Kapitel 7

Mit Schleifen und Zusammenfassungen arbeiten

In diesem Kapitel:

Microsoft Excel Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-460-6

Schleifen werden eingesetzt, wenn innerhalb des Codes eine Wiederholung bis zu einem bestimmten Ereignis erfolgen soll. Es gibt verschiedene Schleifenarten, die je nach Einsatzgebiet verwendet werden können. Welche das sind, werden Sie auf den folgenden Seiten erfahren. Sie lernen zudem, wie mittels `With`-Anweisungen Codeblöcke zusammengefasst und möglichst übersichtlich gestaltet werden können.

Mit Schleifen arbeiten

In VBA gibt es verschiedene Varianten an Schleifen, die je nach Anwendungsgebiet eingesetzt werden können. Die wichtigsten davon werden Ihnen in diesem Kapitel vorgestellt.

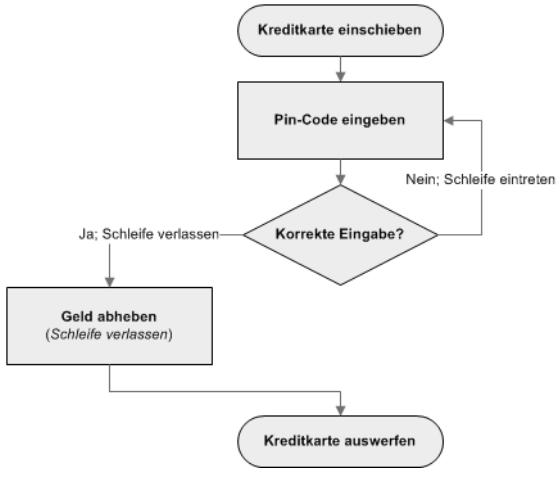
BEISPIEL Die Beispiele zum Thema *Schleifen* befinden sich innerhalb der Begleitdateien zum Buch im Ordner *\Buch\Kap07*. Die Mappe nennt sich *Bsp07_01.xlsm*. Die Prozeduren befinden sich in *Modul1*.

Können Sie sich noch an unseren Bankautomaten aus Kapitel 6 erinnern? An ihm lässt sich auch sehr schön erläutern, was in einer Schleife geschieht.

Sie stehen also wiederum vor dem Bankautomaten und schieben Ihre Kreditkarte ein. Als Erstes werden Sie nach Ihrem Pin-Code gefragt. Bei all den Passwörtern und Pin-Codes, die man sich heutzutage merken muss, kann man schnell etwas durcheinander bringen. Sie geben einen Code ein und haben leider den falschen verwendet. Der Bankautomat gibt Ihnen eine weitere Möglichkeit und Sie können den Code nochmals eingeben. Ohne es zu merken, befinden Sie sich schon mitten in einer Schleife. Der Bankautomat wird Sie erst zu einer weiteren Aktion leiten, wenn Sie den Code richtig eingegeben haben.

Um den Sachverhalt zu verdeutlichen, sehen wir uns auch diesmal eine Skizze an (siehe Abbildung 7.1). Natürlich ist das Ganze sehr vereinfacht dargestellt, denn der Bankautomat würde es kaum zulassen, dass Sie den Pin-Code beliebig oft eingeben.

Abbildg. 7.1 Eine Schleife auf dem Papier entwerfen



For...Next-Schleifen durchlaufen

Bei `For...Next` handelt es sich um eine Zählschleife. Dies bedeutet, dass ein Zähler verwendet wird, der bei jedem Schleifendurchlauf erhöht oder reduziert wird. Der Zähler sagt aus, wie oft die Schleife durchlaufen werden soll. Für gewöhnlich wird dazu eine Variable verwendet. Um die Variable einer Zählschleife leicht zu erkennen, haben sich viele Programmierer darauf geeinigt, sie als `i` zu bezeichnen.

Um zu verdeutlichen, was genau in den nachstehenden Beispielen geschieht, erfolgt die Ausgabe jeweils im zweiten Tabellenblatt der Arbeitsmappe, in der sich der Code befindet (`ThisWorkbook`), jeweils in einer anderen Spalte.

