Kapitel 4

Das Objekt Worksheet (Tabellen programmieren)

In diesem Kapitel:

Mithilfe des `Worksheet`-Objekts, das eine Tabelle repräsentiert, können Sie auf eine bzw. auch auf alle Tabellen einer Arbeitsmappe über die `Worksheets`-Auflistung zugreifen. In der Auflistung `Worksheets` sind standardmäßig alle Tabellen einer Arbeitsmappe verzeichnet.

Sehen Sie sich einmal Abbildung 4.1 an, um einen ersten Eindruck von der Befehlsvielfalt des Objekts `Worksheet` zu bekommen.

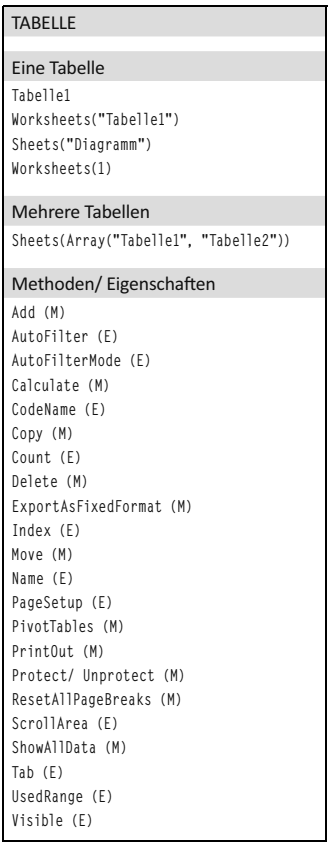


Abbildung 4.1: Die wichtigsten Methoden und Eigenschaften des Objekts Worksheet.

Im oberen Bereich von Abbildung 4.1 sehen Sie verschiedene Möglichkeiten, eine Tabelle anzusprechen. Wir haben dies bereits im ersten Kapitel dieses Buchs besprochen. Im unteren Bereich der Abbildung sehen die wichtigsten Eigenschaften (*E*) und Methoden (*M*) für dieses Objekt.

Die folgenden Beispiele sind in der Arbeitsmappe *Start.xlsm* im Modul *mdl_Tabellen* zu finden.

Lernen Sie auf den nächsten Seiten, wie Sie das Objekt `Worksheet` in der Praxis einsetzen können.

Tabellen dokumentieren, filtern und durchsuchen

Auf den nächsten Seiten erfahren Sie, wie Sie unter anderem ein Inhaltsverzeichnis aller Tabellen einer Arbeitsmappe erstellen und verlinken können. Des Weiteren werden Sie alle Tabellen einer Arbeitsmappe nach Formeln und Verknüpfungen durchsuchen und die Fundstellen dokumentieren.

Tabelleninhaltsverzeichnis erstellen und verlinken

Das erste Makro in diesem Kapitel ist schon der Knaller. Es erstellt in der Tabelle *tbl_Tabellen* ein Inhaltsverzeichnis aller Tabellen der Mappe und hinterlegt sogar für jeden Eintrag einen Hyperlink.

```
Sub TabellenListenUndInhaltsverzeichnisErstellen()
 Dim Blatt As Worksheet
 Dim Zeile As Long

  With tbl_Tabellen
    .UsedRange.Clear
    Zeile = 1

    For Each Blatt In Worksheets

      .Range("A" & Zeile).Value = Blatt.Name
      .Hyperlinks.Add Anchor:=.Range("A" & Zeile), _
                      Address:="", _
                      SubAddress:="'" & Blatt.Name & "'!A1"
       Zeile = Zeile + 1

    Next Blatt

  End With

End Sub
```

Listing 4.1: Ein Inhaltsverzeichnis aller Tabellen in Sekundenschnelle erstellt.

Zu Beginn des Makros aus Listing 4.1 deklarieren wir eine Variable vom Typ `Worksheet` mit dem Namen `Blatt`. Damit gewinnen wir automatisch Zugriff auf alle Methoden und Eigenschaften dieses Objekts. Die Variable `Zeile` vom Typ `Long` dient später zur Verwaltung der Zeilen in der Tabelle *tbl_Tabellen*. Nachdem wir mit der Anweisung `With` erklärt haben, in welcher Tabelle wir das Inhaltsverzeichnis anlegen möchten, wenden wir die Methode `Clear` auf den benutzten Bereich dieser Tabelle an, um den Bereich sicherheitshalber vorher zu löschen. Danach setzen wir den Inhalt der Variablen `Zeile` auf den Wert 1. Damit legen wir fest, dass der erste Eintrag ab der ersten Zeile dieser Tabelle geschrieben werden soll.

Mit einer `For Each...Next`-Schleife, die Sie im zweiten Kapitel, »Die wichtigsten Sprachelemente von Excel-VBA«, kennengelernt haben, arbeiten wir uns Tabelle für Tabelle in der aktuellen Arbeitsmappe vor und schreiben die Namen der Tabelle mithilfe der Eigenschaft `Name` hübsch untereinander in die Tabelle *tbl_Tabellen*.

Danach wenden wir die Methode `Add` an, um in der jeweiligen Zelle einen Hyperlink einzufügen. Im Parameter `Anchor` geben Sie den sogenannten Anker (die Zelle) des Hyperlinks an. Der Parameter `Address` bleibt für dieses Beispiel leer. Im letzten Parameter `SubAddress` geben Sie das Sprungziel des Hyperlinks an. Dieses Sprungziel besteht aus dem eigentlichen Namen der Tabelle sowie der gewünschten Zellenkoordinate.

Wir dürfen nicht vergessen, die Zählvariable `Zeile` nach jedem Schreibvorgang um den Wert 1 zu erhöhen, damit der nächste Tabellenname jeweils in die nächste Zeile geschrieben werden kann.

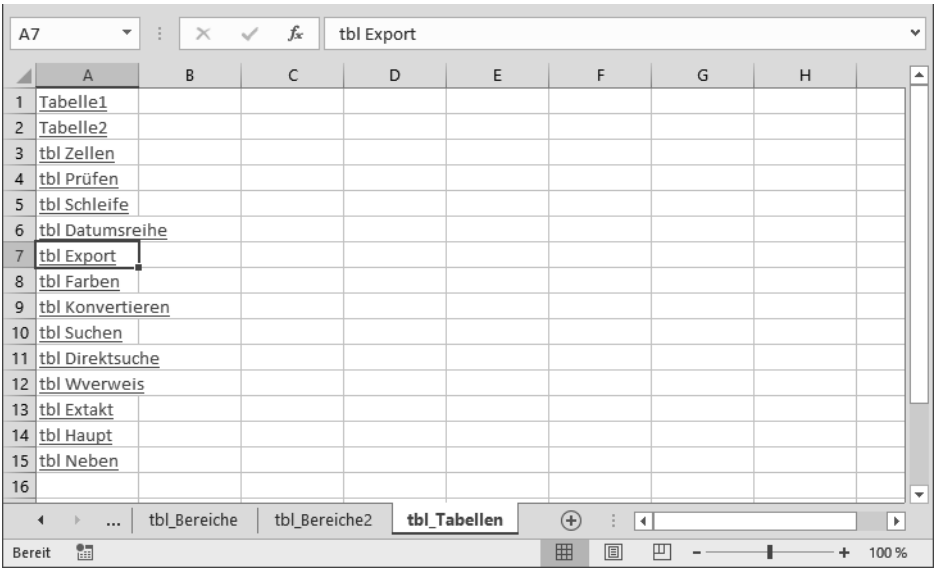


Abbildung 4.2: Alle Tabellen der Mappe werden untereinander aufgelistet.

