-Dropdowns automatisch angelegt werden. Dazu können Sie das Makro aus Listing 3.33 einsetzen:

```
Sub GültigkeitslisteEinfügen()
    Dim strAuswahl As String
```



```
    strAuswahl = "Deutschland, Spanien, Frankreich, Italien"

    With Tabelle22.Range("A1:A10").Validation
         .Delete
         .Add Type:=xlValidateList, _
         AlertStyle:=xlValidAlertStop, Formula1:=strAuswahl
         .InCellDropdown = True
         .InputTitle = "Land auswählen"
         .InputMessage = "Bitte das gewünschte Land auswählen"
         .ErrorTitle = "Fehler"
         .ErrorMessage = "Dieses Land ist nicht vorgesehen"
         .ShowInput = True
         .ShowError = True
    End With

End Sub
```

Listing 3.33 Eine Gültigkeitsliste als Zellen-Dropdown automatisch anlegen

Deklarieren Sie zunächst zu Beginn des Makros aus Listing 3.33 eine `String`-Variable mit dem Namen `strAuswahl`. Danach füllen Sie sie, indem Sie die gewünschten Länder mit Komma getrennt in diese Variable packen.

Danach sparen Sie sich eine Menge Schreibarbeit, indem Sie das Objekt `Validation` für den Zellenbereich A1:A10 für Tabelle22 über die Anweisung `With` angeben.

Wenden Sie zunächst die Methode `Delete` an, um gegebenenfalls eine bereits existierende Gültigkeitsregel in diesem Bereich zu entfernen.

Über die Methode `Add` fügen Sie eine neue Gültigkeitsregel ein. Dabei geben Sie über die Konstante `xlValidateList` an, dass Sie gerne eine Liste anlegen möchten. Im Parameter `AlertStyle` geben Sie über die Konstante `xlValidAlertStop` bekannt, dass Sie keine anderen Länder als die in der Liste befindlichen akzeptieren. Im Parameter `Formula1` verweisen Sie auf die vorher gefüllte `String`-Variable `strAuswahl`.

Um das Dropdown-Symbol in den Zellen anzuzeigen, weisen Sie der Eigenschaft `InCellDropdown` den Wert `True` zu.

Den Titel, der automatisch in einer QuickInfo angezeigt wird, wenn Sie den Cursor auf eine Zelle setzen, legen Sie über die Eigenschaft `InputTitle` fest. Die dazugehörige Meldung erfassen Sie mit Hilfe der Eigenschaft `InputMessage`.

Analog dazu können Sie den Fehlertitel und die Fehlermeldung über die Eigenschaften `ErrorTitle` und `ErrorMessage` festlegen.

Damit beide Meldungen überhaupt verfügbar sind, setzen Sie den Wert `True` für die Eigenschaften `ShowInput` und `ShowError`.

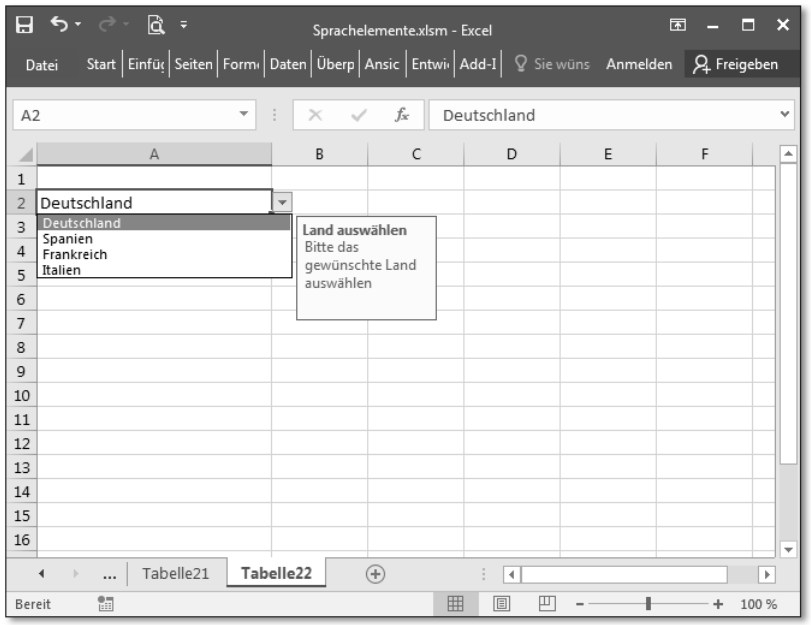


Abbildung 3.32 Die Gültigkeitsliste wurde erfolgreich eingefügt.

