PEARSON Education

jetzt lerne ich

Start ohne Vorwissen

VBA mit Excel

Arbeitsabläufe automatisieren

BERND HELD

Markt+Technik

aktuell zu Excel 2007

Kapitel 3

jetzt lerne ich

Zellen und Bereiche programmieren

Das wichtigste Objekt in Excel ist sicherlich die Zelle. Dort finden die wesentlichen Aktionen statt. In der Regel werden Zellen ausgewertet, gefüllt, formatiert und vieles mehr. In diesem Kapitel lernen Sie daher ganz konkret, wie diese Zellen angesprochen und bearbeitet werden. Da Sie spezielle Aufgaben nicht jeweils nur für eine Zelle, sondern auch für ganze Bereiche durchführen, ist das vorherige Kapitel mit den Schleifen die Voraussetzung für dieses Kapitel.

Das `Range`-Objekt repräsentiert eine Zelle oder gar mehrere Zellen. Des Weiteren kann ein `Range`-Objekt auch eine oder mehrere Zeilen bzw. Spalten darstellen.

3.1 Zellen und Bereiche markieren

Das Markieren von Zellen und Bereichen ist bei der Programmierung im Prinzip gar nicht notwendig, da Zellen auch per Makro bearbeitet werden können, ohne »daraufzusitzen«. Trotzdem ist es für bestimmte Aufgaben wichtig, Zellen- und Bereichsadressen auszulesen. Zellen bzw. Bereiche lassen sich mit der Methode `Select` markieren.

Tabelle 3.1: Die Markierungsformen im Überblick

Aktion	Befehl
Markierung einzelner Zellen	`Range("A1").Select` oder `Cells(1,1).Select`
Markierung eines Datenbereichs	`Range("A1:C10").Select`
Markierung nicht zusammenhängender Zellen	`Range("A1,A3,A5,A7").Select`
Markierung mehrerer Bereiche	`Range("A1:A10,C1:C10").Select`
Markieren aller Zellen einer Tabelle	`Cells.Select`

3.1.1 Die übersichtlichere Mehrfachauswahl

Sofern mehr als zwei Bereiche markiert werden sollen, wird die Schreibweise, wie in der Tabelle 3.1 beschrieben, ein wenig unübersichtlich. Hier hilft die Methode `Union` wie in dem Listing 3.1 beschrieben.

Listing 3.1: Mehrere Bereiche definieren und zusammenführen

```
Sub MehrereBereicheMarkieren()
Dim Ber1 As Range, Ber2 As Range
Dim Ber3 As Range, Ber4 As Range
Dim Bereiche As Range

    With Sheets("Tabelle1")

    Set Ber1 = .Range("A1:A5")
    Set Ber2 = .Range("C1:C5")
    Set Ber3 = .Range("A10:A15")
    Set Ber4 = .Range("C10:C15")
    Set Bereiche = Union(Ber1, Ber2, Ber3, Ber4)

    Bereiche.Select

    End With

End Sub
```

Mit der Anweisung `Set` definieren Sie zuerst die einzelnen Zellbereiche `Ber1`–`Ber4`. Danach werden diese Einzelbereiche mithilfe der Methode `Union` in einem Block vereint, der den Namen `Bereiche` trägt. Dieser Block lässt sich nun individuell ansteuern und wie hier hinterlegt über die Methode `Select` komplett markieren.

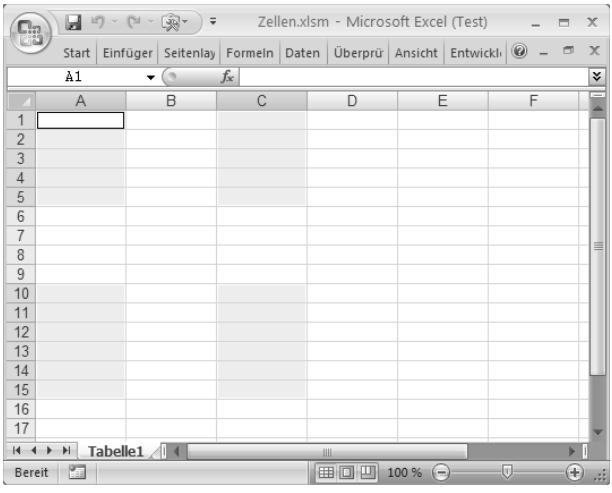


Abb. 3.1: Mehrere Bereiche auf einmal markieren

3.1.2 Die letzte benutzte Zelle einer Tabelle ansteuern

Wenn Sie mit Schleifen arbeiten und die Abbruchbedingung der Schleife beispielsweise von der letzten belegten Zelle auf dem Tabellenblatt abhängig machen möchten, dann wird das Makro in Listing 3.2 helfen. Dort wird die `SpecialCells`-Methode verwendet, der man die Konstante `xlCellTypeLastCell` mitgibt. Diese Methode liefert Ihnen die Zellenadresse der letzten Zelle im verwendeten Bereich.

Listing 3.2: Die letzte Zelle im benutzten Bereich einer Tabelle ansteuern

```
Sub LetzteZelleImBenutztenBereichErmitteln()

  With Sheets("Tabelle2")

  .Activate
  .Cells.SpecialCells(xlCellTypeLastCell).Select

  End With

  MsgBox "Letzte Zelle im benutzten Bereich: " & ActiveCell.Address


End Sub
```

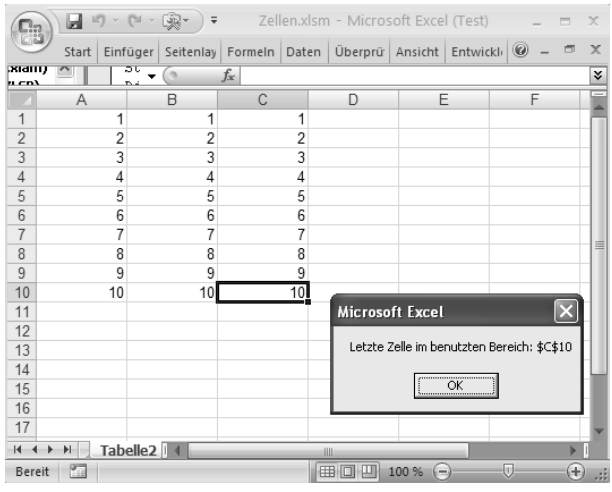


Abb. 3.2: Die letzte Zelle in der Tabelle wird markiert.

Weitere interessante Konstanten dieser Methode entnehmen Sie der Tabelle 3.2.

Tabelle 3.2: Die möglichen Konstanten der Methode SpecialCells

Konstante	Beschreibung
`xlCellTypeAllFormatConditions`	markiert alle formatierten Zellen
`xlCellTypeAllValidation`	Zellen mit Gültigkeitsregeln werden markiert
`xlCellTypeBlanks`	gibt alle leeren Zellen an
`xlCellTypeComments`	Zellen mit Kommentaren werden markiert
`xlCellTypeConstants`	Zellen mit Konstanten werden markiert
`xlCellTypeFormulas`	Zellen mit Formeln werden markiert
`xlCellTypeLastCell`	letzte Zelle im benutzten Bereich wird angesteuert
`xlCellTypeSameFormatConditions`	Zellen mit gleichem Format werden markiert
`xlCellTypeSameValidation`	Zellen mit gleichen Gültigkeitskriterien werden markiert
`xlCellTypeVisible`	alle sichtbaren Zellen werden markiert

3.1.3 Die letzte Zelle einer Spalte markieren

Ähnlich wie beim vorherigen Makro wird nun wiederum eine letzte Zelle angesteuert. Im folgenden Beispiel aus Listing 3.3 wird die letzte Zelle aus Spalte B markiert.

Listing 3.3: Die letzte Zelle einer Spalte markieren

```
Sub LetzteZelleAusSpalteErmitteln()

With Sheets("Tabelle2")

  .Activate
  .Range("B1048576").End(xlUp).Select

  End With
  MsgBox "Letzte Zelle in Spalte B: " & ActiveCell.Address

End Sub
```

Über die Eigenschaft `End` gelangen Sie an das Ende einer Spalte. Dazu übergeben Sie der Eigenschaft die Konstante `xlUp`. Intern führt Excel dann zunächst einen Sprung ans Ende der Spalte, also in die Zelle B1048576, durch und springt danach nach oben, bis zur letzten gefüllten Zelle der Spalte.