Kennen Sie den Unterschied zwischen den Befehlen `Worksheets` und `Sheets`? Über den Befehl `Worksheets(1)` können Sie das erste Tabellenblatt einer Arbeitsmappe ansprechen, mit `Sheets(1)` sprechen Sie ebenfalls das erste Blatt in der Arbeitsmappe an.

Aber: `Sheets(1)` kann genauso gut auch ein Diagrammblatt sein, `Worksheets(1)` ist jedoch immer nur eine Excel-Tabelle. Da es für Diagramme andere Befehle als für normale Tabellen gibt, ist die Verwechslung beider Begriffe etwas gefährlich. Verwenden Sie also `Worksheets`, wenn Sie Tabellen ansprechen möchten!

Tabellen durchsuchen und dokumentieren

In der folgenden Aufgabe wird die komplette Arbeitsmappe nach Formeln und Verknüpfungen durchsucht. Die dabei gefundenen Zellen sollen im Direktfenster der Entwicklungsumgebung dokumentiert werden. Starten Sie zu diesem Zweck das Makro aus Listing 4.2.

```
Sub FormelKundschafter()
  Dim Blatt As Worksheet
  Dim zelle As Range

  'Äußere Schleife.
  For Each Blatt In Worksheets
```

```
    For Each zelle In Blatt.UsedRange

     If zelle.HasFormula Then
        Debug.Print Blatt.Name & " /" & zelle.Address & " / " & zelle.FormulaLocal
    End If

    Next zelle

  Next Blatt

End Sub
```

Listing 4.2: Alle Formeln werden inklusive ihrer Adresse im Direktfenster ausgegeben.

Zu Beginn des Makros deklarieren wir zwei Objektvariablen. Mithilfe der Variablen `Blatt` vom Typ `Worksheet` basteln wir die äußere `For Each...Next`-Schleife, die Tabelle für Tabelle in der Arbeitsmappe verarbeitet. Über den Einsatz der Variablen `Zelle` vom Typ `Range` verarbeiten wir alle Zellen des benutzten Bereichs und setzen dazu eine weitere `For Each...Next`-Schleife ein.

Über die Eigenschaft `HasFormula` können wir prüfen, ob die jeweilige Zelle eine Formel enthält. In diesem Fall geben wir den Namen der Tabelle über die Eigenschaft `Name`, die Koordinate über die Eigenschaft `Address` sowie den Formeltext über die Eigenschaft `FormulaLocal` im Direktfenster der Entwicklungsumgebung aus.

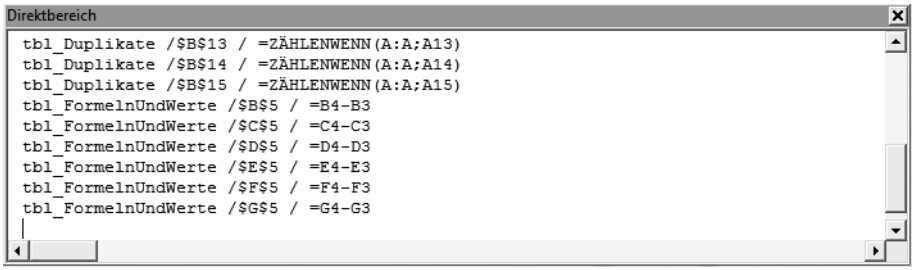
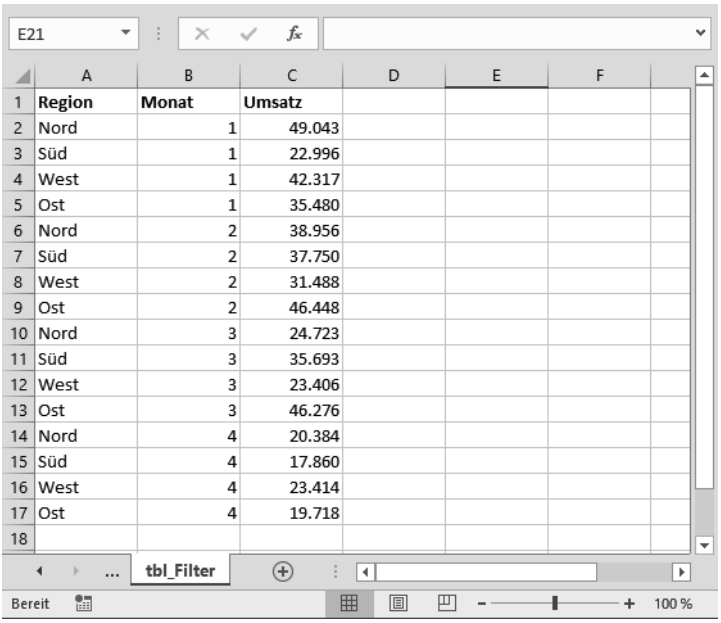


Abbildung 4.3: Namen der Tabelle, Koordinate und Formeltext wurden ausgelesen.

Tabellen filtern mit einem Kriterium

Sicher kennen Sie den Datenfilter in Excel, über den Sie bestimmte Zeilen einer Tabelle filtern können. Im folgenden Beispiel sollen in der Tabelle *tbl_Filter* alle Zeilen, die in Spalte A die Region *Süd* haben, gefiltert werden. Sehen Sie sich zunächst die Ausgangssituation in Abbildung 4.4 an.



	A	B	C	D	E	F
1	Region	Monat	Umsatz			
2	Nord	1	49.043			
3	Süd	1	22.996			
4	West	1	42.317			
5	Ost	1	35.480			
6	Nord	2	38.956			
7	Süd	2	37.750			
8	West	2	31.488			
9	Ost	2	46.448			
10	Nord	3	24.723			
11	Süd	3	35.693			
12	West	3	23.406			
13	Ost	3	46.276			
14	Nord	4	20.384			
15	Süd	4	17.860			
16	West	4	23.414			
17	Ost	4	19.718			
18						

Abbildung 4.4: Eine Umsatzliste nach Region und Monat.

Setzen Sie den Datenfilter wie in Listing 4.3 gezeigt ein.

```
Sub AutofilterRegionFiltern()

  With tbl_Filter

    If Not .AutoFilterMode = True Then
        .Range("A1").AutoFilter
    End If

    .Range("A1").AutoFilter Field:=1, Criteria1:="=Süd"

  End With

End Sub
```

Listing 4.3: Alle Zeilen mit der Region Süd werden gefiltert und angezeigt.