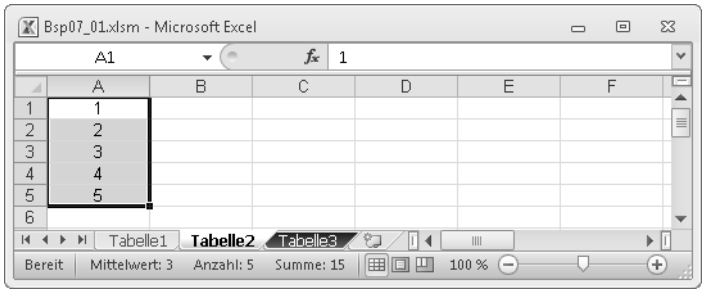
In der untenstehenden Prozedur wird ein einfacher Zähler verwendet. Die Schleife wird fünfmal durchlaufen. Bei jedem Durchlauf wird der Zähler um den Wert 1 erhöht. Den Zähler benutzen wir zugleich als Zeilenindex. Auf dem Tabellenblatt werden somit in Spalte *A* fünf Zahlen eingetragen.

Listing 7.1 *For...Next* mit Zähler

```
Sub ForNext_Zaehler()
   Dim i As Integer

   For i = 1 To 5
      ThisWorkbook.Worksheets(2).Cells(i, 1) = i
   Next i
End Sub
```

Abbildg. 7.2 Die Schleife wurde fünfmal in Einzelschritten durchlaufen und dabei jeweils um 1 erhöht



TIPP Falls eine Schleife sehr umfangreich ist und die Ausführung lange andauert, haben Sie die Möglichkeit, sie vorzeitig zu verlassen. Drücken Sie dazu die Esc-Taste. Unter Umständen kann es eine Weile dauern, bis eine Reaktion erfolgt. Warten Sie ab, bis das Debugger-Fenster angezeigt wird, und klicken Sie darin auf die Schaltfläche *Beenden*.

For...Next-Schleifen schrittweise durchlaufen

Eine Schleife muss nicht zwingend in Einzelschritten ausgeführt werden. Sie können die Schrittweite selbst bestimmen, indem Sie das Schlüsselwort `Step` verwenden.

Im nachfolgenden Beispiel wird der Zähler von 1 bis 50 festgelegt. Da die Schrittweite 10 beträgt, werden auf dem Tabellenblatt wiederum fünf Zahlen eingetragen. Die Ausgabe erfolgt in Spalte *B*.

Microsoft Excel Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-460-6

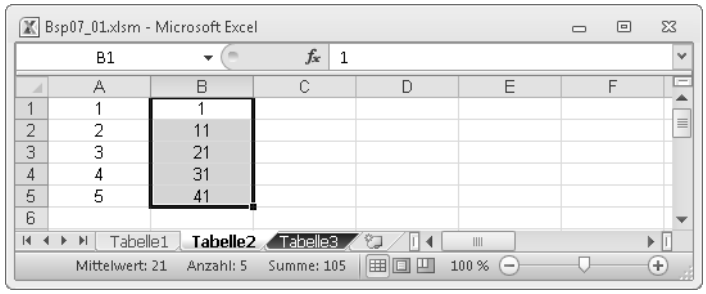
Diesmal können wir die Variable i nicht als Zeilenindex verwenden, da sonst zwischen den Zeilen jeweils neun leere Zellen entstehen würden. Die Ausgabe soll in einem Block erfolgen. Wir nehmen deshalb die Variable x zu Hilfe und platzieren sie innerhalb der Schleife. Sie wird bei jedem Durchlauf um den Wert 1 erhöht und dient als Zeilenindex.

Listing 7.2 *For...Next* stufenweise durchlaufen

```
Sub ForNext_Schrittweise()
   Dim i As Integer
   Dim x As Integer

   For i = 1 To 50 Step 10
      x = x + 1
      ThisWorkbook.Worksheets(2).Cells(x, 2) = i
   Next i
End Sub
```

Abbildg. 7.3 Die Schleife wurde fünfmal durchlaufen und jeweils um einen 10er-Schritt erhöht



For...Next-Schleifen rückwärts durchlaufen

Sie können eine Schleife auch rückwärts durchlaufen. Beachten Sie, dass wir hier den Zähler bei 50 ansetzen und dann in 10er-Minusschritten rückwärts zählen.