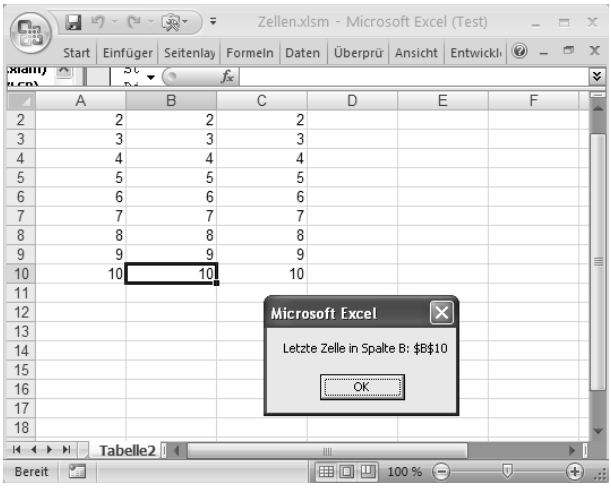


Abb. 3.3: Die letzte Zelle aus Spalte B wird ermittelt.

3.1.4 Die letzte Zelle einer Zeile markieren

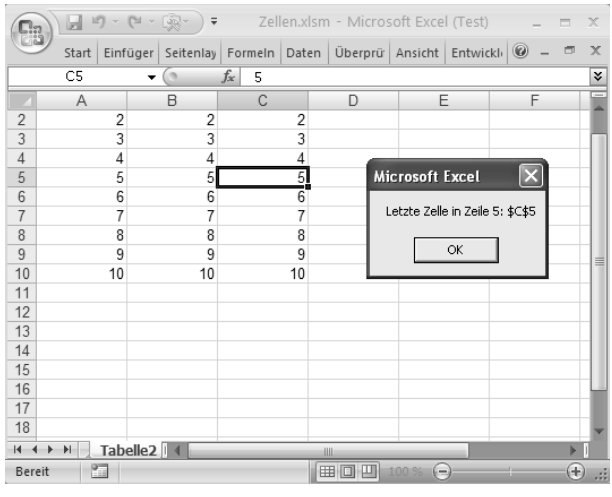
Analog zum vorherigen Beispiel wird im Makro aus Listing 3.4 die letzte Zelle der Zeile 5 angesteuert.

Listing 3.4: Die letzte Zelle einer Zeile wird ermittelt.

```
Sub LetzteZelleAusZeileErmitteln()

With Sheets("Tabelle2")

  .Activate
  .Range("B5").End(xlToLeft).Select

  End With
  MsgBox "Letzte Zelle in Zeile 5: " & ActiveCell.Address

End Sub
```

Über die Eigenschaft `End` gelangen Sie an das Ende einer Zeile. Dazu übergeben Sie der Eigenschaft die Konstante `xlToLeft`. Intern führt Excel dann zunächst einen Sprung ans Ende der Zeile, also in die Zelle IV5, durch und springt danach nach links, bis zur letzten gefüllten Zelle der Zeile.

Abb. 3.4: Die letzte Zelle aus Zeile 5 wird ermittelt.



3.1.5 Alle Formelzellen markieren

Mithilfe der Methode `SpecialCells` lassen sich in einer Tabelle auch blitzschnell alle Zellen mit Formeln markieren. Das Makro für diese Aufgabe können Sie in Listing 3.5 nachsehen.

Listing 3.5: Alle Zellen mit Formeln werden markiert.

```
Sub FormelzellenMarkieren()

On Error GoTo fehler

  With Sheets("Tabelle3")

    .Activate
    .Cells.SpecialCells(xlCellTypeFormulas).Select

  End With
  Exit Sub


fehler:
   MsgBox Err.Description, vbCritical

End Sub
```

Übergeben Sie der Methode `SpecialCells` die Konstante `xlCellTypeFormulas`, um alle Formelzellen der Tabelle zu ermitteln. Mit der Methode `Select` werden diese Zellen anschließend markiert.

Sollte die Tabelle überhaupt keine Formeln enthalten, würde das Makro ohne eine Fehlerbehandlung abstürzen. Daher verzweigen Sie im Fehlerfall über `On Error GoTo` direkt in den Paragrafen `fehler`. Dort wird auf das Objekt `Err` zurückgegriffen. In diesem Objekt werden im Fehlerfall automatisch alle Informationen über den Fehler zur Verfügung gestellt. Über die Eigenschaft `Description` lässt sich der genaue Wortlaut der Fehlermeldung darstellen.

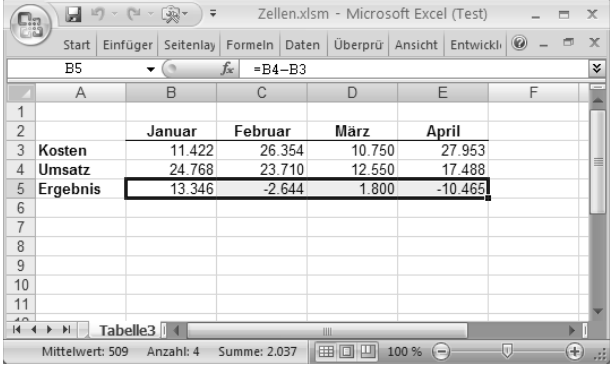


Abb. 3.5: Alle Formelzellen werden markiert.

3.1.6 Die umliegenden Zellen markieren

Über die Eigenschaft `UsedRange` lässt sich der verwendete Bereich Ihrer Tabelle ermitteln. Damit haben Sie aber jeweils den kompletten verwendeten Bereich – möglicherweise möchten Sie aber nur einen Teilbereich davon adressieren. Mithilfe der Eigenschaft `CurrentRegion` lassen sich beispielsweise der umliegende Bereich einer Zelle ermitteln. Der ermittelte Bereich wird durch die erste Leerzeile bzw. Leerspalte begrenzt.

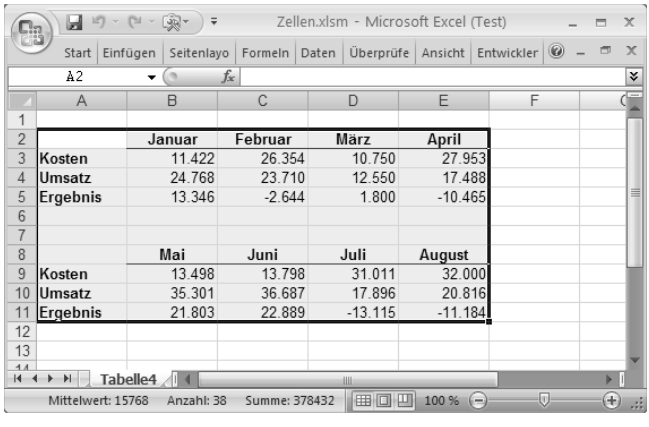
Sehen Sie den Unterschied beider Eigenschaften im folgenden Beispiel aus den Makros der Listings 3.6 und 3.7.

Listing 3.6: Der benutzte Bereich einer Tabelle wird markiert.

```
Sub BenutztenBereichMarkieren()

    Sheets("Tabelle4").UsedRange.Select

End Sub
```

Die Eigenschaft `UsedRange` repräsentiert den verwendeten Bereich in der angegebenen Tabelle.

Abb. 3.6: Der benutzte Bereich der Tabelle wird komplett markiert.



	A	B	C	D	E
1					
2		Januar	Februar	März	April
3	Kosten	11.422	26.354	10.750	27.953
4	Umsatz	24.768	23.710	12.550	17.488
5	Ergebnis	13.346	-2.644	1.800	-10.465
6					
7					
8		Mai	Juni	Juli	August
9	Kosten	13.498	13.798	31.011	32.000
10	Umsatz	35.301	36.687	17.896	20.816
11	Ergebnis	21.803	22.889	-13.115	-11.184
12					
13					

Listing 3.7: Der umliegende Bereich einer Zelle wird markiert.

```
Sub UmliegendenBereichMarkieren()

    Sheets("Tabelle4").Range("A2").CurrentRegion.Select

End Sub
```

Mithilfe der Eigenschaft `CurrentRegion` wird, ausgehend von der Zelle A2, der umliegende Bereich dieser Zelle markiert und über die Methode `Select` ausgewählt.