Eine Ampelfunktion für Monatsumsätze erstellen

Im letzten Beispiel in diesem Kapitel integrieren wir eine Umsatz-Monatstabelle mit einer Ampelfunktion mit Hilfe der bedingten Formatierung von Excel. Sehen Sie sich dazu einmal Abbildung 3.33 an.

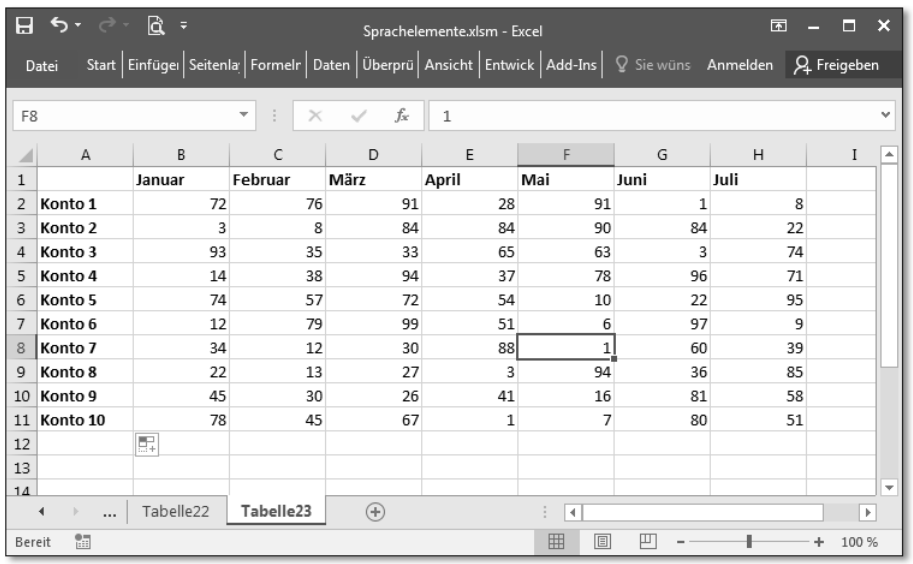


Abbildung 3.33 Die Ausgangssituation – eine Umsatzliste nach Monaten und Konten

Für die Ampel sollen folgende Regeln gelten:

- Werte > 67 % → grüne Ampel
- Werte 33–67 % → gelbe Ampel
- Werte < 33 % → rote Ampel

Setzen Sie diese Regeln im Makro aus Listing 3.34 um:

```
Sub AmpelfunktionBedingtesFormat()
  Dim rngBereich As Range

   Set rngBereich = Tabelle23.Range("B2:H11")
   rngBereich.FormatConditions.Delete
   rngBereich.FormatConditions.AddIconSetCondition
   rngBereich.FormatConditions(1).IconSet = _
    ActiveWorkbook.IconSets(xl3TrafficLights2)
    With rngBereich.FormatConditions(1).IconCriteria(2)
        .Type = xlConditionValuePercent
        .Value = 33
        .Operator = xlGreater
    End With
    With rngBereich.FormatConditions(1).IconCriteria(3)
        .Type = xlConditionValuePercent
        .Value = 67
        .Operator = xlGreater
    End With

End Sub
```

Listing 3.34 Eine Ampelfunktion für einen Bereich einfügen

Deklarieren Sie im ersten Schritt eine Objektvariable mit dem Namen `rngBereich` vom Typ `Range`. Geben Sie im nächsten Schritt mit Hilfe der Anweisung `Set` bekannt, welche Tabelle und welchen Bereich darin Sie verarbeiten möchten.