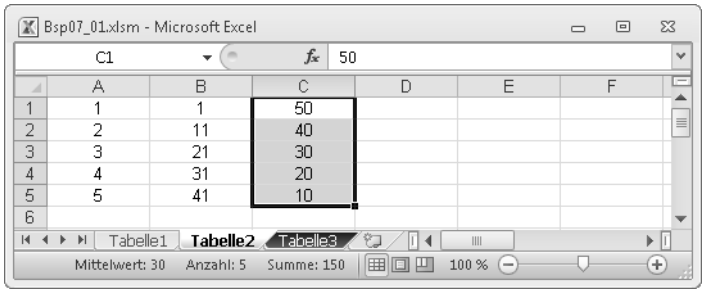
Die fünf Zahlen werden in die Spalte *C* eingetragen. Die Reihenfolge ist diesmal, aufgrund des Rückwärtsdurchlaufes, absteigend.

Listing 7.3 *For...Next* rückwärts durchlaufen

```
Sub ForNext_Rueckwaerts()
   Dim i As Integer
   Dim x As Integer

   For i = 50 To 1 Step -10
      x = x + 1
      ThisWorkbook.Worksheets(2).Cells(x, 3) = i
   Next i
End Sub
```

Abbildg. 7.4 Die Schleife wurde fünfmal in 10er-Rückwärtsschritten durchlaufen



For…Next-Schleifen vorzeitig verlassen

Sie können eine Schleife vorzeitig verlassen, wenn eine bestimmte Bedingung erfüllt wird. Dies geschieht hier in einer `If`-Entscheidung. Es ist zwar geplant, dass die Schleife zehnmal durchlaufen wird, dennoch wird sie abgebrochen, wenn der Zähler > 5 erreicht. Um aus der Schleife auszutreten, wird die Anweisung `Exit For` angewendet.

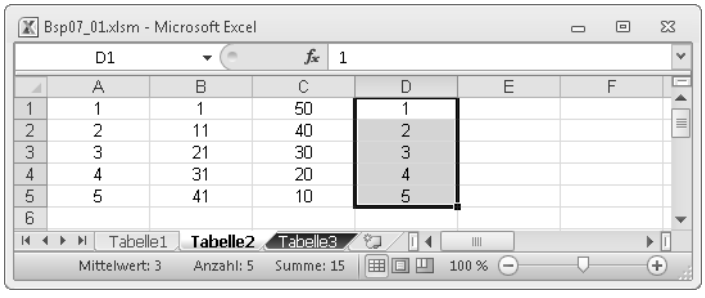
Listing 7.4 *For…Next* vorzeitig verlassen

```
Sub ForNext_Verlassen()
   Dim i As Integer

   For i = 1 To 10
      If i > 5 Then
         Exit For
      End If

      ThisWorkbook.Worksheets(2).Cells(i, 4) = i
   Next i
End Sub
```

Abbildg. 7.5 Die Schleife wurde nach fünf Durchläufen abgebrochen



For…Next-Schleifen verschachteln

Sie haben die Möglichkeit, mehrere Schleifen ineinander zu verschachteln. Dabei entstehen eine äußere und eine innere Schleife.

Für unser Beispiel nehmen wir einen weiteren `i`-Zähler hinzu. Die Variable `i1` dient der äußeren Schleife und die Variable `i2` der inneren.

Die äußere Schleife wird im folgenden Beispiel dreimal durchlaufen; die innere jedoch neunmal. Der Grund für diese »Ungleichheit« liegt darin, dass bei jedem Durchlauf der äußeren Schleife die gesamte innere Schleife mit ausgeführt wird.