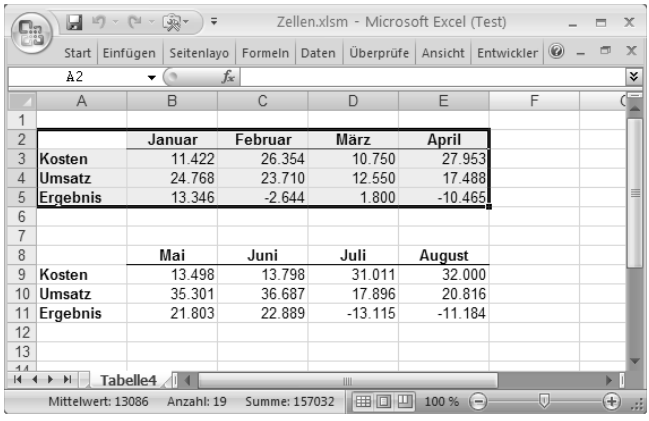


Abb. 3.7: Den umliegenden Bereich um Zelle A2 markieren

3.2 Zellen und Bereiche formatieren

Im folgenden Abschnitt lernen Sie einige Möglichkeiten kennen, wie Zellen und Bereiche formatiert werden können. Dabei reicht die Palette von Hintergrundfarben, Schriftfarben sowie Schriftschnitten bis hin zu Rahmen und Schattierungen.

3.2.1 Alle verbundenen Zellen kennzeichnen

Nicht immer sind verbundene Zellen in einer Tabelle auf den ersten Anblick sichtbar. Sollen alle verbundenen Zellen einer Tabelle optisch hervorgehoben werden, dann übernimmt das Makro aus Listing 3.8 diese Aufgabe.

Listing 3.8: Verbundene Zellen einfärben

```
Sub VerbundeneZellenFormatieren()
Dim Zelle As Range

    For Each Zelle In Worksheets("Tabelle5").UsedRange

        If Zelle.MergeCells = True Then
             Zelle.Interior.ColorIndex = 4
        Else
             Zelle.Interior.ColorIndex = xlColorIndexNone
        End If

    Next Zelle

End Sub
```

Über eine `For Each...Next`-Schleife wird in der Tabelle5 jede einzelne Zelle angesprochen, die aktuell verwendet wird. Hierzu wird die Eigenschaft

UsedRange eingesetzt. In einer If-Abfrage erfolgt die Ermittlung, ob diese Zelle einem Zellenverbund angehört. Wenn ja, dann gibt die Eigenschaft MergeCells den Wert True zurück und die Zelle wird mit der Farbe GRÜN gefärbt.

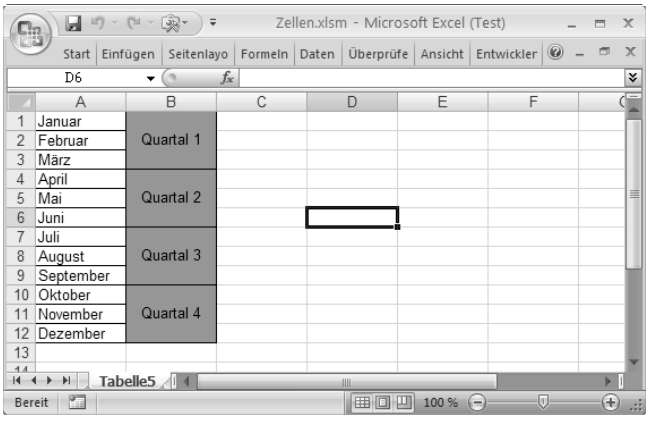


Abb. 3.8: Die verbundenen Zellen aus Spalte B wurden eingefärbt.

3.2.2 Die Schrift anpassen

In der nächsten Aufgabe sollen innerhalb einer Markierung alle Zellen mit der Schriftart COURIER und dem Schriftgrad 12 formatiert. Das Makro für diese Aufgabe können Sie in Listing 3.9 sehen. Markieren Sie vor dem Start des Makros ein paar beliebige Zellen, die bereits Texte oder Zahlen enthalten.

Listing 3.9: Alle markierten Zellen formatieren

```
Sub SchriftenFormatieren()
Dim Zelle As Range

 For Each Zelle In Selection

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

Next Zelle

End Sub
```

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

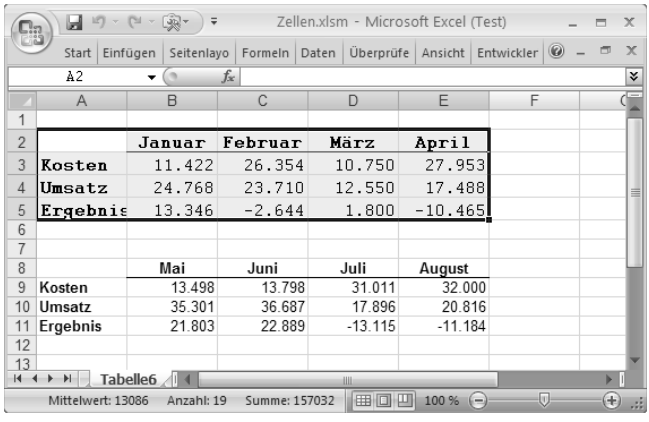


Abb. 3.9: Alle markierten Zellen wurden umformatiert.

Entnehmen Sie weitere wichtige Eigenschaften des Objekts `Font` aus der folgenden Tabelle.

Tabelle 3.3: Die wichtigsten Eigenschaften für die Schriftgestaltung

Eigenschaft	Beschreibung
`Bold`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text fett formatiert ist.
`ColorIndex`	Diese Eigenschaft gibt die Farbe des Rahmens, der Schriftart oder des Innenraums zurück. Es existieren in Excel genau 56 Farben.
`FontStyle`	Diese Eigenschaft sagt aus, welcher Schriftschnitt verwendet wird. Möglich sind u.a. Fett- und Kursivdruck.
`Italic`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text kursiv formatiert ist.
`OutLineFont`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text als Konturschriftart formatiert wird.
`Shadow`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text als schattierte Schriftart formatiert wird.
`Strikethrough`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text als horizontal durchgestrichen dargestellt wird.
`Subscript`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text tiefergestellt formatiert wird.

Tabelle 3.3: Die wichtigsten Eigenschaften für die Schriftgestaltung (Forts.)

Eigenschaft	Beschreibung
`Superscript`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text hochgestellt formatiert wird.
`Underline`	Diese Eigenschaft liefert den Wert `True`, wenn ein Text unterstrichen formatiert wird. Dabei können Sie u.a. Text einfach oder doppelt unterstreichen.

3.2.3 Extremwerte kennzeichnen

Im nächsten Beispiel aus Listing 3.10 sollen in einem Bereich alle Werte über 100 hervorgehoben werden. Diese Aufgabe lässt sich elegant über eine Schleife lösen, die alle Zellen des Bereichs kontrolliert und die »Ausreißer« farblich hervorhebt.

Listing 3.10: Alle Werte größer oder gleich 100 werden optisch hervorgehoben.

```
Sub ExtremwerteHervorheben()
Dim Zelle As Range
Dim Bereich As Range

Set Bereich = Sheets("Tabelle7").Range("A1:E12")

For Each Zelle In Bereich

 If Zelle.Value >= 100 Then
    Zelle.Interior.ColorIndex = 4
 Else
    Zelle.Interior.ColorIndex = xlColorIndexNone
 End If

Next Zelle

End Sub
```

Deklarieren Sie zunächst einmal zwei Objektvariablen vom Typ `Range`. Danach wird der Bereich, der abgearbeitet werden soll, mithilfe der Anweisung `Set` bekannt gegeben. Anschließend werden mit einer `For Each...Next`-Schleife alle Zellen innerhalb dieses Bereichs durchlaufen. In der Schleife selbst wird der Wert der Zellen überprüft. Liegt der Wert dabei über 100, dann wird der Hintergrund dieser Zelle über die Eigenschaft `ColorIndex` farblich angepasst.

Vergessen Sie bei der `If`-Anweisung nicht den `Else`-Zweig, der eingesetzt wird, um Werte außerhalb der »Wertgrenze« wieder farblich zurückzusetzen. Es kann schließlich vorkommen, dass sich Werte im vorgegebenen Bereich geändert haben. Beispielsweise könnte in eine Zelle mit einem Wert über 100 ein Wert kleiner als 100 eingetragen werden. Ohne den Einsatz des `Else`-Zweiges würde der Farbindex dieser Zelle nicht zurückgesetzt werden und die Farbe bliebe bestehen.

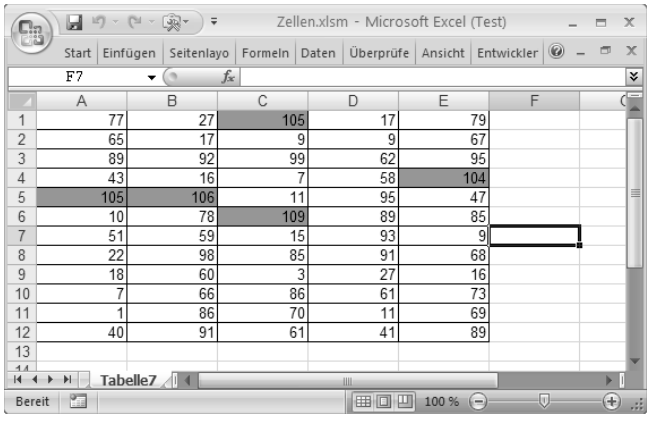


Abb. 3.10: Extremwerte über 100 werden mit der Hintergrundfarbe Grün hinterlegt.