Mithilfe der Eigenschaft `AutoFilterMode` können Sie überprüfen, ob Filterpfeile in der Tabelle überhaupt angezeigt werden. Falls nicht, wenden Sie die Methode `AutoFilter` an, um die Filterpfeile einzublenden. Stellen Sie danach den gewünschten Filter ein und übergeben Sie im Parameter `Field` die Spaltennummer und im Parameter `Criteria1` das gewünschte Auswahlkriterium.

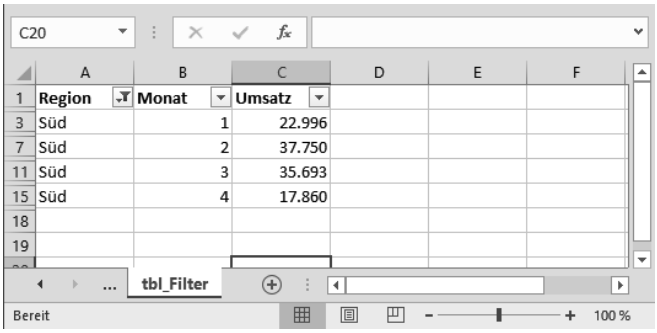


Abbildung 4.5: Alle Datensätze mit der Region Süd werden angezeigt.

Tabellen filtern mit mehreren Kriterien

Bei der nächsten Aufgabenstellung wird ein Filter angewendet, der zwei Kriterien erfüllen muss. So sollen alle Zeilen gefiltert werden, die im Wertebereich zwischen 20000 und 30000 liegen. Dazu erfassen Sie das Makro aus Listing 4.4.

```
Sub AutoFilterMehrereBedingungen()

  With tbl_Filter
      If Not .AutoFilterMode = True Then
          .Range("A1").AutoFilter
      End If

      .ShowAllData
      .Range("A1").AutoFilter Field:=3, Criteria1:=">=20000", _
                      Operator:=xlAnd, Criteria2:="<=30000"

      .Range("A1").Sort Key1:=.Range("C1"), Order1:=xlAscending, Header:=xlYes

    End With

End Sub
```

Listing 4.4: Alle Umsätze im Bereich von 20000 bis 30000 regionunabhängig filtern und sortieren.

Nach erfolgter Prüfung des Filterstatus mithilfe der Eigenschaft `AutoFilterMode` wird die Methode `ShowAllData` eingesetzt, um gegebenenfalls alle Zeilen in der Tabelle anzuzeigen. Anschließend wenden wir die Methode `AutoFilter` an und übergeben dieser zwei Kriterien über die Parameter `Criteria1` und `Criteria2`. Da ein Von-bis-Wertebereich abgefragt wird, verwenden wir als Operator das Schlüsselwort `xlAnd`. Direkt im Anschluss sortieren wir über die Methode `Sort` den Bereich auf Basis der Spalte C absteigend.

Seit Excel-Version 2007 gibt es zwar eine neue `Sort`-Methode, da diese neuere Methode aber leider den Nachteil hat, dass sie in früheren Versionen nicht einsetzbar ist und uns für die normale Sortierung keinerlei Vorteile bringt, habe ich mich entschieden, eine lauffähige Methode zu beschreiben, die wirklich in allen Excel-Versionen funktioniert.

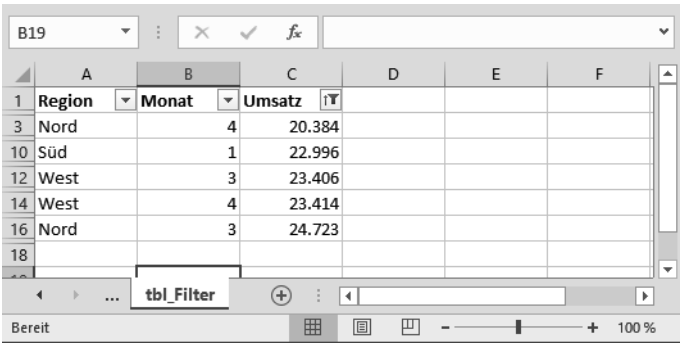


Abbildung 4.6: Filtern und anschließend sortieren in einem Arbeitsgang.

Tabellen einrichten und schützen

Auf den nächsten Seiten geht es rund um das Thema Schutz von Excel-Tabellen und um die Vorbereitung darauf. Sie erfahren, wie Sie Bildlaufbereiche einrichten, Tabellenbereiche sperren und Tabellen schützen.

Bildlaufbereiche für Tabellen festlegen

Im ersten Kapitel des Buchs sind wir bereits mit der Eigenschaft `ScrollArea` in Berührung gekommen. Dabei haben Sie erfahren, dass Sie den Bildlaufbereich einer Tabelle, also den Bereich einer Tabelle, in dem sich der Anwender aufhalten darf, zwar über das Eigenschaftenfenster setzen, aber leider eben nicht dauerhaft einstellen können.

Glücklicherweise können Sie diese Einstellung über das Ereignis `Workbook_Open` auch dauerhaft vornehmen. Führen Sie zur Festlegung des Codes, der bei Eintreten dieses Ereignisses ausgeführt werden soll, im Projekt-Explorer einen Doppelklick auf die Rubrik *DieseArbeitsmappe* aus und wählen Sie im linken Drop-down-Listenfeld über dem Codefenster das Objekt `Workbook` aus. Ergänzen Sie danach den Ereigniscode wie in Listing 4.5 gezeigt.

```
Private Sub Workbook_Open()
 'ScrollArea für einen bestimmten Bereich einer Tabelle einstellen
 tbl_ErsteMakros.ScrollArea = "A1:A10"

End Sub
```

Listing 4.5: Das Ereignis Open wird automatisch beim Öffnen der Mappe ausgeführt.

Geben Sie der Eigenschaft `ScrollArea` bekannt, wo der Aufenthaltsbereich in der Tabelle eingestellt werden soll.

Erfahren Sie mehr über den Einsatz von Ereignissen in Kapitel 7, »Die Ereignisprogrammierung in Excel«.

Bereiche in Tabellen sperren

Eine manuell eher etwas aufwendigere Aufgabe ist das Sperren von einzelnen Zellen, die Formeln enthalten. Viele Kalkulationen gehen schief, wenn versehentlich Formeln überschrieben oder gelöscht werden. Daher können Sie ganz gezielt bestimmte Zellen für eine Eingabe sperren. Dieser Sperrmechanismus ist jedoch standardmäßig nur in Verbindung mit dem Tabellenschutz möglich. Zur Vorbereitung auf den eigentlichen Schutz der Tabelle bestimmen Sie in der Tabelle *tbl_FormelnUndWerte* die Zellen, die Sie als nicht überschreibbar kennzeichnen möchten.