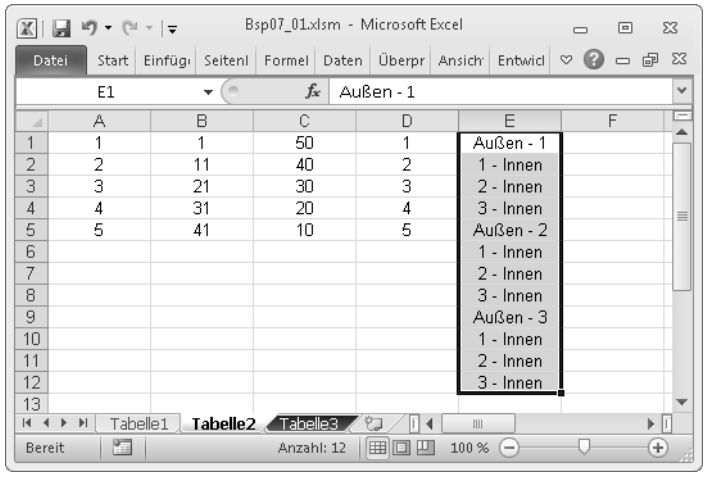
Listing 7.5 Schleifen verschachteln

```
Sub ForNext_Verschachtelt()
   Dim i1 As Integer
   Dim i2 As Integer
   Dim x As Integer

   For i1 = 1 To 3
      x = x + 1
      ThisWorkbook.Worksheets(2).Cells(x, 5) = "Außen - " & i1

      For i2 = 1 To 3
         x = x + 1
         ThisWorkbook.Worksheets(2).Cells(x, 5) = i2 & " - Innen"
      Next i2
   Next i1
End Sub
```

Abbildg. 7.6 Der Verhalten der beiden verschachtelten Schleifen



Microsoft Excel Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-460-6

Mittels *For Each* Tabellenblätter ansprechen

`For Each`-Schleifen werden im Zusammenhang mit Objekten wie zum Beispiel Zellen oder Tabellenblättern eingesetzt. Nach Möglichkeit sollte diese Schleife der `For...Next`-Schleife vorgezogen werden, denn sie ist schneller. Die Schleife wird jeweils durch eine `For Each`-Anweisung eingeleitet und muss mit einer `Next`-Zeile abgeschlossen werden.

HINWEIS Beim Arbeiten mit einer `For Each`-Schleife ist es wichtig, bei der Variablendeklaration das richtige Objekt anzugeben. Manchmal ist es nicht ganz einfach, dieses zu ermitteln. Alternativ zum Objekt kann in der Regel auch mit dem Datentyp `Variant` gearbeitet werden.

Manuell kann es sehr umständlich sein, jedem einzelnen Tabellenblatt einer Arbeitsmappe einen Blattschutz zuzuweisen. Jedes Tabellenblatt muss dabei einzeln aktiviert werden. Mit der folgenden Prozedur wird dies automatisiert.

Zu Beginn der Prozedur wird das Objekt `Worksheet` an die Variable `ws` übergeben. Dieses wir innerhalb der Schleife vermehrt verwendet. Unter anderem im Kopf der Schleife. Dort wird festgelegt, dass jedes `Worksheet` der Sammlung `Worksheets` durchlaufen wird, also jedes Tabellenblatt. Die Methode für den Blattschutz nennt sich `Protect`.

Listing 7.6 Blattschutz für jedes Tabellenblatt einrichten

```
Sub BlattschutzMappeEin()
   Dim ws As Worksheet

   For Each ws In Worksheets
      ws.Protect
   Next ws
End Sub
```

Um den Blattschutz für alle Tabellenblatter wieder zu entfernen, erstellen wir eine zweite Prozedur. Das Pendant zu `Protect` lautet `Unprotect`.

Listing 7.7 Blattschutz wieder aufheben

```
Sub BlattschutzMappeAus()
   Dim ws As Worksheet

   For Each ws In Worksheets
      ws.Unprotect
   Next ws
End Sub
```