3.2.4 Bereiche einrahmen

In dem folgenden Beispiel aus Listing 3.11 werden die beiden Bereiche A2:E5 sowie A8:E11 mit einem Rahmen versehen.

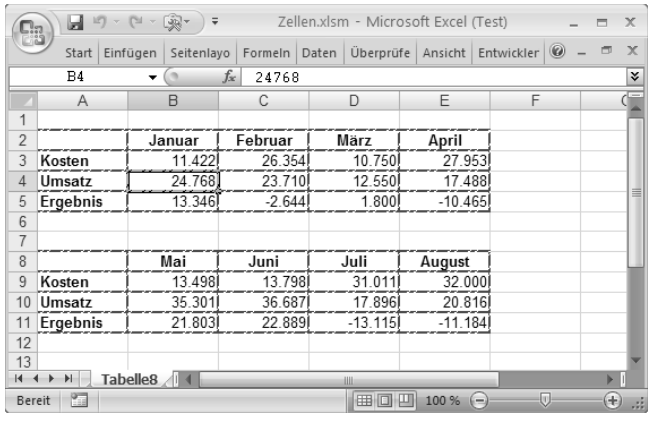
Listing 3.11: Mehrere Bereiche werden eingerahmt.

```
Sub RahmenFestlegen()
Dim Bereich1 As Range
Dim Bereich2 As Range
Dim Gesamt As Range

With Sheets("Tabelle8")
  Set Bereich1 = .Range("A2:E5")
  Set Bereich2 = .Range("A8:E11")
  Set Gesamt = Union(Bereich1, Bereich2)

   With Gesamt.Borders
       .ColorIndex = 3
       .LineStyle = xlSlantDashDot
   End With

End With

End Sub
```

Zu Beginn werden drei Objektvariablen vom Typ `Range` deklariert. Danach werden die zu umrahmenden Bereiche den ersten beiden Objektvariablen `Bereich1` und `Bereich2` mithilfe der Anweisung `Set` zugewiesen. Mit der Methode `Union` werden die beiden einzelnen Bereiche zu einem Gesamtbereich zusammengefasst und können nun unter dem Namen `Gesamt` als ein Bereich angesprochen werden.

Mit dem Objekt `Borders` wird nun die Berahmung des Bereichs festgelegt. Dieses Objekt bietet Ihnen hierfür vielfältige Eigenschaften an. Unter anderem lässt sich über die Eigenschaft `ColorIndex` die Rahmenfarbe festlegen. Die Eigenschaft `LineStyle` definiert die Form des Rahmens.

Abb. 3.11: Eine individuelle Rahmung von Bereichen durchführen



In der nächsten Tabelle sehen Sie weitere Möglichkeiten, um die Form eines Rahmens zu definieren.

Tabelle 3.4: Die weiteren Möglichkeiten der Rahmenformen

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

3.3 Zelleninhalte manipulieren

In diesem Abschnitt werden Zelleninhalte verändert. Dazu werden eine ganze Reihe praktischer Aufgaben vorgestellt.

3.3.1 Daten nach Datentransfer bereinigen

Wenn Daten von anderen Systemen wie Großrechnern oder Datenbankservern eingelesen werden, haben Sie vielleicht schon einmal festgestellt, dass unerwünschte Zeichen mit übertragen wurden. Diese Zeichen nennt man »nicht druckbare Zeichen«. Auch kann es vorkommen, dass Excel Zahlenwerte nicht richtig erkennt und diese stattdessen als Text konvertiert. Auf den ersten Blick fällt das nicht immer gleich auf. Erst wenn man versucht, die Daten weiterzuverarbeiten, zum Beispiel grafisch oder in Kalkulationen, entstehen Probleme.

Im folgenden Beispiel aus Listing 3.12 werden alle Zellen einer Markierung noch einmal geprüft und – wo notwendig – »geputzt«. Dabei werden Formelzellen von der Bereinigung ausgenommen.

Listing 3.12: Alle Zellen im benutzten Bereich werden gesäubert.

```
Sub ZellenBereinigen()
Dim Zelle As Range

For Each Zelle In ActiveSheet.UsedRange

    With Zelle

      If .HasFormula = False Then
         .Value = Application.WorksheetFunction.Clean(.Value)
      End If

    End With

Next Zelle

End Sub
```

Im benutzten Bereich der Tabelle wird die Säuberungsaktion nur bei Zellen durchgeführt, die keine Verknüpfungen und keine Formeln enthalten. Für diese Abfrage wird die Eigenschaft `HasFormula` verwendet. Gibt diese Eigenschaft den Wert `True` zurück, handelt es sich um eine Zelle, die eine Formel oder Verknüpfung enthält, im anderen Fall gibt die Eigenschaft `HasFormula` den Wert `False` zurück. Danach entfernt die Funktion `Clean` alle nicht druckbaren Zeichen.

3.3.2 Excel vorschreiben richtig zu rechnen

Im nächsten Beispiel wird Excel genötigt, importierte Zahlenwerte auch als solche zu erkennen. Am schnellsten kann geprüft werden, ob die Daten richtig nach Excel übertragen wurden, indem man versucht, eine Summe über die Zahlenwerte zu bilden. Meldet die Funktion den Wert 0, kann man davon ausgehen, dass Excel die Zahlenwerte als Text interpretiert. Üblicherweise wird man im nächsten Schritt wahrscheinlich probieren, den Zellen noch einmal ein Zahlenformat über den Dialog Zellen Formatieren zuzuweisen. Aber auch diese Aktion bringt nicht den gewünschten Erfolg. Wenn es sich nur um ein paar Werte handelt, die falsch wiedergegeben werden, lassen sich diese durch Markierung der Zelle und dann per Druck auf die Taste F2 und anschließend auf die ↵-Taste manuell umsetzen. Damit wird Excel »gezwungen«, neu zu rechnen. Bei größeren Datenmengen ist diese Vorgehensweise natürlich recht langwierig und nervig. Sie müssen daher eine Lösung finden, die Excel automatisch veranlasst, die Daten zu korrigieren und dann neu zu rechnen. Dazu lässt sich zum Beispiel jeder Zellenwert mit der Zahl 1 multiplizieren. Damit wird der eigentliche Wert der Zelle nicht verändert und Sie gelangen zum gewünschten Ergebnis. Das Makro für diese Aufgabe ist im Listing 3.13 dargestellt.

Listing 3.13: Excel dazu bewegen, importierte Zahlenwerte auch richtig zu interpretieren

```
Sub ZahlenwerteRichtigErkennen()
Dim Zelle As Range

For Each Zelle In Selection

  If IsNumeric(Zelle.Value) = True And _
     Zelle.HasFormula = False Then

     Zelle.Value = Zelle.Value * 1

  End If

Next Zelle

End Sub
```

Zu Beginn wird eine Objektvariable vom Typ `Range` definiert. Danach werden mithilfe einer `For Each...Next`-Schleife alle markierten Zellen angesteuert. Innerhalb der Schleife erfolgt eine Prüfung, ob überhaupt ein numerischer Wert (`IsNumeric`) vorliegt oder ob sich in der jeweiligen Zelle eine Formel (`HasFormula`) befindet. Nur wenn die erste Prüfung `True` zurückgibt und die zweite den Wert `False`, wird eine Kalkulation durchgeführt und der Inhalt der jeweiligen Zelle neu beschrieben, indem der alte Wert mit der Zahl 1 multipliziert wird.

3.3.3 Bestimmte Zeichen aus Zellen entfernen

Beim Eliminieren bestimmter Zeichen aus Zellen, was in der Fachsprache als »parsen« bezeichnet wird, lässt sich mit dem folgenden Makro viel mühevolle Handarbeit ersparen.

Im folgenden Beispiel aus Listing 3.14 werden alle Nullen, Minuszeichen und Punkte aus allen markierten Zellen entfernt. Sehen Sie sich jedoch zunächst die Abbildung 3.12 näher an.

