

jetzt lerne ich

# VBA mit Excel

Der einfache Einstieg in das Programmieren  
von Tabellen und Diagrammen

BERND HELD



Markt+Technik

## 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 immer 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.



Alle hier vorgestellten Makros finden Sie auf der mitgelieferten CD-ROM im Verzeichnis *Kap03* unter dem Namen *Zellen.xls*.



### 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 »darauf zu sitzen«. 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 folgenden 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")

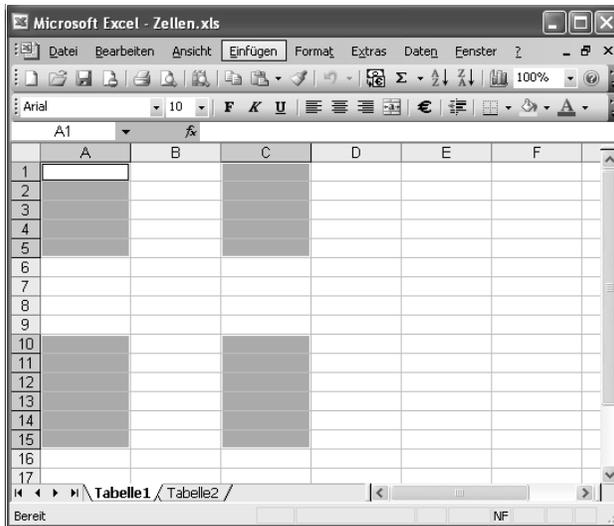
        Set Ber1 = .Range("A1:A5")
        Set Ber2 = .Range("C1:C5")
        Set Ber3 = .Range("A10:A15")
        Set Ber4 = .Range("C10:C15")
        Set Bereiche = Union(Ber1, Ber2, Ber3, Ber4)

        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 beschrieben über die Methode `Select` komplett markieren.



**Bild 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 Ihnen das Makro in Listing 3.2 helfen. Dort wird die `SpecialCells`-Methode verwendet, der Sie die Konstante `xlCellTypeLastCell` mitgeben. Diese Methode liefert Ihnen die Zellenadresse der letzten Zelle im verwendeten Bereich.

```
Sub LetzteZelleImBenutztenBereichErmitteln()

    With Sheets("Tabelle2")

        .Activate
        .Cells.SpecialCells(xlCellTypeLastCell).Select

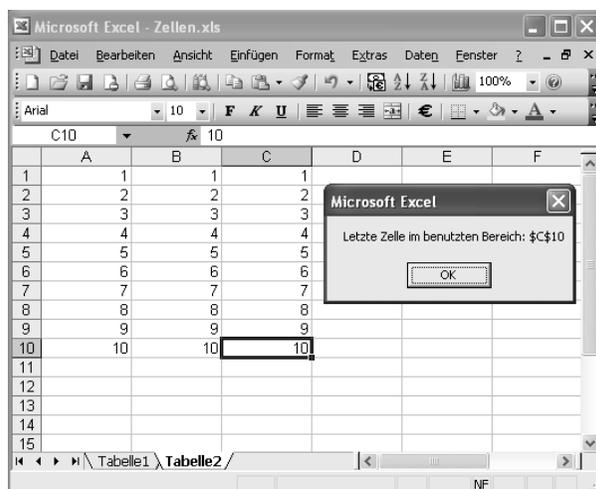
    End With

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

End Sub
```