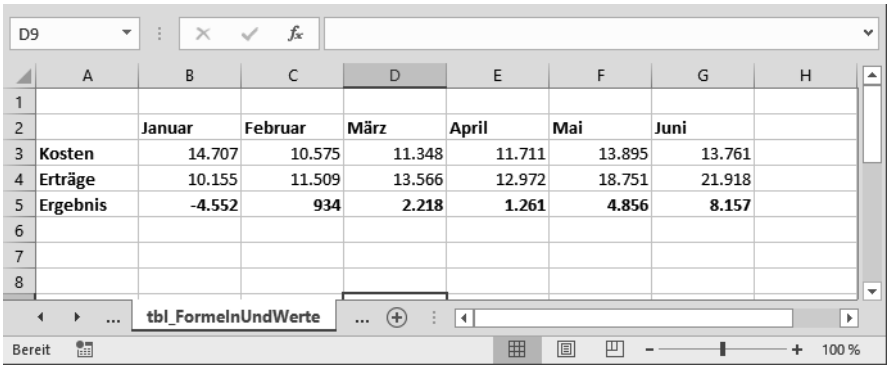


Abbildung 4.7: Eine Monatsdarstellung von Kosten und Erträgen.

In Abbildung 4.7 sehen Sie einige Zellen, die mit dem Schriftschnitt *Fett* formatiert sind. Genau diese Zellen sollen zukünftig geschützt werden. Um sie vor einer versehentlichen Löschung zu schützen, müssen Sie den Zellen das Merkmal *Gesperrt* über das Dialogfeld *Zellen formatieren/Schutz* zuweisen. Eine weitere Einstellungsmöglichkeit ist im gleichen Dialogfeld das Kontrollkästchen *Ausgeblendet*. Damit werden die eigentlichen Formeln versteckt. Beide Schalter sind aber, wie schon gesagt, erst nach der Aktivierung des Tabellenschutzes wirksam. Diesen Tabellenschutz werden wir gleich als Nächstes behandeln.

Die soeben beschriebenen manuell vorgenommenen Sperreinstellungen können Sie auch programmatisch festlegen, wie das Makro aus Listing 4.6 zeigt.

```
Sub TabelleVorbereiten()
  Dim Zelle As Range

  With tbl_FormelnUndWerte

    For Each Zelle In .UsedRange

      If Zelle.Font.Bold = True Then
         Zelle.Locked = True
         Zelle.FormulaHidden = True
      Else
         Zelle.Locked = False
         Zelle.FormulaHidden = False
      End If
```

```
    Next Zelle

  End With

End Sub
```

Listing 4.6: Das Sperrmerkmal nur für Zellen mit Fettschrift einstellen.

Rein optisch sehen Sie nach dem Start des Makros aus Listing 4.6 nichts. Wenn Sie aber das Dialogfeld *Zellen formatieren* für eine Zelle ohne Fettschrift über die Tastenkombination Strg+1 aufrufen, können Sie auf der Registerkarte *Schutz* erkennen, dass die Kontrollkästchen *Gesperrt* und *Ausgeblendet* nicht aktiviert sind. Führen Sie den Gegentest auch einmal für eine Zelle mit Fettschrift durch. Dort müssen die Kontrollkästchen *Gesperrt* und *Ausgeblendet* aktiviert sein.

Tabellenschutz für eine Tabelle einstellen und zurücksetzen

Nachdem Sie eben entschieden haben, welche Zellen einer Tabelle gesperrt sein sollen, können Sie nun damit beginnen, den Tabellenschutz für die Tabelle einzustellen. Im folgenden Makro aus Listing 4.7 wird ein Tabellenschutz für die Tabelle *tbl_FormelnUndWerte* eingestellt, und dabei wird das Kennwort *Test* festgelegt.

```
Sub TabellenSchutzAn()

   tbl_FormelnUndWerte.Protect Password:="Test"

End Sub
```

Listing 4.7: Die Tabelle wird mit einem Passwort geschützt.

In diesem Einzeiler wenden wir die Methode `Protect` an und übergeben im Parameter `Password` das gerade definierte Passwort. Dass das hier festgelegte Passwort eher ungeeignet ist, versteht sich von selbst. Dieses Passwort wird wahrscheinlich von jedem zuerst ausprobiert. Da Sie das Passwort ja nicht händisch erfassen müssen, können Sie sich Gedanken machen, auch ein etwas anspruchsvolleres Passwort anzugeben.

Machen Sie sich aber dazu nicht zu viele Gedanken – egal wie lang und schwer das eingestellte Passwort auch ist – es ist eine Angelegenheit von wenigen Sekunden, das Passwort aufzuhebeln. Dazu gibt es massenhaft Gratistools im Internet. Der Blattschutz von Excel ist eher ein Schutz vor versehentlichem Überschreiben von Daten, mehr nicht. Wenn Sie einen zuverlässigen Schutz Ihrer Mappen und Tabellen wünschen, kann ich Ihnen die Software *LockXLS* empfehlen, die ich selbst getestet und im Einsatz habe. Was jetzt noch fehlt, ist ein Makro, das uns die gerade geschützte Tabelle wieder entsperrt.

```
Sub TabellenSchutzAus()

   tbl_FormelnUndWerte.Unprotect Password:="Test"

End Sub
```

Listing 4.8: Den Tabellenschutz wieder aufheben.

Zum Aufheben des eingestellten Tabellenschutzes verwenden wir die Methode `Unprotect` und übergeben als Parameter das gleiche Kennwort wie in Listing 4.7.

Alle Tabellen einer Mappe schützen

Wenn Sie einmal alle Tabellen einer Arbeitsmappe händisch geschützt haben, wissen Sie, dass Sie diesen Vorgang wirklich Tabelle für Tabelle vornehmen müssen. Dabei müssen Sie das Kennwort jeweils zweimal eingeben. Das ist ein irrer Aufwand, der über den Einsatz eines Makros auf weniger als eine Sekunde verkürzt werden kann. Das Makro aus Listing 4.9 schützt alle Tabellen in der Mappe, wobei der Schutz hierbei etwas aufgeweicht wird. Es soll dem Anwender möglich sein, trotz des eingestellten Blattschutzes noch Zellen zu formatieren.

```
Sub TabellenSchützen()
  Dim Blatt As Worksheet

  For Each Blatt In Worksheets

    Blatt.Protect Password:="Test", AllowFormattingCells:=True

  Next Blatt

End Sub
```

Listing 4.9: Alle Tabellen in einem Aufwasch schützen.

Zu Beginn deklarieren wir eine Objektvariable vom Typ `Worksheet`. Danach wenden wir eine `For Each...Next`-Schleife an, um die Mappe Tabelle für Tabelle abzuarbeiten. Innerhalb der Schleife kommt die Methode `Protect` zum Einsatz. Über den Parameter `Password` geben wir das gewünschte Kennwort an. Im Argument `AllowFormattingCells` legen wir fest, dass der Anwender trotz eingestellten Blattschutzes Zellen formatieren darf.