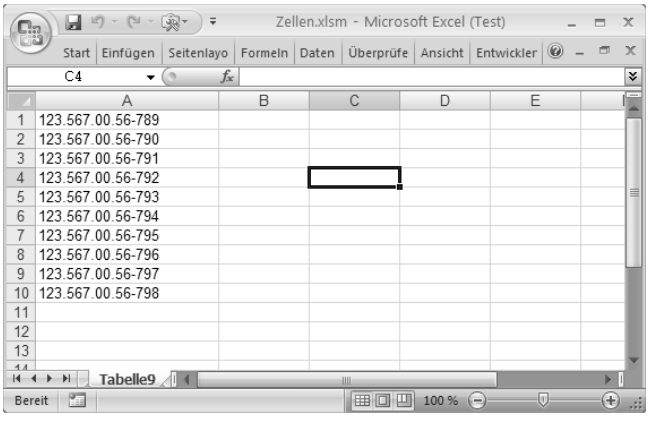


Abb. 3.12: Die Ausgangsliste mit diversen Sonderzeichen

Starten Sie jetzt das folgende Makro, um die gerade genannten Zeichen aus den Zellen zu entfernen.

```
Sub BestimmteZeichenEliminieren()
Dim Zelle As Range
Dim intz As Integer
Dim StrNeu As String

For Each Zelle In Selection

  For intz = 1 To Len(Zelle)

    Select Case Mid(Zelle, intz, 1)
     Case 0, ".", "-"

     Case Else
       StrNeu = StrNeu & Mid(Zelle, intz, 1)

    End Select

  Next intz
```

Listing 3.14: Definierte Zeichen werden aus allen markierten Zellen entfernt.

```
    Zelle.Value = StrNeu
    StrNeu = ""

Next Zelle

End Sub
```

In einer `For Each...Next`-Schleife werden alle markierten Zellen abgearbeitet. In der zweiten »verschachtelten« Schleife wird eine Zelle mithilfe der Funktion `Mid` Zeichen für Zeichen zerlegt. Diese Funktion hat drei Argumente. Im ersten Argument wird die Zelle angegeben, die zerlegt werden soll. Das zweite Argument gibt die jeweilige Startposition in dieser Zelle an. Das dritte Argument gibt die Länge an, d.h., wie viele Zeichen ausgewertet werden sollen. Da Zeichen für Zeichen ausgewertet wird, ist die dort zu verwendende Zahl die 1. Nun wird das jeweilige Zeichen über eine `Select Case`-Anweisung ausgewertet. Sofern die Zeichen nicht den im ersten `Case`-Zweig angegebenen Zeichen entsprechen, werden diese einzeln in die Hilfsvariable `strNeu` übertragen. Wurden alle Zeichen einer Zelle abgearbeitet, wird diese mit dem neuen bereinigten Wert aus der Variablen `strNeu` überschrieben.

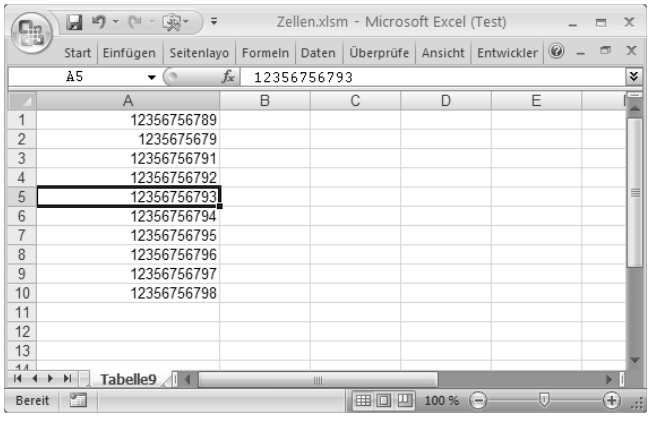


Abb. 3.13: Alle definierten Zeichen wurden entfernt.

Nach dem gleichen Strickmuster lassen sich beispielsweise auch numerische von alphanumerischen Zeichen trennen, wie im Makro aus Listing 3.15 dargestellt.

Listing 3.15: Alle alphanumerischen Zeichen aus den markierten Zellen entfernen

```
Sub AlphaNumerischeZeichenEntfernen()
Dim Zelle As Range
Dim intz As Integer
Dim StrNeu As String

For Each Zelle In Selection

  For intz = 1 To Len(Zelle)

    Select Case Mid(Zelle, intz, 1)

     Case 0 To 9
        StrNeu = StrNeu & Mid(Zelle, intz, 1)

     Case Else

    End Select

  Next intz

  Zelle.Offset(0, 1).Value = StrNeu
  StrNeu = ""

Next Zelle

End Sub
```

Das Makro aus Listing 3.15 entspricht weitestgehend dem Makro aus Listing 3.14. Jedoch wird der Zelleninhalt im ersten `Case`-Zweig nach den gültigen Zahlen 0 bis 9 untersucht. Entspricht das jeweilige Zeichen einer dort festgelegten Zahl, dann wird diese in der Hilfsvariablen `strNeu` Zeichen für Zeichen zusammengesetzt.

Die Ausgabe des Ergebnisses erfolgt in diesem Beispiel nicht in der gleichen Zelle, sondern in der Zelle rechts neben der markierten Zelle. Um diesen »Versatz« hinzubekommen, wenden Sie die Eigenschaft `Offset` an, die zwei Argumente hat. Im ersten Argument lässt sich eine eventuelle Zeilenverschiebung angeben. Da Sie aber in der jeweiligen Zeile bleiben möchten, wird dort die Zahl 0 (= keine Verschiebung) angegeben. Im zweiten Argument wird die Spaltenverschiebung definiert. Da das Ergebnis genau eine Spalte weiter nach rechts dargestellt werden soll, trägt man in diesem Argument den Wert 1 ein.

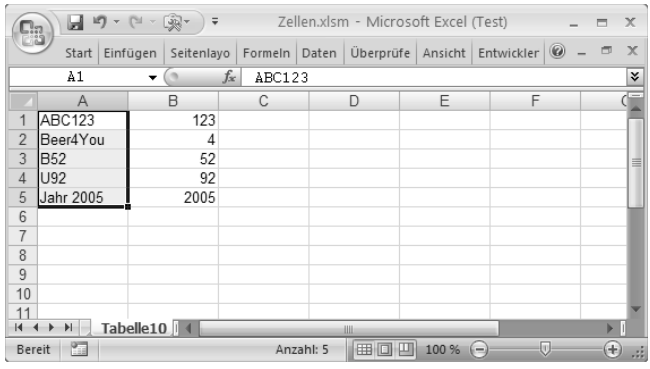


Abb. 3.14: In Spalte B finden Sie nur noch numerische Werte.

3.3.4 Namen drehen

Erhalten Sie Daten, die innerhalb der Zelle gedreht werden müssen (z.B.: Vorname Name → Name Vorname), dann hilft hierbei das Makro aus Listing 3.16:

Listing 3.16: Der Vor- sowie Nachname wurde gedreht.

```
Sub NamenDrehen()
Dim Zelle As Range

For Each Zelle In Selection

    Zelle.Offset(0, 1).Value = _
    Mid(Zelle.Value, InStr(Zelle.Value, " ") + 1)
    Zelle.Offset(0, 2).Value = _
    Left(Zelle.Value, InStr(Zelle.Value, " ") - 1)

Next Zelle

End Sub
```

Das Makro geht gegenüber der Aufgabenstellung sogar noch einen Schritt weiter, trennt Namen von Vornamen und schreibt diese in die Zellen nebenan. Hierbei wird die Funktion `Mid` eingesetzt, um den Nachnamen aus der Zelle in Spalte A zu extrahieren. Da zwischen Vor- und Nachnamen ein Leerzeichen liegt, lässt sich die exakte Position dieses Leerzeichens mithilfe der Funktion `Instr` ermitteln. Diese Position muss dann noch um den Wert 1 erhöht werden, da das Leerzeichen nicht übertragen werden soll.

Den Vornamen können Sie direkt über die Funktion `Left` aus der jeweiligen Zelle der Spalte A extrahieren. Der Vornamen muss bis zu einem Zeichen vor dem ermittelten Leerzeichen übertragen werden. Auch für diesen Zweck wird die Funktion `Instr` verwendet und von der gefundenen Position der Wert 1 subtrahiert.