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

Bild 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 in Listing 3.3 wird die letzte Zelle aus Spalte B markiert.

```
Sub LetzteZelleAusSpalteErmitteln()

With Sheets("Tabelle2")

    .Activate
    .Range("B65536").End(xlUp).Select

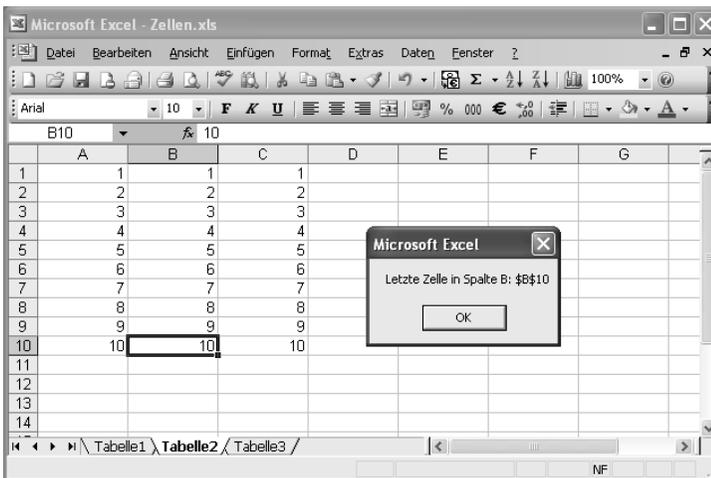
End With

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

End Sub
```

*Listing 3.3:  
Die letzte Zelle  
einer Spalte  
markieren*

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 B65536, durch und springt danach nach oben bis zur letzten gefüllten Zelle der Spalte.



*Bild 3.3:  
Die letzte Zelle  
aus Spalte B  
wird ermittelt*

### 3.1.4 Die letzte Zelle einer Zeile markieren

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

*Listing 3.4:* Sub LetzteZelleAusZeileErmitteln()  
Die letzte Zelle  
einer Zeile  
wird ermittelt

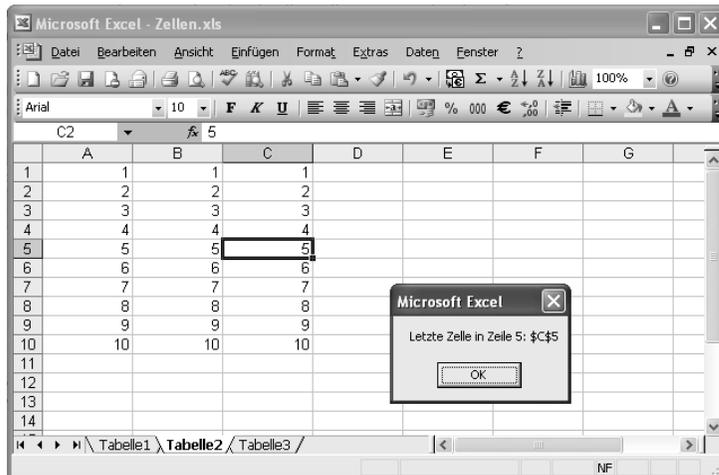
```
With Sheets("Tabelle2")
    .Activate
    .Range("B5").End(xlToLeft).Select

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.

*Bild 3.4:*  
Die letzte Zelle  
aus Zeile 5  
wird ermittelt



### 3.1.5 Alle Formelzellen markieren

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

```

Sub FormelzellenMarkieren()
On Error GoTo fehler

With Sheets("Tabelle3")

    .Activate
    .Cells.SpecialCells(xlCellTypeFormulas).Select

End With
Exit Sub

```

*Listing 3.5:  
Alle Zellen mit  
Formeln wer-  
den markiert*

```

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 anschließend diese Zellen 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 Paragraphen `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.



	Januar	Februar	März	April
Kosten	11.422	26.354	10.750	27.953
Umsatz	24.768	23.710	12.550	17.488
Ergebnis	13.346	-2.644	1.800	-10.465

*Bild 3.5:  
Alle Formelzel-  
len werden  
markiert*

### 3.1.6 Die umliegenden Zellen markieren

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

Sehen Sie den Unterschied der beiden Eigenschaften in den folgenden Beispielen aus den Makros der Listings 3.6 und 3.7.