Wenden Sie danach die Methode `Delete` an, um eventuell bereits existierende bedingte Formate in diesem Bereich zu entfernen. Nutzen Sie danach die Methode `AddIconSetCondition`, um mit einem Symbolsatz arbeiten zu können. Den Symbolsatz legen Sie über das Objekt `IconSet` fest. Über die Konstante `xl3TrafficLights2` geben Sie vor, dass Sie mit dreidimensionalen Ampelsymbolen arbeiten möchten. Danach stellen Sie die Formatierungskriterien über das Objekt `IconCriteria` fest. In der Eigenschaft `Type` legen Sie fest, dass Excel die prozentuale Berechnung anwenden soll. Über die Eigenschaft `Value` legen Sie diesen Prozentwert selber fest. In der Eigenschaft `Operator` geben Sie über die Konstante `xlGreater` an, dass die Formatierung dann gelten soll, wenn der in `Value` eingestellte Prozentsatz überschritten wird.

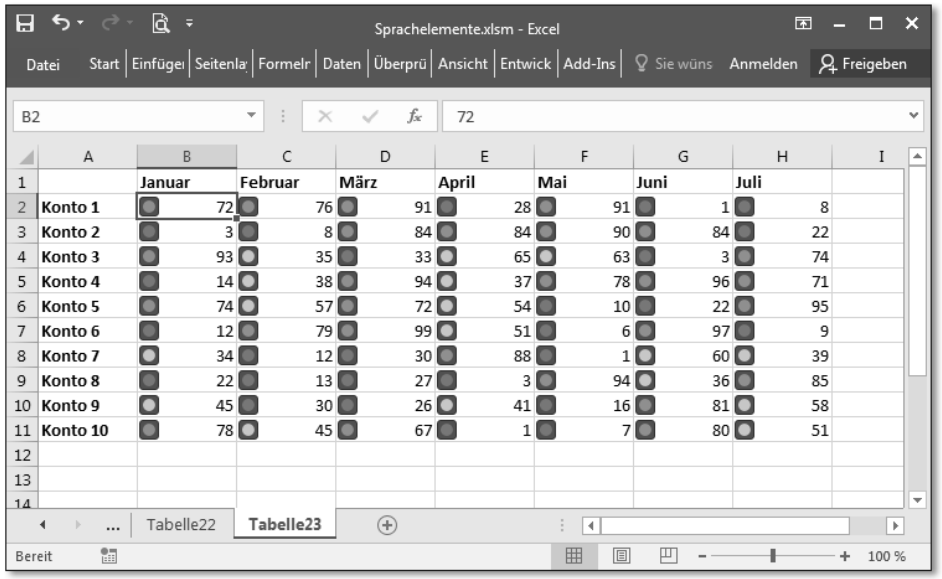


Abbildung 3.34 Die Ampeln wurden erfolgreich integriert.

Kapitel 6
Tabellen und Diagramme programmieren

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

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

Kapitelbegleitende Beispiele zum Download

Sie finden alle Beispiele in der Datei *Tabellen.xlsm* aus dem Downloadpaket, das Sie von *http://www.rheinwerk-verlag.de/3891* 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, also ganz rechts, einfügen, 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 zu ermitteln, welche die gewünschte Einfügeposition des neuen Tabellenblattes ist, 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 selbständig 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 mit 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 nach Monatsnamen benennen

Ermitteln Sie zunächst einmal 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.

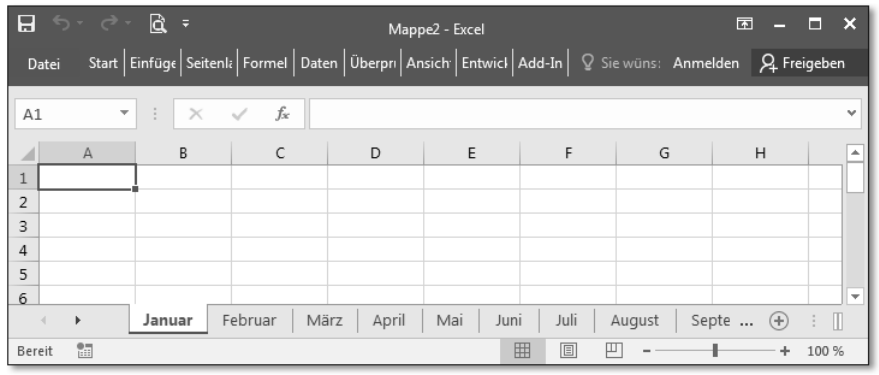


Abbildung 6.1 Die Monatstabellen wurden automatisch angelegt.