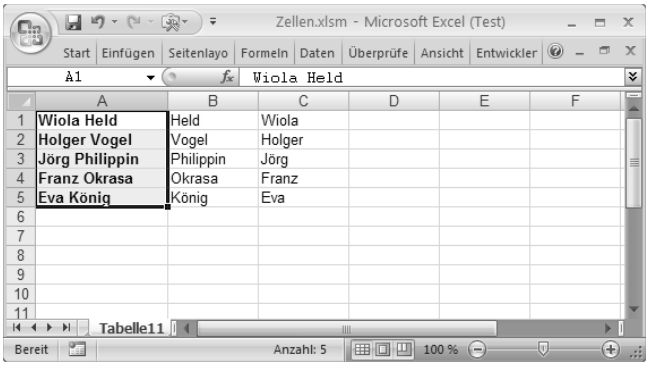


Abb. 3.15: Eine Zelle zerlegen und in anderer Form wieder zusammensetzen

3.3.5 Kommentare aus Zellentexten generieren

Gerade haben wir gelernt, wie Zelleninhalte überprüft werden und wie ein Teil davon übersichtlich in einer Liste dargestellt werden kann. Stellen Sie sich aber jetzt einmal folgendes Szenario vor: Sie arbeiten an einer Liste und möchten bestimmte Zellen mit neuen Texten überschreiben. Die alten Texte sind aber durchaus noch von Interesse und sollten nicht gänzlich aus der Liste verschwinden. Wie wäre es, wenn diese alten Einträge als Kommentar gesichert werden würden? Damit sind die alten Einträge sogar noch der ursprünglichen Zelle zugeordnet und die Inhalte können nun jederzeit verändert werden. Für diese Aufgabe erstellen Sie das folgende Makro in Listing 3.17:

Listing 3.17: Zelleninhalte als Kommentartext übernehmen

```
Sub KommentareAusZellenInhaltBilden()
Dim Kom As Comment
Dim Zelle As Range

For Each Zelle In Selection

  On Error Resume Next
    Set Kom = Zelle.AddComment
    Kom.Text Date & vbLf & Zelle.Value
    Kom.Visible = True

Next Zelle

End Sub
```

Das Makro in Listing 3.17 verschiebt mithilfe des Befehls `Text` den Inhalt der einzelnen Zellen innerhalb der Markierung jeweils in ein Kommentarfenster. Jeder Kommentar wird zusätzlich noch über die Funktion `Date` mit dem Datum ausgestattet. Damit die Kommentare angezeigt werden, setzen Sie die Eigenschaft `Visible` auf den Wert `True`.

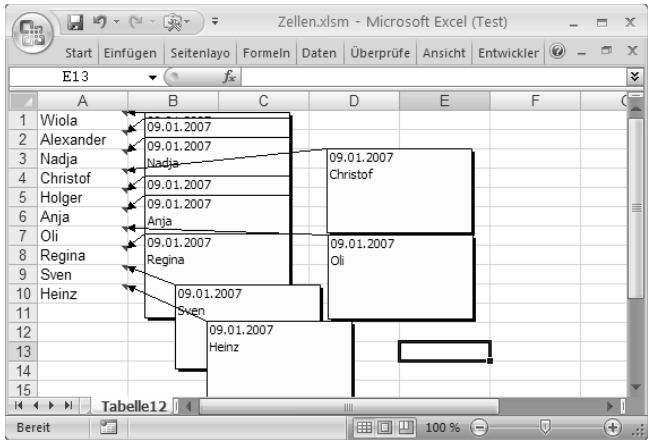


Abb. 3.16: Aus den Zelleninhalten wurden Kommentare mit dem aktuellen Datum gebildet.

3.3.6 Formeln durch Festwerte ersetzen

Wenn alle Formeln, Funktionen und externe Verknüpfungen in einem Bereich durch Festwerte ersetzt werden sollen, damit beispielsweise keine Änderungen mehr stattfinden können, übernimmt das Makro aus Listing 3.18 automatisch diese Aufgabe.

Listing 3.18: Alle Formelzellen werden durch Festwerte ersetzt.

```
Sub FormelnInTextUmsetzen()
Dim Zelle As Range

For Each Zelle In ActiveSheet.UsedRange

  If Zelle.HasFormula = True Then
     Zelle.Formula = Zelle.Value
  End If

Next Zelle

End Sub
```

Zur Lösung dieser Aufgabe durchläuft das Makro alle Zellen im benutzten Bereich mithilfe einer `For Each...Next`-Schleife. Innerhalb der Schleife erfolgt eine Prüfung mit der Funktion `HasFormula`, ob die jeweilige Zelle eine Formel enthält. Sofern dies zutrifft, wird der Zelle durch die Eigenschaft `Value` der Wert zugewiesen, den die Formel aktuell als Ergebnis darstellt.

3.3.7 Bestimmte Zelleninhalte löschen

Bei der folgenden Aufgabe sollen alle negativen Werte aus einem Bereich gelöscht werden. Sehen Sie sich hierzu die Abbildung 3.17 an.

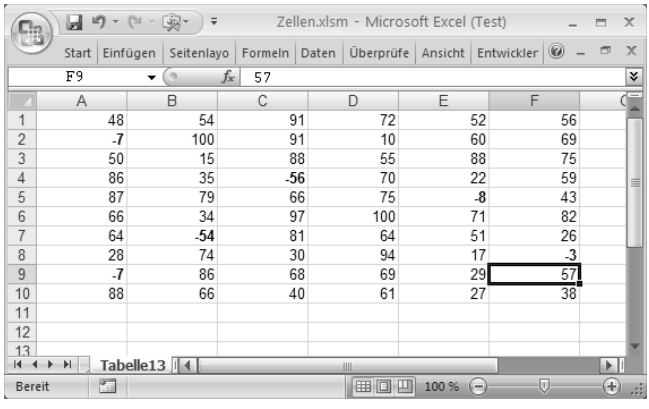


Abb. 3.17: Die Ausgangstabelle mit negativen sowie positiven Werten

	A	B	C	D	E	F
1	48	54	91	72	52	56
2	-7	100	91	10	60	69
3	50	15	88	55	88	75
4	86	35	-56	70	22	59
5	87	79	66	75	-8	43
6	66	34	97	100	71	82
7	64	-54	81	64	51	26
8	28	74	30	94	17	-3
9	-7	86	68	69	29	57
10	88	66	40	61	27	38
11						
12						
13						

Für den besseren Überblick wurden die Zellen, die negative Werte enthalten, vorab mit dem Schriftschnitt Fett formatiert. Die Aufgabe besteht jetzt darin, diese Zellen zu löschen. Starten Sie hierfür das Makro aus Listing 3.19.

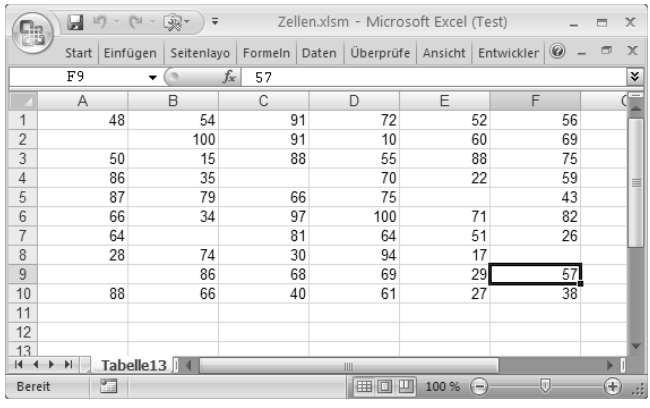
Listing 3.19: Die Inhalte von negativen Zellen entfernen

```
Sub ZelleninhalteLöschenWennNegativ()
Dim Zelle As Range
Dim Bereich As Range

  Set Bereich = Sheets("Tabelle13").Range("A1:F10")

  For Each Zelle In Bereich

     If Zelle.Value < 0 Then
        Zelle.ClearContents
     End If

  Next Zelle

End Sub
```

Zunächst werden zwei Objektvariable vom Typ `Range` deklariert. Danach wird mithilfe der Anweisung `Set` der Variablen `Bereich` bekannt gegeben, auf welcher Tabelle sich der Bereich befindet, der korrigiert werden soll. Nun werden über eine Schleife alle Zellen dieses Bereiches abgearbeitet. Innerhalb der Schleife wird überprüft, ob der Wert der jeweiligen Zelle kleiner als null ist. Sofern dies zutrifft, wird über die Methode `ClearContents` der Inhalt aus der Zelle gelöscht.

Die Methode `ClearContents` löscht lediglich den Inhalt einer Zelle. Sollen gleichzeitig der Wert und die Formate gelöscht werden, dann setzen Sie die Methode `Clear` ein. Diese Methode »putzt« die Zelle komplett. Wenn es darum geht, nur Formatierungen aus einer Zelle zu löschen, wird die Methode `ClearFormats` benötigt.

Abb. 3.18: Die negativen Werte wurden entfernt.