Weitere Aufweichungen des Tabellenschutzes können Sie der Syntax entnehmen, wenn Sie die Methode `Protect` markieren und die Taste F1 drücken. Die Syntax dazu lautet etwas grob zusammengefasst:

```
Ausdruck.Protect(Password, DrawingObjects, Contents, Scenarios, UserInterfaceOnly,
AllowFormattingCells, AllowFormattingColumns, AllowFormattingRows, AllowInsertingColumns,
AllowInsertingRows, AllowInsertingHyperlinks, AllowDeletingColumns, AllowDeletingRows,
AllowSorting, AllowFiltering, AllowUsingPivotTables)
```

Welche Bedeutung diese Schalter haben, können Sie eigentlich ganz einfach in Erfahrung bringen. Wechseln Sie dazu zur normalen Excel-Oberfläche und klicken Sie im Menüband *Überprüfen* auf *Blatt schützen*.

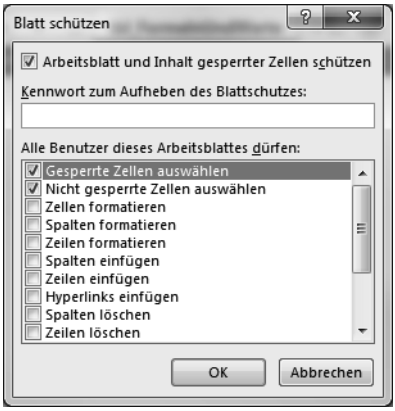


Abbildung 4.8: Die verfügbaren Schalter beim Tabellenschutz anwenden.

Die deutschsprachigen Optionen können mit den englischen Schaltern der Methode `Protect` gut abgeglichen werden.

Das Makro zum Entsperren aller Tabellen sehen Sie in Listing 4.10.

```
Sub TabellenEntsperren()
  Dim Blatt As Worksheet

  For Each Blatt In Worksheets

   Blatt.Unprotect Password:="Test"

  Next Blatt

End Sub
```

Listing 4.10: Alle Tabellen wieder entsperren

Wir wenden die Methode `UnProtect` in einer Schleife an und übergeben dieser Methode als einzigen Parameter das Kennwort, um den Tabellenschutz aufzuheben.

Ein kleines Sicherheitsleck gibt es noch. Wechseln Sie einmal zurück zur Standardoberfläche von Excel und klicken Sie im Menüband *Entwicklertools* auf die Schaltfläche *Makros*.

Abbildung 4.9: Unser Makro TabellenSchutzAus wird hier angezeigt.

Da die Tabellenschutzmakros in diesem Fenster angeboten werden, können sie auch von jedermann gestartet werden, was eher nicht so praktisch ist. Wie aber kann verhindert werden, dass sensible Makros dort angezeigt werden? Ganz einfach – indem Sie in den Kopf des jeweiligen Makros das Schlüsselwort `Private` schreiben.

```
Private Sub TabellenSchützen()
Private Sub TabellenEntsperren()
```

Wird ein Makro als `Private` deklariert, wird es nicht im *Makro*-Dialogfeld angezeigt.

Tabellenblätter anlegen, drucken und exportieren

Anhand der nachfolgenden Beispiele lernen Sie, wie Sie Tabellen anlegen, drucken und exportieren können.

Tabellen anlegen und benennen

Im nächsten Beispiel aus Listing 4.11 werden Sie eine neue Mappe anlegen und darin genau zwölf Tabellen einfügen. Diese sollen dann automatisch die Namen der zwölf Monate erhalten.

```
Sub MappeAnlegenMonatsTabellenEinfügen()
  Dim intAnzahl As Integer

  'Zunächst die eingestellte Anzahl von Tabellen aus dem System abfragen
  intAnzahl = Application.SheetsInNewWorkbook
```

```
  'dann diese Standardeinstellung temporär ändern
  Application.SheetsInNewWorkbook = 12

  'Arbeitsmappe mit diesen 12 Tabellen einfügen
  Workbooks.Add

  'und den ursprünglich eingestellten Wert wiederherstellen
  Application.SheetsInNewWorkbook = intAnzahl

  For intAnzahl = 1 To 12
    Worksheets(intAnzahl).Name = MonthName(intAnzahl)
  Next intAnzahl

End Sub
```

Listing 4.11: Eine neue Mappe anlegen, zwölf Tabellen einfügen und diese nach den Monaten benennen.

Zu Beginn des Makros deklarieren wir zunächst eine Variable vom Typ `Integer` mit dem wahlfreien Namen `intAnzahl`. Die Deklaration sorgt dafür, dass im Arbeitsspeicher Platz für unsere Variable reserviert wird. Aktuell hat diese Variable noch den Wert 0. Wir füllen die Variable jetzt, indem wir die Standardeinstellung für die Anzahl der Tabellen über die Eigenschaft `SheetsInNewWorkbook` abfragen. Diese Eigenschaft ist unterhalb des höchsten Objekts in Excel, also unter `Application`, angesiedelt. Wir überschreiben den standardmäßig eingestellten Wert nun mit den geforderten zwölf Tabellenblättern. Danach wenden wir die Methode `Add` an, um eine neue Arbeitsmappe anzulegen. Am Ende des Makros müssen wir den vorher eingestellten Standard wiederherstellen. Dazu weisen wir der Eigenschaft `SheetsInNewWorkbook` den Inhalt der Variablen `intAnzahl` zu.

Anschließend verwenden wir die Variable `intAnzahl` zu einem anderen Zweck, nämlich zum Zweck der Benennung der Tabellen. In einer `For...Next`-Schleife werden die einzelnen Tabellen nacheinander verarbeitet. In der Schleife geben wir den Tabellen die Namen der Monate eines Jahres. Dazu weisen wir der Eigenschaft `Name` das Ergebnis der Funktion `MonthName` zu. Diese Funktion erwartet einen Wert zwischen 1 und 12 und liefert Ihnen dazu automatisch den landesspezifischen Monatsnamen. Wenn Sie also in der Systemsteuerung ein anderes Land, beispielsweise Polnisch, einstellen, gibt das Makro die Monatsnamen auf Polnisch aus und benennt die Tabellen dementsprechend auch polnisch.