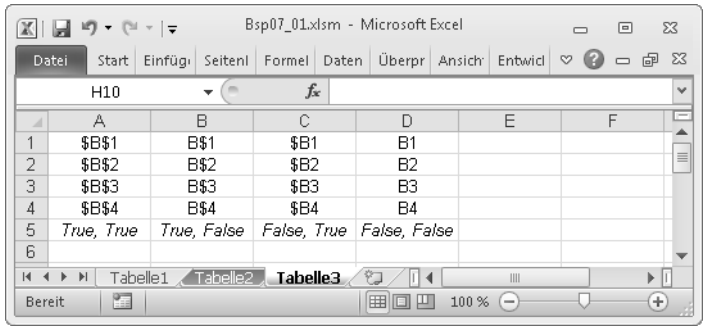
Mittels *For Each* Zellen abarbeiten

Sehen wir uns ein weiteres Beispiel in Bezug auf `For Each`-Schleifen an. Diesmal werden die Zellen des Bereiches *B1:B4* durchlaufen. Unser Ziel ist es, die Zelladressen zu ermitteln. Dies geschieht mittels der Eigenschaft `Address`. Diese gibt, ohne die Angabe von Argumenten, einen absoluten Zellbezug zurück *B1*, *B2* usw. Jedes der beiden Dollarzeichen kann mittels des booleschen Wertes `False` ausgeblendet werden. Im runden Klammerpaar von `Address` können zwei Argumente eingetragen werden, für jedes Dollerzeichen eines. Um beide Dollarzeichen zu unterdrücken, lautet die Anweisung somit `Address(False, False)`. Dadurch erfolgt eine Rückgabe von *B1*, *B2* usw.

Microsoft Excel Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-460-6

Das erste `False` steht für den Zeilenbezug und das zweite für den Spaltenbezug. Das erscheint auf den ersten Blick verdreht, folgt aber demselben Prinzip, dass in VBA immer wieder bei Indexangaben für Zellen auszumachen ist, wie Sie noch erfahren werden. An erster Stelle steht die Zeile und erst danach folgt die Spalte.

Abbildg. 7.7 Zelladressen und deren Argumente (absolut und relativ)



In unserer Prozedur verwenden wir zwei Variablen. Die Variable `i` wird benutzt, um die Zelladressen untereinander in die einzelnen Zellen zu schreiben. Die Variable `c` benötigen wir für die Objekt-Schleife. Der Buchstabe `c` steht für `Cell` und somit für Zelle. Sie können theoretisch auch jeden anderen Buchstaben benutzen, aber in der VBA-Programmierung hat sich generell dieser Buchstabe für Zellenobjekte eingebürgert.

Die zweite Variable, den Buchstaben `i`, verwenden wir, um die Zelladressen des Bereichs *B1:B4* untereinander in die einzelnen Zellen einzutragen. Der Zähler `i` muss bei jedem Schleifendurchlauf um den Wert 1 erhöht werden.

Die Ausgabe erfolgt auf dem dritten Tabellenblatt in unserer Beispielmappe. Damit das Tabellenblatt nicht in nahezu jeder Codezeile wiederholt eingetragen wird, fassen wir es in einer `With`-Anweisung zusammen. Näheres zum Thema Zusammenfassungen erfahren Sie später in diesem Kapitel.

Listing 7.8 Zelladressen für den Bereich *B1:B4* in einer *For Each*-Schleife ermitteln

```
Sub Zelladressen()
   Dim c As Range
   Dim i As Integer

   With ThisWorkbook.Worksheets(3)
      .Unprotect

      For Each c In .Range("B1:B4")
         i = i + 1
         .Cells(i, 1) = c.Address
         .Cells(i, 2) = c.Address(True, False)
         .Cells(i, 3) = c.Address(False, True)
         .Cells(i, 4) = c.Address(False, False)
      Next c
   End With
End Sub
```


Microsoft Excel Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-460-6

Do While...Loop benutzen

Eine `Do...Loop`-Schleife wird so lange ausgeführt, bis ein bestimmter Zustand eintrifft. Es gibt zwei Arten von `Do...Loop`-Schleifen. Die eine verwendet das Schlüsselwort `While` (solange) und die andere `Until` (bis). Sie finden nachfolgend je ein Beispiel. Jede `Do...Loop`-Schleife wird durch eine `Do`-Zeile begonnen und durch eine `Loop`-Zeile abgeschlossen.

Im ersten Beispiel wird mit `While` gearbeitet. Die Zellen der Spalte *A* in *Tabelle2* sollen so lange durchlaufen werden, bis die erste leere Zelle gefunden wird. Dazu wird der Zähler `i` verwendet, dem vor Eintritt in die Schleife der Wert 1 zugewiesen wird. Innerhalb der Schleife wird der Zähler jeweils um den Wert 1 erhöht. Beachten Sie den Operator Ungleich (<>), den wir in der ersten Zeile der Schleife einsetzen. Er veranlasst, dass die Schleife so lange mit `While` durchlaufen wird, bis eine leere ("") Zelle gefunden wird. Am Ende der Prozedur wird die erste freie Zelladresse der Spalte *A* ausgegeben.