	A	B	C	D	E	F
1	48	54	91	72	52	56
2		100	91	10	60	69
3	50	15	88	55	88	75
4	86	35		70	22	59
5	87	79	66	75		43
6	66	34	97	100	71	82
7	64		81	64	51	26
8	28	74	30	94	17	
9		86	68	69	29	57
10	88	66	40	61	27	38
11						
12						
13						

3.3.8 Bezüge umsetzen

Wenn in einer Tabelle »relative« Zellenbezüge in »absolute« Bezüge umgewandelt werden sollen, dann ist das manuell eine mitunter langwierige Aufgabe. Diese Aufgabe übernimmt besser ein Makro, das alle relativen Zellenbezüge in absolute Bezüge umwandelt.

Listing 3.20: Zellenbezüge umwandeln

```
Sub AbsoluteBezüge()
Dim Zelle As Range

  For Each Zelle In Selection

      If Zelle.HasFormula = True Then
         Zelle.Formula = Application.ConvertFormula _
        (Zelle.Formula, xlA1, , xlAbsolute)
      End If

  Next Zelle

End Sub
```

In einer `For Each...Next`-Schleife werden alle Zellen innerhalb der vorgegebenen Markierung abgearbeitet. Mithilfe der Eigenschaft `HasFormula` wird in der Schleife zunächst geprüft, ob die jeweilige Zelle eine Formel enthält. Wenn ja, dann meldet diese Eigenschaft den Wert `True`. In diesem

Fall werden mit der Methode `ConvertFormula` die relativen Bezüge in absolute Bezüge umgewandelt. Dieser Methode wird als Erstes die Zelle übergeben, die umgesetzt werden soll. Dann legen Sie über eine Konstante den Referenz-Stil fest und geben letztendlich, wiederum über eine Konstante, an, wie die Bezüge umgesetzt werden sollen (`xlAbsolute` bzw. `xlRelativ`).

3.4 Bereiche auswerten

Standardmäßig werden Funktionen wie SUMME, SVERWEIS, ZÄHLENWENN etc. über die normale Oberfläche von Excel in die Tabellen eingetragen. Selbstverständlich besteht auch die Möglichkeit, diese Funktionen über VBA in die einzelnen Zellen zu schreiben. Dabei können diesen Funktionen ganze Bereiche übergeben werden, um das Ergebnis dieser Funktionen auf schnellste Art und Weise zu ermitteln.

Lernen Sie auf den nächsten Seiten ein paar typische Praxisbeispiele kennen.

3.4.1 Einen Bereich summieren

Im ersten Beispiel aus diesem Abschnitt soll ein Datenbereich aufsummiert werden. Dazu werden in einer Tabelle ein paar Umsätze in Spalte B erfasst. In Spalte A wird das dazugehörige Datum abgelegt.

Ermitteln Sie jetzt die Summe aller Umsätze, die in der Spalte B stehen und setzen hierfür das Makro aus Listing 3.21 ein.

Listing 3.21: Alle Umsätze aus Spalte B summieren

```
Sub SummeBilden()
Dim Bereich As Range

Set Bereich = Sheets("Tabelle14").Columns(2)

MsgBox "Die Summe aller Beträge ergibt: " & _
        Application.WorksheetFunction.Sum(Bereich), vbInformation

End Sub
```

Deklarieren Sie im ersten Schritt des Makros aus Listing 3.21 eine Objektvariable vom Typ `Range` und geben über die Anweisung `Set` bekannt, wo der Bereich liegt, der summiert werden soll. Danach wird die Tabellenfunktion SUMME durch Verwendung des Objektes `WorksheetFunction` aufgerufen. Dieses Objekt enthält nahezu alle Tabellenfunktionen, die standardmäßig in Excel eingesetzt werden können. Unter anderem auch die Funktion `Sum`, der der vorher definierte Bereich übergeben wird. Am Ende des Makros wird das Ergebnis der Formel am Bildschirm in einem extra Fenster dargestellt.

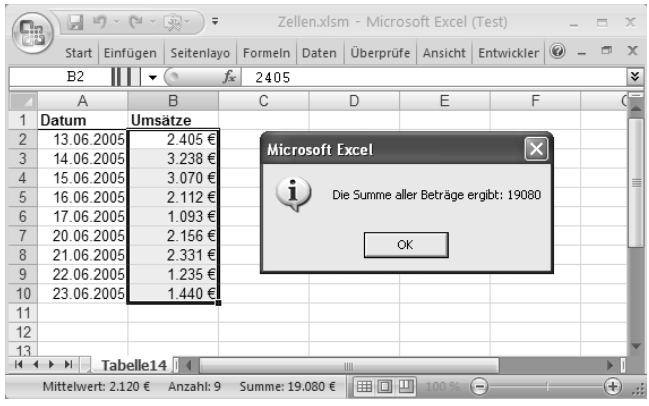


Abb. 3.19: Kontrollieren Sie die Summe in der Statusleiste.

Alle VBA-Tabellenfunktionen müssen in der englischen Syntax angegeben werden, was leicht verwirren kann, wenn Sie täglich in der normalen Arbeitsoberfläche mit den deutschen Tabellenfunktionen arbeiten. Microsoft bietet Ihnen aber mit der Auslieferung einer extra Tabelle, in der eine Gegenüberstellung der deutschen und englischen Funktionen enthalten ist, eine Unterstützung an. Diese Arbeitsmappe heißt VBA-LISTE.XLS und befindet sich in einem Unterverzeichnis von Office. Suchen Sie diese Arbeitsmappe am besten mit der Windows-Suchen-Funktion.

3.4.2 Einen Bereich bedingt summieren

Sollen aus einer Tabelle nur bestimmte Werte summiert werden, beispielsweise nur Umsätze, die einen Wert größer als 2.000 Euro aufweisen, dann kann diese Aufgabe über das folgende Makro aus Listing 3.22 gelöst werden:

Listing 3.22: Eine bedingte Summierung durchführen

```
Sub SummeBildenGrößer2000()
Dim Bereich As Range

  Set Bereich = Sheets("Tabelle14").Columns(2)

  MsgBox "Die Summe aller Beträge ergibt: " & _
         Application.WorksheetFunction.SumIf(Bereich, ">2000"), _
         vbInformation

End Sub
```

Die Funktion SumIf benötigt in diesem Beispiel zwei Argumente. Im ersten Argument geben Sie den Bereich an, in dem die Daten zu summieren sind. Im zweiten Argument wird die Bedingung für die Summierung angegeben.

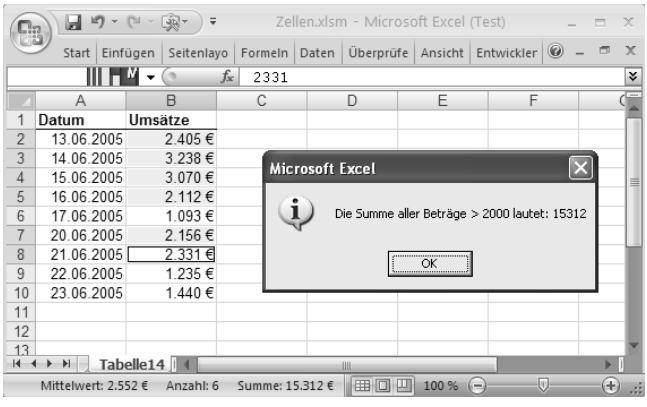


Abb. 3.20: Es werden nur Umsätze summiert, die größer als 2.000 Euro sind.

3.4.3 Numerische Werte zählen

Im Beispiel aus Listing 3.23 sollen in einer Tabelle alle numerischen Werte gezählt werden.

```
Sub ZählenNumerischeWerte()
Dim intz As Integer
Dim Bereich As Range

Set Bereich = Sheets("Tabelle15").Range("A1:D10")

intz = Application.WorksheetFunction.Count(Bereich)

 MsgBox "Anzahl der numerischen Werte:" & _
 vbLf & intz, vbInformation

End Sub
```

Listing 3.23: Alle numerischen Zellen werden gezählt.

Mithilfe der Funktion COUNT (= ANZAHL) werden nur die numerischen Werte einer Liste gezählt. Der zu überprüfende Bereich wird direkt im Anschluss an diese Tabellenfunktion in Klammern angegeben.

Ist es egal, was in einer Zelle steht, sei es nun ein numerischer oder ein alphanumerischer Wert, dann muss zur Zählung dieser Zellen die Tabellenfunktion CountA (= ANZAHL2) verwendet werden.