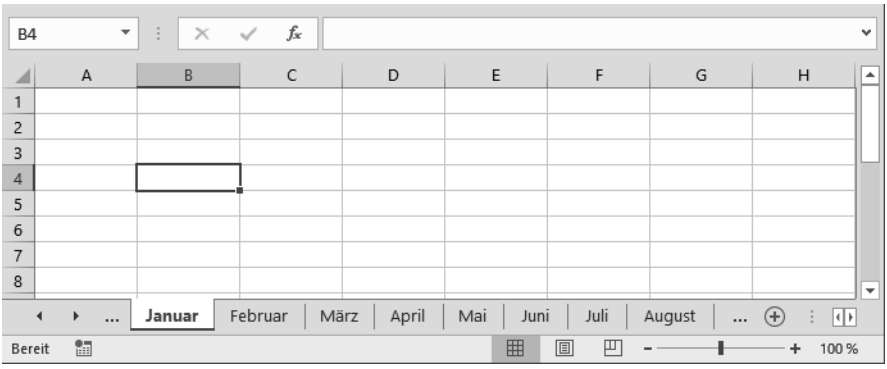


Abbildung 4.10: Die neue Mappe mit den zwölf Monaten wurde automatisch angelegt.

Eine Tabelle drucken

Für den normalen Ausdruck einer Tabelle steht Ihnen die Methode `PrintOut` zur Verfügung. Wenn es denn ein PDF sein soll, verwenden Sie die Methode `ExportAsFixedFormat`. Im Beispiel aus Abbildung 4.11 wird die Tabelle *tbl_Tabelle* gedruckt. Dazu wird vorher jedoch das Seitenformat noch festgelegt.

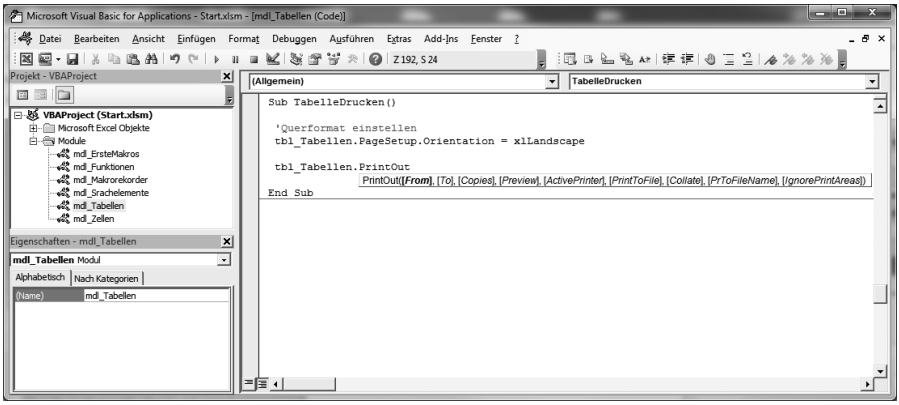


Abbildung 4.11: Der Ausdruck erfolgt über die Methode PrintOut.

Anhand der Quickinfo, die automatisch erscheint, wenn Sie die Methode `PrintOut` geschrieben und die Leertaste gedrückt haben, sehen Sie, dass diese Methode einige Parameter besitzt, mit denen Sie bestimmen können, was wo wie oft gedruckt werden soll. Die einzelnen Parameter sind in der Onlinehilfe anschaulich beschrieben, sodass ich an dieser Stelle nicht weiter darauf eingehen möchte.

Alle sichtbaren Tabellen einer Mappe ausdrucken

Im folgenden Beispiel aus Listing 4.12 sollen alle sichtbaren Tabellen einer Mappe gedruckt werden.

```
Sub NurEingeblendeteTabellenDrucken()
  Dim Blatt As Worksheet

  For Each Blatt In ThisWorkbook.Worksheets

   If Blatt.Visible = xlSheetVisible Then
      Blatt.PrintOut
   End If

  Next Blatt

End Sub
```

Listing 4.12: Alle sichtbaren Tabellen einer Mappe drucken.

In einer `For Each...Next`-Schleife werden alle Tabellen, bei denen uns die Eigenschaft `Visible` die Konstante `xlSheetVisible` meldet, mithilfe der Methode `PrintOut` ausgedruckt.

Tabelle als PDF ausgeben

Seit Excel-Version 2007 ist es standardmäßig möglich, eine Tabelle direkt im PDF-Format auszugeben. Im Makro aus Listing 4.13 wird die Tabelle *tbl_Tabellen* in ein PDF-Dokument umgewandelt und nach der Erstellung am Bildschirm angezeigt.

```
Sub TabelleAlsPDFAusgeben()

  tbl_Tabellen.ExportAsFixedFormat Type:=xlTypePDF, _
              Quality:=xlQualityStandard, OpenAfterPublish:=True

End Sub
```

Listing 4.13: Eine Tabelle in ein PDF-Dokument umwandeln.

Bei der Methode `ExportAsFixedFormat` gibt es einige Parameter, die Sie verwenden können, um beispielsweise die Druckqualität einzustellen oder auch die Möglichkeit zu nutzen, das PDF nach der Erstellung am Bildschirm anzuzeigen.

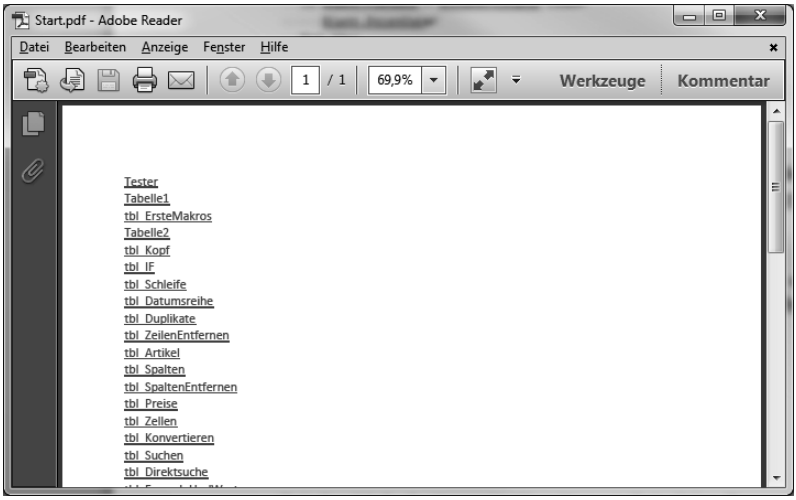


Abbildung 4.12: Das gerade erstellte PDF-Dokument wird automatisch geöffnet.

Eine Tabelle exportieren

Leider gibt es keinen gleichnamigen Befehl für den Export von Tabellen. Um eine Tabelle beispielsweise in eine neue Arbeitsmappe zu exportieren, wird die Methode `Copy`, wie in Listing 4.14 gezeigt, verwendet.

```
Sub TabelleExportieren()

   tbl_Tabellen.Copy

End Sub
```

Listing 4.14: Ein einfacher Copy-Befehl reicht aus, um eine Tabelle zu exportieren.

Individuelle Kopf- und Fußzeilen erstellen

Im folgenden Makro soll eine individuelle Kopf- und Fußzeile für alle Tabellen einer Arbeitsmappe erstellt werden. Da wir ja mehr können wollen als der normale Anwender in Excel, haben wir gerade für diese Aufgabenstellung zusätzliche Anforderungen eingebaut, die mit normalen Bordmitteln nicht möglich sind. Entnehmen Sie die Aufgabenstellung für dieses Beispiel direkt dem Makro aus Listing 4.15.