Listing 7.9 Erste freie Zelle ermitteln

```
Sub DoWhileLoop()
   Dim i As Integer

   i = 1

   Do While ThisWorkbook.Worksheets(2).Range("A" & i) <> ""
      i = i + 1
   Loop

   MsgBox Range("A" & i).Address
End Sub
```

Abbildg. 7.8 Die Ausgabe der Zelladresse

Do Until...Loop kennen lernen

Im unserem zweiten Beispiel in Bezug auf `Do Loop` verwenden wir `Until`. Das Ergebnis dieser Schleife ist dasselbe wie zuvor. Der Unterschied neben der Anweisung `Until` besteht im Operator. Diesmal wird ein Gleichheitszeichen (=) verwendet. Die Schleife wird so lange ausgeführt, bis die erste leere Zeile in der Spalte *A* gefunden wird.

Listing 7.10 Eine andere Methode zum Ermitteln der ersten freien Zelle

```
Sub DoUntilLoop()
   Dim i As Integer

   i = 1
```

Listing 7.10 Eine andere Methode zum Ermitteln der ersten freien Zelle *(Fortsetzung)*

```
   Do Until ThisWorkbook.Worksheets(2).Range("A" & i) = ""
      i = i + 1
   Loop

   MsgBox Range("A" & i).Address
End Sub
```

While...Wend verwenden

Die Verwendung der `While...Wend`-Schleife ist der `Do...Loop`-Schleife sehr ähnlich. Die Schleife wird ebenfalls so lange ausgeführt, bis der gewünschte Zustand eintrifft. Um dies zu verdeutlichen, ist das nachfolgende Codebeispiel analog der vorangegangenen Beispiele aufgebaut. Diese Schleifenart wird immer durch `While` eingeleitet und durch `Wend` abgeschlossen.

Listing 7.11 *While...Wend* zum Ermitteln der ersten freien Zelle

```
Sub WhileWend()
   Dim i As Integer

   i = 1

   While ThisWorkbook.Worksheets(2).Range("A" & i) <> ""
      i = i + 1
   Wend

   MsgBox Range("A" & i).Address
End Sub
```

Codezeilen zusammenfassen

Eine `With`-Anweisung haben Sie bereits in Kapitel 1 im Zusammenhang mit der Bereinigung von Makrorekorder-Code kennen gelernt. `With`-Anweisungen sind dazu da, mehrere Anweisungen für ein bestimmtes Objekt zusammenzufassen. Neben der Tatsache, dass Sie sich etwas Schreibarbeit ersparen, wird damit die Geschwindigkeit von Prozeduren erhöht. Der Code kann auf diese Weise zudem übersichtlicher gestaltet werden.

Eine `With`-Anweisung wird jeweils durch das Schlüsselwort `With` eingeleitet. Danach folgt die Angabe des Objekts. Abgeschlossen wird eine `With`-Anweisung durch eine `End With`-Zeile. Zwischen diesen beiden Zeilen werden die gewünschten Angaben zum Objekt zusammengefasst. Eine `With`-Anweisung macht nur Sinn, wenn darin mehrere Angaben zum Objekt erfolgen. Wenn das Objekt nur einmal angesprochen wird, kann auf die `With`-Anweisung verzichtet werden.

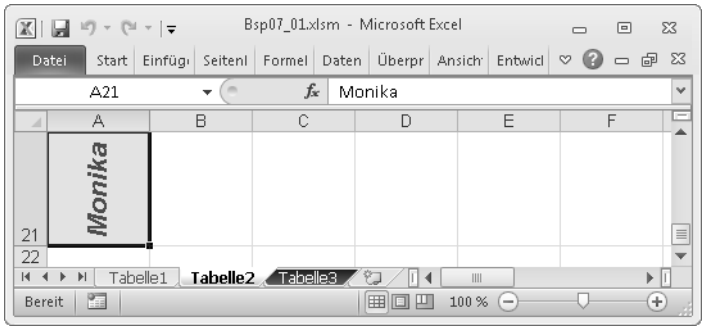
BEISPIEL Die Beispiele zum Thema *Zusammenfassungen* befinden sich innerhalb der Begleitdateien zum Buch im Ordner *\Buch\Kap07*. Die Mappe nennt sich *Bsp07_01.xlsm*. Die Prozeduren befinden sich in *Modul2*.