Setzen Sie danach eine Schleife des Typs `For Each ... Next` ein, in der Sie alle 12 Tabellen nacheinander verarbeiten. In der Schleife führen Sie die Benennung der Tabellen durch. Dazu setzen Sie die Funktion `MonthName` ein. Diese Funktion benötigt einen Wert zwischen 1 und 12, um den gewünschten Monat direkt aus der Windows-Sys-

temsteuerung in der dort eingestellten Landessprache zu holen. Diesen Wert leiten Sie über die Eigenschaft `Index` der jeweiligen Tabelle ab.

6.2.2 Eine neue Mappe mit den nächste 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.

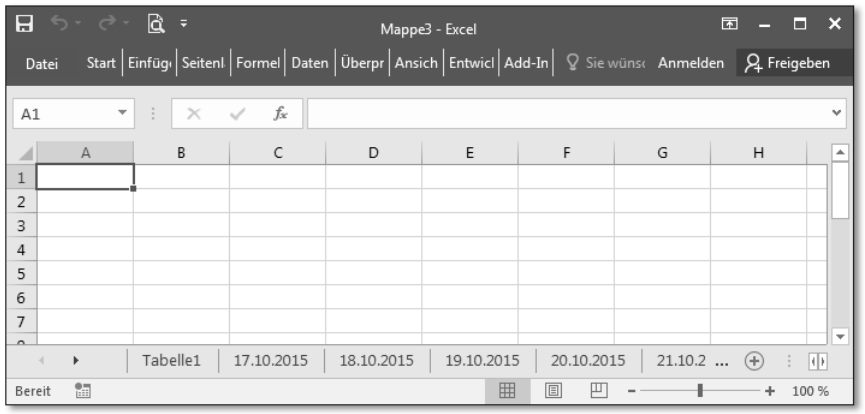


Abbildung 6.2 Eine neue Mappe für die nächsten 14 Tage steht bereit.

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 führen Sie die Benennung beginnend vom aktuellen Tagesdatum durch. Mit jedem Schleifendurchlauf wird der Schleifenzähler, der gleichzeitig auch der Tageszähler ist, um den Wert 1 inkrementiert.

6.2.3 Tabelle einfügen und benennen kombinieren

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

Mit Hilfe 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 einmal 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 können Sie selbstverständlich auch mit VBA erreichen. Im Folgenden erfahren Sie, wie Sie das machen.

6.4.1 Mehrere Tabellen gruppieren

```
Sub MehrereTabellenblätterMarkieren()

  On Error Resume Next
```



```
  Sheets(Array("Tabelle2", "Tabelle3", "Tabelle5")).Select

End Sub
```

Listing 6.9 Mehrere Tabellenblätter 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.

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 mit Hilfe 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 mit Hilfe 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

Mit Hilfe 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 Tabellenblattes 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 es, eine Funktion zu haben, die immer vorher prüft, ob eine Tabelle sich in der Arbeitsmappe befindet. In diesem Fall bräuchten Si 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.

Mit Hilfe 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 Tabellenamen 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 diese 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. Mit Hilfe 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

    For Each Blatt In ActiveWorkbook.Worksheets
      If Application.WorksheetFunction.CountA(Blatt.Cells) = 0 Then
        Application.DisplayAlerts = False
        Blatt.Delete
        Application.DisplayAlerts = True
      End If
    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 mit Hilfe 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.

Anwender der Versionen Excel 2007 bis 2016 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 der Anwender 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 mit Hilfe 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, der 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 Versionen von Excel 97 bis Excel 2016 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, `UserInterFaceOnly`, nimmt den Wert `True` an. Damit schützen Sie die Benutzeroberfläche, jedoch keine Makros. Ohne Angabe dieses Arguments gilt der Schutz für Makros und 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 durchzuführen. 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). Anwender der Versionen Excel 2007 bis 2016 klicken im Ribbon Überprüfen auf die Schaltfläche Blatt schützen.