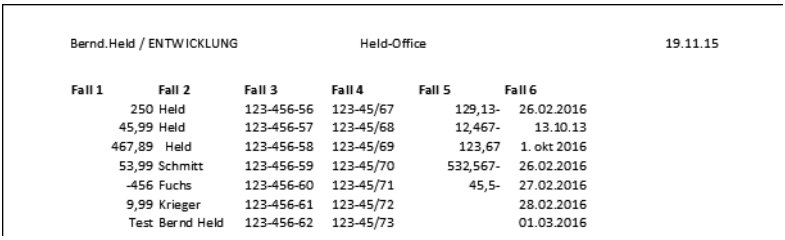
```
'Aufgabe: Individuelle Kopf- und Fußzeilen einrichten
'Kopfzeile, links  --> Anwendername lt. Windows-Anmeldung, Computernamen
'Kopfzeile, Mitte  --> Firmenname
'Kopfzeile, rechts --> Datum (Format: 12.03.16)
'Fußzeile, links   --> Erstelldatum & letzte Änderung (untereinander)
'Fußzeile, Mitte   --> Leer
'Fußzeile, rechts  --> Pfad, inkl. Dateinamen

Sub KopfUndFußzeileEinstellen()
 Dim Blatt As Worksheet
 Dim strFirma As String

 'Abfragen, ob die Dokumenteigenschaft Firma überhaupt gesetzt ist
 If ActiveWorkbook.BuiltinDocumentProperties("company") = "" Then
 'Wenn nicht, dann Firma über InputBox abfragen
  strFirma = InputBox("Bitte den Firmennamen eingeben", "Firma angeben")
  ActiveWorkbook.BuiltinDocumentProperties("company") = strFirma
 Else
  strFirma = ActiveWorkbook.BuiltinDocumentProperties("company")
 End If

 For Each Blatt In Worksheets

   With Blatt.PageSetup

     .LeftHeader = Environ("Username") & " / " & Environ("Computername")
     .CenterHeader = strFirma
     .RightHeader = Format(Date, "DD.MM.YY")
     .LeftFooter = "Erstelldatum: " & _
      Format(ActiveWorkbook.BuiltinDocumentProperties("creation date"), "DD.MM.YY") & vbLf & _
       "Letzte Änderung: " & _
        Format(ActiveWorkbook.BuiltinDocumentProperties("last save time"), "DD.MM.YY") & _
        vbLf & ActiveWorkbook.FullName
      .CenterFooter = "" 'Löscht einen evtl. gesetzten Eintrag
      .RightFooter = ""

   End With

 Next Blatt

End Sub
```

Listing 4.15: Eine individuelle Kopf- und Fußzeile zusammenstellen.

Der Firmenname kann über die Dokumenteigenschaft `BuiltinDocumentProperties("company")` abgefragt werden. Wenn diese standardmäßig nicht gesetzt ist, wird der Anwender dazu aufgefordert, seinen Firmennamen einzugeben. Der so eingegebene Firmenname wird im Anschluss direkt in die Dokumenteigenschaft geschrieben und dann über die Eigenschaft `CenterHeader` in die Mitte der Kopfzeile platziert.

Über die Eigenschaft `LeftHeader` können wir auf den linken Bereich der Kopfzeile zugreifen. Dort wenden wir die Funktion `Environ` an, über die wir mittels der Konstanten `Username` und `Computername` den Namen des angemeldeten Users sowie den Namen des Computers abfragen. Über die Eigenschaft `RightHeader` haben wir Zugriff auf den rechten Teil der Kopfzeile. Dort schreiben wir ein formatiertes Datum hinein. Das Datum erhalten wir direkt aus der Funktion `Date`, das gewünschte Format stellen wir mithilfe der Funktion `Format` ein.

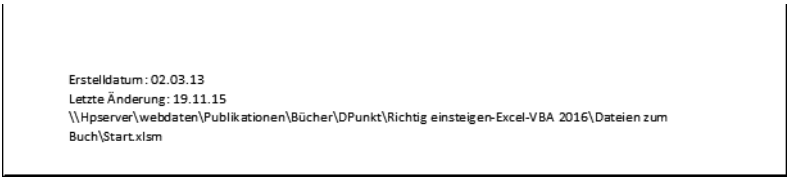
Für die Fußzeile setzen wir die Eigenschaft `LeftFooter` ein, um Zugriff auf den linken Bereich der Fußzeile zu erhalten. Dort sollen einige Zeilen untereinandergeschrieben werden. Den Zeilenumbruch bekommen wir mithilfe der Konstanten `vbLf` hin. Die Informationen darüber, wann die Mappe angelegt und wann die letzte Änderung daran registriert wurde, erhalten wir über die Konstanten `creation date` und `Last save time`, die wir direkt aus den Dokumenteigenschaften beziehen. Den Eigenschaften `CenterFooter` und `RightFooter` weisen wir einen sogenannten Leerstring zu. Damit wird gleichzeitig ein eventuell vorher dort stehender Text gelöscht.



Bernd.Held / ENTWICKLUNG Held-Office 19.11.15

Fall 1	Fall 2	Fall 3	Fall 4	Fall 5	Fall 6
250	Held	123-456-56	123-45/67	129,13-	26.02.2016
45,99	Held	123-456-57	123-45/68	12,467-	13.10.13
467,89	Held	123-456-58	123-45/69	123,67	1. okt 2016
53,99	Schmitt	123-456-59	123-45/70	532,567-	26.02.2016
-456	Fuchs	123-456-60	123-45/71	45,5-	27.02.2016
9,99	Krieger	123-456-61	123-45/72		28.02.2016
Test	Bernd Held	123-456-62	123-45/73		01.03.2016

Abbildung 4.13: Die Kopfzeile …



Erstelldatum: 02.03.13
Letzte Änderung: 19.11.15
\\Hpserver\webdaten\Publikationen\Bücher\DPunkt\Richtig einsteigen-Excel-VBA 2016\Dateien zum Buch\Start.xlsm

Abbildung 4.14: … sowie die dazugehörige Fußzeile.

Tabellen verstecken oder löschen

Auf den folgenden Seiten beschreibe ich, wie Sie Tabellen ein- und ausblenden sowie einzelne Tabellen löschen können.

Tabellen ein- und ausblenden

Erinnern Sie sich an den Anfang des Buchs? Dort haben wir gemeinsam über das Eigenschaftenfenster bestimmte Tabellen ein- und ausgeblendet, indem wir der Eigenschaft `Visible` eine Konstante zugewiesen haben. Im folgenden Makro aus Listing 4.16 wird die Tabelle *tbl_Tabellen* ein- und wieder ausgeblendet.

```
Sub TabelleEinUndAusblenden()

  If tbl_Tabellen.Visible = xlSheetVisible Then
     tbl_Tabellen.Visible = xlSheetHidden
  Else
     tbl_Tabellen.Visible = xlSheetVisible
  End If

End Sub
```

Listing 4.16: Ein Makro, das beides kann.