In der folgenden Prozedur werden der definierten Zelle verschiedene Formatierungen zugewiesen. Alle Anweisungen betreffen `ThisWorkbook.Worksheets(2).Range("A20")` und können somit zusammengefasst werden.

Listing 7.12 Verwendung von *With*

```
Sub WithVerwenden()
   With ThisWorkbook.Worksheets(2).Range("A20")
      .Interior.Color = vbYellow      ' Gelber Hintergrund
      .Value = "Monika"               ' Zellinhalt
      .HorizontalAlignment = xlCenter ' Horizontal zentriert
      .VerticalAlignment = xlCenter   ' Vertikal zentriert
      .Orientation = 90               ' 90° Drehung
      .Font.Bold = True               ' Fett
      .Font.Italic = True             ' Kursiv
      .Font.Size = 16                 ' 16 Punkt Schriftgröße
      .Font.Color = vbRed             ' Rote Textfarbe
   End With
End Sub
```

Abbildg. 7.9 Die formatierte Ausgabe erfolgt an entsprechender Stelle auf dem gewünschten Tabellenblatt



`With`-Anweisungen können verschachtelt werden. Bei der Betrachtung des vorangegangenen Beispiels fällt auf, dass das Objekt `Font` mehrmals verwendet wird. Das bedeutet, dass auch dieses Objekt in einer weiteren `With`-Anweisung zusammengefasst werden kann. Das Ergebnis der verschachtelten Zusammenfassung sieht wie folgt aus:

Listing 7.13 Verschachtelte *With*-Anweisungen

```
Sub VerschachteltesWith()
   With ThisWorkbook.Worksheets(2).Range("A21")
      .Interior.Color = vbYellow      ' Gelber Hintergrund
      .Value = "Monika"               ' Zellinhalt
      .HorizontalAlignment = xlCenter ' Horizontal zentriert
      .VerticalAlignment = xlCenter   ' Vertikal zentriert
      .Orientation = 90               ' 90° Drehung

      With .Font
         .Bold = True               ' Fett
         .Italic = True             ' Kursiv
         .Size = 16                 ' 16 Punkt Schriftgröße
         .Color = vbRed             ' Rote Textfarbe
      End With

   End With
End Sub
```

Geschwindigkeit von Prozeduren messen

Das Thema Geschwindigkeit wurde in diesem Buch bereits mehrmals angesprochen und ist sehr wichtig in der VBA-Programmierung. Viele Codes lassen sich auf verschiedene Weise programmieren. Die Meinungen über den jeweiligen Programmierstil gehen für gewöhnlich auseinander. In einem sind sich die Programmierer jedoch einig: Die Prozedur soll möglichst schnell sein. Die Geschwindigkeit einer Prozedur lässt sich, unter zu Hilfenahme einer API-Funktion (Application Programming Interface, Anwendungsprogrammierschnittstelle), messen.

HINWEIS Näheres zum Thema API erfahren Sie in Kapitel 24.

BEISPIEL Die nachfolgenden Beispiele befinden sich innerhalb der Begleitdateien zum Buch im Ordner *\Buch\Kap07*. Die Mappe nennt sich *Bsp07_02.xlsm*.

Bevor wir einen Vergleich vornehmen, werden wir uns anhand eines Codefragments den Kern der API-Funktion ansehen. Direkt nach `Option Explicit`, also über allen Prozeduren, wird die API-Deklaration `Private Declare Function` vorgenommen. Jede Prozedur in dem Modul kann somit auf die Funktion zugreifen. Die Funktion `GetTickCount` gibt die Systemzeit in Millisekunden zurück. Mittels `Lib "kernel32"` kann auf die benötigte DLL (Dynamic Link Library) zugegriffen werden. Innerhalb der Prozedur verwenden wir die Variable `lngStart`. Dieser übergeben wir die aktuelle Systemzeit, um den Startwert festzulegen. Danach folgt die Prozedur, die gemessen werden soll (nächstes Codebeispiel). Am Ende der Prozedur geben wir die Differenz zwischen der aktuellen Uhrzeit `GetTickCount` und der Startzeit, die in der Variablen `lngStart` gespeichert wurde, im Direktfenster aus. Der Wert wird in Millisekunden angezeigt. Um den Wert zusätzlich in Sekunden anzuzeigen, wird die Berechnung durch den Faktor 1.000 dividiert.