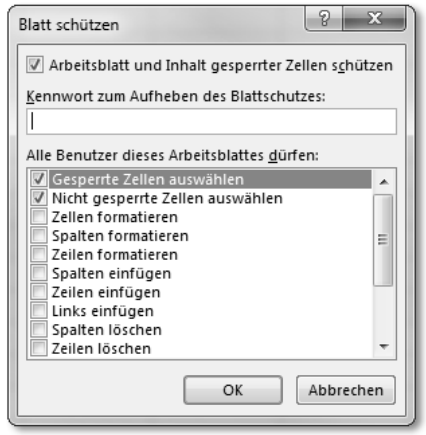


Abbildung 6.3 Erweiterte Schutzmöglichkeiten ab Excel 2002

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

```
Sub TabelleSchützen()

  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

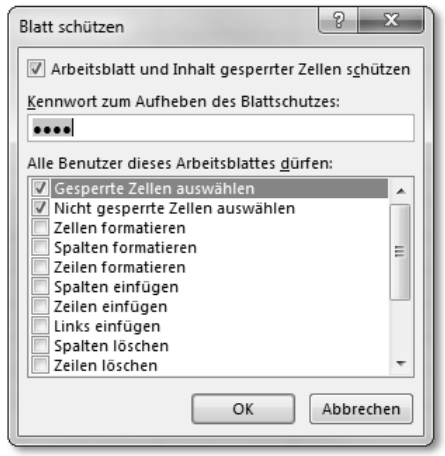


Abbildung 6.4 Der integrierte Dialog »Blatt schützen« wurde aufgerufen.

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 *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 Wert 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 spielt. Je nach Vergleichsergebnis werden die einzelnen Tabellen dann innerhalb der Arbeitsmappe mit Hilfe der Methode `Move` verschoben oder nicht.

6.8.5 Kopf- und Fußzeilen einrichten

Standardmäßig werden in Excel 2000 bis 2016 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 2016«).

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.

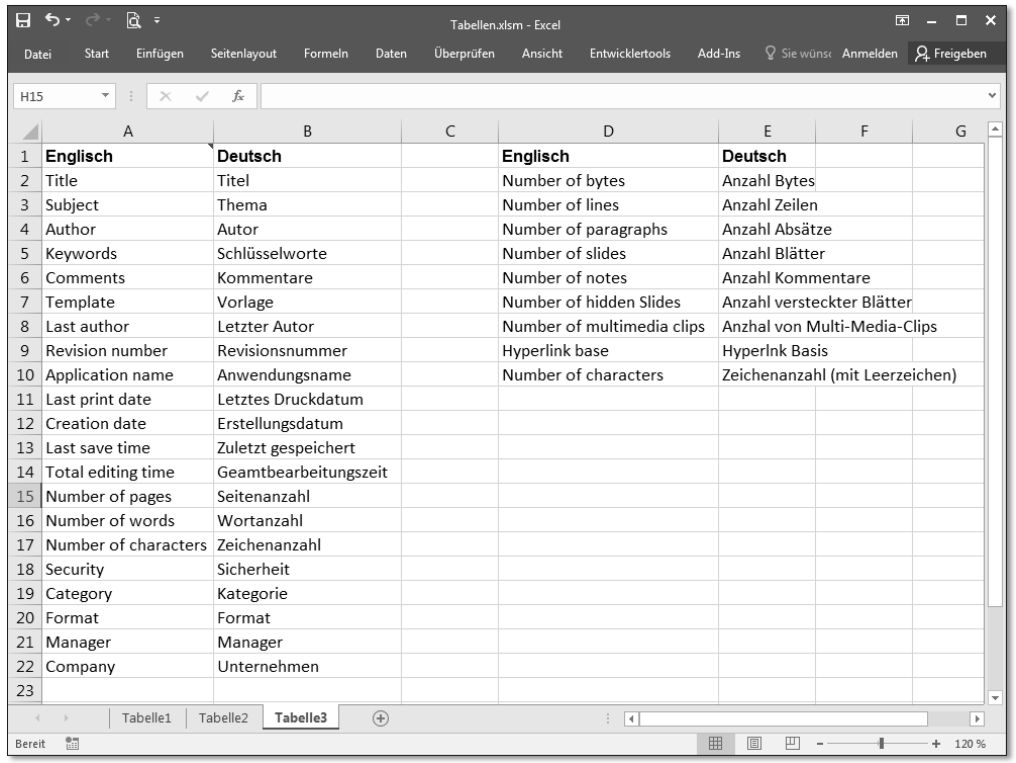
Anwender der Version Excel 2007 klicken auf die runde Office-Schaltfläche links oben und wählen aus dem Menü den Befehl VORBEREITEN • EIGENSCHAFTEN.

Anwender der Versionen Excel 2010 und 2016 klicken 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	Hyperlnk 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

Mit Hilfe 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 eine zweistellige Zahl an, um den Schriftgrad anzugeben.
&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 = _
    ThisWorkbook.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` wird der Abstand zum unteren Papierrand angegeben. Mit Hilfe der Eigenschaft `FooterMargin` wird der Abstand der Fußzeile ebenfalls vom unteren Papierrand eingestellt. 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 heruntergestellt 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 der 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 beziehungsweise 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 mit Hilfe 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 mit Hilfe der Eigenschaft `FullName` ermittelt.