Im Makro aus Listing 4.16 wird geprüft, wie der augenblickliche Status der Tabelle ist. Ist die Tabelle sichtbar, wird sie ausgeblendet. Ist sie ausgeblendet, wird sie eingeblendet.

Alle Tabellen bis auf eine ausblenden

Im folgenden Makro aus Listing 4.17 werden alle Tabellen mit Ausnahme der Tabelle *tbl_Tabellen* ausgeblendet.

```
Sub AlleTabellenBisAufEineAusblenden()
  Dim Blatt As Worksheet

  For Each Blatt In Worksheets

    Select Case Blatt.CodeName

      Case "tbl_Tabellen"

      Case Else
        Blatt.Visible = xlSheetVeryHidden

    End Select

  Next Blatt

End Sub
```

Listing 4.17: Alle Tabellen bis auf die Tabelle tbl_Tabellen werden ausgeblendet.

Auch für diese Aufgabe können wir die `For Each...Next`-Schleife verwenden, um alle Tabellen nacheinander abzuarbeiten. Innerhalb der Schleife arbeiten wir mit der `Select Case`-Struktur und fragen den Codenamen der Tabelle über die entsprechende Eigenschaft ab.

Das Makro bleibt nur dann dauerhaft lauffähig, wenn Sie auf den Codenamen der Tabelle zugreifen, auf den der normale Anwender keinen Zugriff hat. Das bedeutet, dass der Standardanwender die Tabellen benennen kann, wie er möchte, und Ihr Makro wird trotzdem weiterhin wie gewünscht ausgeführt. Der `Case Else`-Zweig wird für alle Tabellen durchlaufen, deren Codename nicht *tbl_Tabellen* lautet. Diese Tabellen werden über die Konstante `xlSheetVeryHidden`, die Sie der Eigenschaft `Visible` zuweisen, auf sichere Art und Weise ausgeblendet.

Möchten Sie noch weitere Tabellen im eingeblendeten Status erhalten, erweitern Sie den ersten `Case`-Zweig um weitere Tabellennamen, jeweils getrennt voneinander durch ein Komma und eingeschlossen in doppelte Anführungszeichen zu Beginn und am Ende des jeweiligen Tabellennamens.

```
Case "tbl_Tabellen", "tbl_Eingabe", "tbl_Schleife"
```

In diesem Fall würden diese drei Tabellen nicht ausgeblendet werden.

Um die gerade ausgeblendeten Tabellen wieder sichtbar zu machen, wenden Sie das Makro aus Listing 4.18 an.

```
Sub AlleTabellenAnzeigen()
  Dim Blatt As Worksheet

  For Each Blatt In Worksheets

        Blatt.Visible = xlSheetVisible

  Next Blatt

End Sub
```

Listing 4.18: Alle Tabellen nacheinander mithilfe einer Schleife wieder sichtbar machen.

In der Schleife weisen wir der Eigenschaft `Visible` die Konstante `xlSheetVisible` zu, um die Tabellen wieder einzublenden.

Tabellen löschen

Im folgenden Beispiel sollen einige Tabellen aus der Arbeitsmappe entfernt werden. Da bereits bestehende Beispieltabellen nicht gelöscht werden sollen, fügen wir zur Laufzeit des Makros eine neue Tabelle hinzu, die dann gleich wieder gelöscht wird.

```
Sub TabelleEinfügenUndWiederLöschen()

 'Zuerst eine neue Tabelle am Ende der Mappe einfügen
  Worksheets.Add after:=Worksheets(Worksheets.Count)

  'Rückfrage vor dem Löschen der Tabelle temporär ausschalten
  Application.DisplayAlerts = False

  'Tabelle löschen
  ActiveSheet.Delete

  'Rückfrage von Excel wieder einschalten
  Application.DisplayAlerts = True

End Sub
```

Listing 4.19: Eine Tabelle am Ende der Mappe einfügen und wieder löschen.

Damit Sie überhaupt etwas sehen, arbeiten Sie das Makro aus Listing 4.19 bitte Schritt für Schritt im Einzelschrittmodus durch (Taste F8). Mithilfe der Methode `Add` fügen wir der `Worksheets`-Auflistung eine weitere Tabelle am Ende der Mappe hinzu. Um die letzte Tabelle der Mappe zu ermitteln, verwenden wir die Funktion `Count`. Beim Versuch, eine Tabelle zu löschen, meckert Excel normalerweise und fragt Sie, ob Sie die Tabelle wirklich löschen möchten.

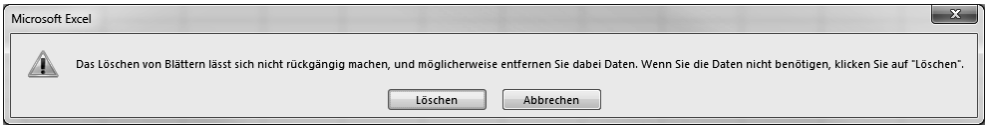


Abbildung 4.15: Die Rückfrage erscheint automatisch beim Versuch, eine Tabelle zu entfernen.

Diese Rückfrage können wir bei der Programmierung von Makros überhaupt nicht gebrauchen, weil wir ja genau wissen, was wir tun. Daher schalten wir diese Rückfragen über das Anwendungsobjekt (`Application`) direkt ab, indem wir der Eigenschaft `DisplayAlerts` den Wert `False` zuweisen.

Zusammenfassung

Sie haben in diesem Kapitel gelernt, wie Sie mithilfe des Objekts `Worksheet` Tabellen programmieren können. Das Objekt `Worksheet` wird dabei sehr oft mit der `For Each...Next`-Schleife verwendet.

In Kapitel 5, »Das Objekt Workbook (Arbeitsmappen programmieren)«, gehen wir in der Objekthierarchie von Excel wieder eine Ebene nach oben. Über dem Objekt `Worksheet` befindet sich das Objekt `Workbook`.

Die Lernkontrolle

Zum Abschluss dieses Kapitels stelle ich Ihnen ein paar Verständnisfragen. Die Antworten auf die hier gestellten Fragen finden Sie im Anhang dieses Buchs.

- »Welche Risiken birgt die Anweisung `Sheets(1).Range("A1").Value = 100`?«
- »Welche drei Konstanten weist die Eigenschaft `Visible` auf?«
- »Über welche Eigenschaft können Sie die Anzahl der Tabellen für eine neue Mappe festlegen?«
- »Wie heißt die Methode, um eine Tabelle in eine neue Mappe zu exportieren?«
- »Wie können Sie verhindern, dass Makros im *Makro*-Fenster angezeigt werden?«
- »Wie können Sie das Anlagedatum einer Mappe abfragen?«
- »Wie heißt der Schalter, den Sie verwenden, um Rückfragen in Excel temporär lahmzulegen?«