Listing 7.14 Zeit messen (Codefragment)

```
Option Explicit
Private Declare Function GetTickCount Lib "kernel32" () As Long

Sub MyDuration()
   Dim lngStart As Long
   lngStart = GetTickCount

   ' [...]

   Debug.Print GetTickCount - lngStart & " Millisekunden"
   Debug.Print (GetTickCount - lngStart) / 1000 & " Sekunden"
End Sub
```

Nachfolgend finden Sie drei Prozeduren, die exakt dasselbe ausführen, jedoch einen unterschiedlichen Programmierstil verwenden. Die erste Prozedur sollte theoretisch die schnellste sein, denn sie verwendet eine `For Each`-Schleife, die bekanntlich für Objekte verwendet werden soll. Zudem fasst sie die Anweisungen für das `Range`-Objekt mit `With` zusammen. Es werden in der Prozedur 1.000 Zellen der Spalte *A* mit gelbem Hintergrund formatiert. Zudem wird den Zellen ein Text zugewiesen. Wenn Sie eine größere Zeitdifferenz zwischen den einzelnen Prozeduren erhalten möchten, ersetzen Sie den Wert 1.000 in allen Prozeduren durch eine beliebig höhere Zahl.

Microsoft Excel Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-460-6

HINWEIS Um ein optimales Testergebnis zu erhalten, sollten Sie darauf achten, dass im Hintergrund keine anderen Programme (z.B. Virenscanner) ausgeführt werden. Am besten schließen Sie sämtliche Applikationen bis auf die Excel-Datei, in der die Tests durchgeführt werden sollen. Das wiederholte Ausführen ein und derselben Prozedur kann dennoch zu unterschiedlichen Ergebnissen führen. Dies hängt davon ab, welche Prozesse im Hintergrund zusätzlich ausgeführt werden.

Listing 7.15 Erster Geschwindigkeitstest (optimaler Programmierstil)

```
Sub Test1()
   Dim lngStart As Long
   Dim c As Range

   lngStart = GetTickCount

      For Each c In Range("A1:A1000")
         With c
            .Interior.Color = vbYellow
            .Value = "Monika"
         End With
      Next c

   Debug.Print (GetTickCount - lngStart) / 1000 & " Sekunden"
End Sub
```

In der zweiten Prozedur wurde die `For Each`-Schleife durch eine `For...Next`-Schleife ausgetauscht. Auf eine `With`-Zusammenfassung wurde verzichtet. Theoretisch sollte diese Prozedur langsamer sein als die vorangegangene.

Listing 7.16 Zweiter Geschwindigkeitstest

```
Sub Test2()
   Dim lngStart As Long
   Dim i As Long

   lngStart = GetTickCount

      For i = 1 To 1000
         Cells(i, 1).Interior.Color = vbYellow
         Cells(i, 1).Value = "Monika"
      Next i

   Debug.Print (GetTickCount - lngStart) / 1000 & " Sekunden"
End Sub
```

Einen deutlichen Geschwindigkeitsverlust können Sie beim Ausführen der nachfolgenden Prozedur erfahren. Der Grund liegt darin, dass beim Durchlaufen der Schleife jeweils die Zelle selektiert wird.

Listing 7.17 Dritter Geschwindigkeitstest

```
Sub Test3()
   Dim lngStart As Long
   Dim i As Long
```

Listing 7.17 Dritter Geschwindigkeitstest *(Fortsetzung)*

```
   lngStart = GetTickCount

      For i = 1 To 1000
         Cells(i, 1).Select
         ActiveCell.Interior.Color = vbYellow
         ActiveCell.Value = "Monika"
      Next i

   Debug.Print (GetTickCount - lngStart) / 1000 & " Sekunden"
End Sub
```