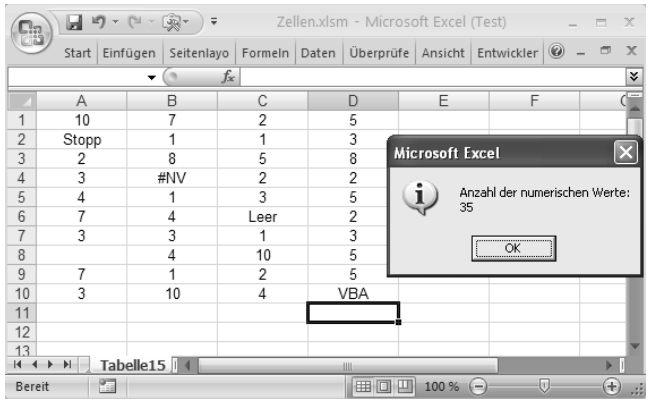


Abb. 3.21: Die Zellen mit numerischem Inhalt werden gezählt.

3.4.4 Eine bedingte Zählung durchführen

Beim nächsten Beispiel greifen Sie auf die Tabelle14 zu, in der vorher alle Umsätze summiert wurden, die größer als 2.000 Euro waren. Diese Umsätze sollen nun gezählt werden. Sehen Sie sich dazu das Makro aus Listing 3.24 an.

Listing 3.24: Eine bedingte Zählung durchführen

```
Sub AnzahlUmsätzeGrößer2000()
Dim Bereich As Range

  Set Bereich = Sheets("Tabelle14").Columns(2)

  MsgBox "Die Anzahl aller Beträge > 2000 lautet: " & _
         Application.WorksheetFunction.CountIf(Bereich, ">2000"), _
         vbInformation

End Sub
```

Zunächst wird der zu zählende Bereich, in dem die Anzahl der Umsätze ermittelt werden sollen, die größer als 2.000 Euro sind, der Funktion `CountIf` übergeben. Die Bedingung >2000 formulieren Sie gleich im Anschluss daran.

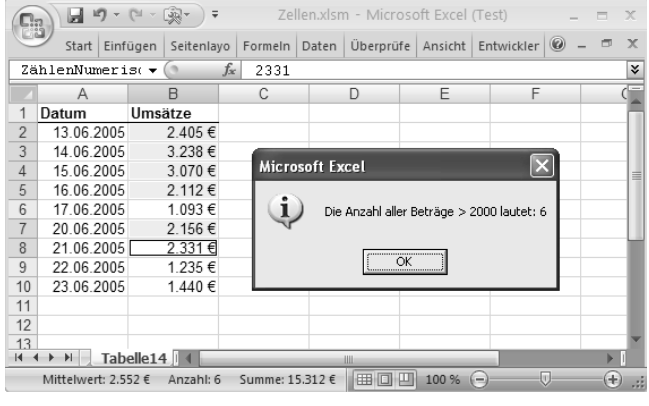


Abb. 3.22: Alle Umsätze ab einem bestimmten Wertebereich werden gezählt.

3.5 Zellen suchen

Das Aufspüren von bestimmten Daten in Tabellen kann entweder über die integrierte Suchen-Funktion von Excel durchgeführt werden, alternativ über den Einsatz der bedingten Formatierung oder über das Programmieren eines Suchen-und-Ersetzen-Makros.

3.5.1 Suche in einem bestimmten Bereich durchführen

Im Beispiel aus Listing 3.25 wird eine Suche durchgeführt, die sich auf einen bestimmten Bereich einer Tabelle beschränkt. Dabei soll eine eindeutige Nummer im Bereich A1:A10 gefunden werden.

Listing 3.25: Eine eindeutige Nummer in einem Bereich finden

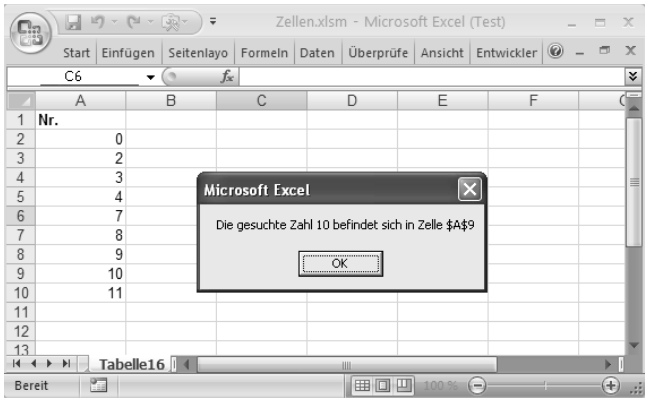
```
Sub NummernSuche()
Dim zelle As Range
Dim Bereich As Range
Dim sBegriff As String

sBegriff = InputBox("Bitte Zahl eingeben:")
If sBegriff = "" Then Exit Sub

Set Bereich = Sheets("Tabelle16").Range("A1:A10")
Set zelle = Bereich.Find(sBegriff, LookAt:=xlWhole, _
                         LookIn:=xlValues)

If zelle Is Nothing Then
    MsgBox "Zahl wurde nicht gefunden!"
Else
    MsgBox "Zahl befindet sich in Zelle " & _
    zelle.Address
End If

End Sub
```

Zunächst wird eine Objektvariable vom Typ `Range` deklariert und danach über die Anweisung `Set` definiert, wo sich der Bereich befindet, der durchsucht werden soll.

Für die Suche wird die Methode `Find` angewendet. Im ersten Argument der Methode befindet sich der Suchbegriff, nach dem gesucht werden soll. Dieser wurde kurz vorher über eine InputBox vom Anwender erfasst. Im Argument `LookAt` kann definiert werden, ob der identische Suchbegriff im Bereich gesucht werden soll oder ob es auch reicht, wenn der Suchbegriff zu Teilen im Bereich gefunden wird. Im Argument `LookIn` können Sie festlegen, ob nach Formeltext bzw. nach Werten gesucht werden soll. Damit die

Suche nur im definierten Bereich durchgeführt wird, stellen Sie den Befehl direkt vor die Methode. Das Ergebnis der Suche wird in der Objektvariablen `Zelle` gespeichert. Kann keine Zelle gefunden werden, die dem Suchkriterium entspricht, dann bleibt diese Variable leer. Über die Prüfung per Anweisung `Is Nothing` kann das diesbezügliche Suchergebnis angezeigt werden. Im anderen Fall lassen sich zum Beispiel die Zellenkoordinaten der Fundzelle über die Eigenschaft `Address` anzeigen.

Abb. 3.23: Die Fundstelle wurde ermittelt.



3.5.2 Nach einem Datum suchen

Im folgenden Beispiel sind in Spalte A Datumsangaben erfasst. Die Aufgabe besteht nun darin, die Zelle, die das aktuelle Tagesdatum enthält, aufzuspüren. Starten Sie zu diesem Zweck das Makro aus Listing 3.26.

Listing 3.26: Das aktuelle Tagesdatum finden

```
Sub DatumSuche()
Dim zelle As Range
Dim Bereich As Range

 Set Bereich = Sheets("Tabelle17").Columns(1)
 Set zelle = Bereich.Find(Date, LookAt:=xlWhole, LookIn:=xlValues)

 If zelle Is Nothing Then
    MsgBox "Das aktuelle Datum wurde nicht gefunden!"
 Else
    MsgBox "Das aktuelle Datum " & Date & " _
            befindet sich in Zelle " & zelle.Address
 End If

End Sub
```

Übergeben Sie der Methode `Find` als Suchbegriff das aktuelle Tagesdatum, das mit der Funktion `Date` abgefragt werden kann. Die übrigen Angaben entnehmen Sie bitte aus der Beschreibung zu Listing 3.25.

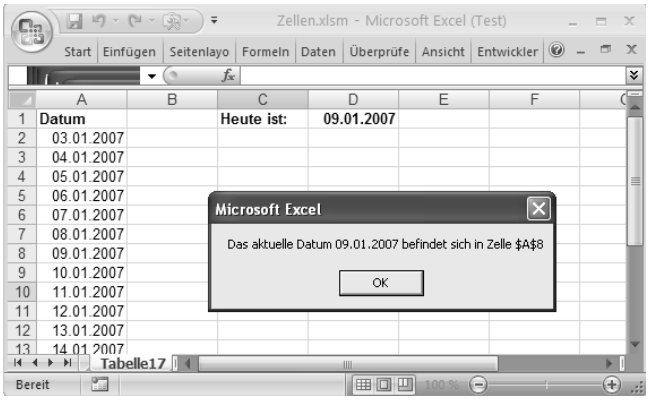


Abb. 3.24: Die Zellenadresse des aktuellen Tagesdatums finden