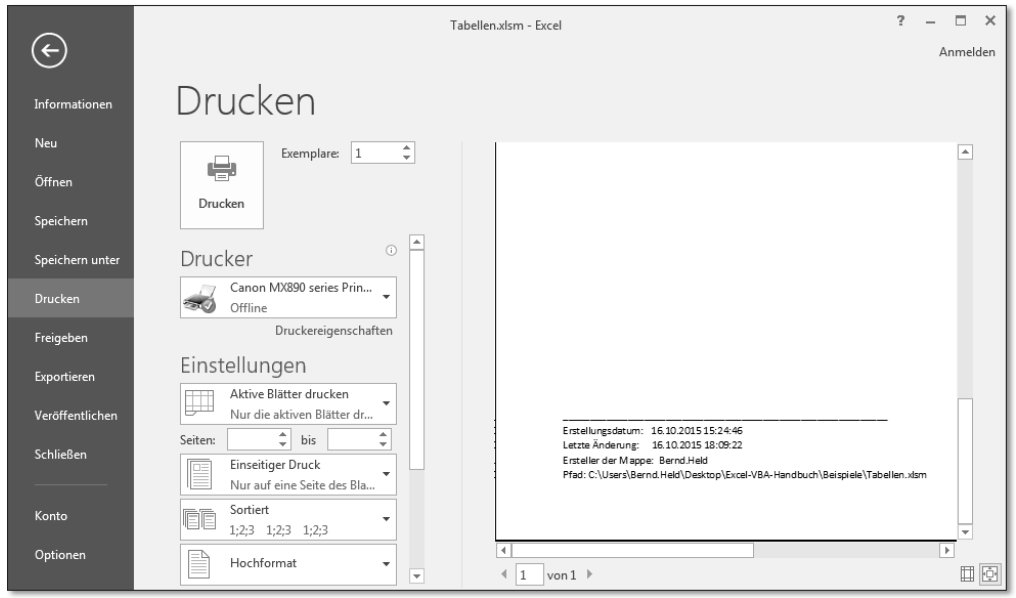


Abbildung 6.7 Eine mehrzeilige Fußzeile einfügen

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 Tabellen Gitternetz ein- und ausschalten

Sollten Sie das Gitternetz einer Tabelle ausschalten wollen, dann 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

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

Wenden Sie dann die Methode `SaveAs` an, um die Mappe zu speichern. Ergänzen Sie dabei mit Hilfe der Funktionen `Format` und `Date` den Dateinamen um einen Datumsstempel.

Mit Hilfe 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 mit Hilfe 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
```

Auf einen Blick

Inhalt

3 Die Sprachelemente von Excel-VBA 99

4 Zellen und Bereiche programmieren 159

8 Eigene Funktionen, reguläre Ausdrücke und API-Funktionen 491

9 Ereignisse programmieren 563

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Bernd Held ist von Haus aus gelernter Informatiker. Zu seinen Spezialgebieten zählen Excel, VBA-Programmierung, Access und allgemeine Office- und Tool-Themen. Er wurde von Microsoft mehrfach als MVP (Most Valuable Professional) für den Bereich Excel ausgezeichnet. Seit 2008 arbeitet er mit einem eigenen Team aus Experten zusammen, das Projekte und Schulungen durchführt, Unternehmen berät und Bücher sowie Fachartikel veröffentlicht.

Bernd Held

VBA mit Excel – Das umfassende Handbuch

950 Seiten, gebunden, 2. Auflage 2015
49,90 Euro, ISBN 978-3-8362-3821-2

www.rheinwerk-verlag.de/3891