*Listing 3.6:* Sub BenutzerBereichMarkieren()  
 Der benutzte Bereich einer Tabelle wird markiert

```

Sub BenutzerBereichMarkieren()
    Sheets("Tabelle4").UsedRange.Select
End Sub
  
```

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

*Bild 3.6:*  
 Der benutzte Bereich der Tabelle wird komplett markiert

The screenshot shows an Excel spreadsheet with the following data:

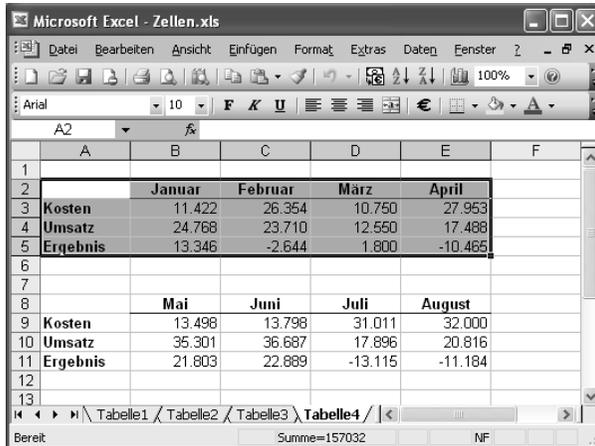
	Januar	Februar	März	April
Kosten	11.422	26.354	10.750	27.953
Umsatz	24.768	23.710	12.550	17.488
Ergebnis	13.346	-2.644	1.800	-10.465
	Mai	Juni	Juli	August
Kosten	13.498	13.798	31.011	32.000
Umsatz	35.301	36.687	17.896	20.816
Ergebnis	21.803	22.889	-13.115	-11.184

*Listing 3.7:* Sub UmliegendenBereichMarkieren()  
 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.



	Januar	Februar	März	April
Kosten	11.422	26.354	10.750	27.953
Umsatz	24.768	23.710	12.550	17.498
Ergebnis	13.346	-2.644	1.800	-10.465
	Mai	Juni	Juli	August
Kosten	13.498	13.798	31.011	32.000
Umsatz	35.301	36.687	17.896	20.816
Ergebnis	21.803	22.889	-13.115	-11.184

*Bild 3.7:  
Den umliegenden Bereich von 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 Blick sichtbar. Sollen alle verbundenen Zellen einer Tabelle optisch hervorgehoben werden, übernimmt das Makro in Listing 3.8 diese Aufgabe.

```
Sub VerbundeneZellenFormatieren()
Dim Zelle As Range

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

    If Zelle.MergeCells = True Then
        Zelle.Interior.ColorIndex = 4
    End If
End For
```

*Listing 3.8:  
Verbundene Zellen einfärben*

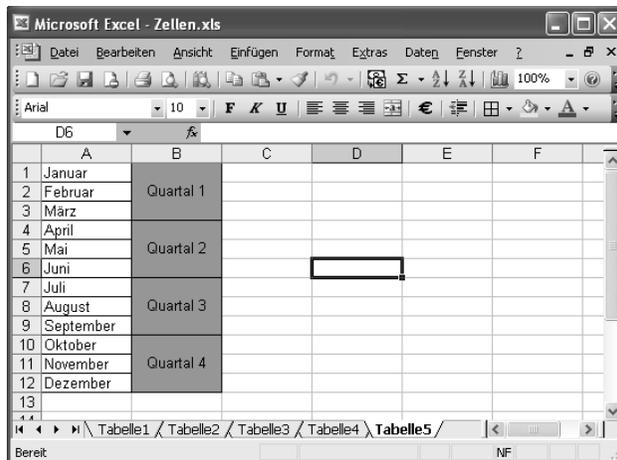
```
Else
    Zelle.Interior.ColorIndex = xlColorIndexNone
End If
```

```
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.

*Bild 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 werden. Das Makro für diese Aufgabe können Sie in Listing 3.9 finden. Markieren Sie vor dem Start des Makros ein paar beliebige Zellen, die bereits Texte oder Zahlen enthalten.

*Listing 3.9:* Sub SchriftenFormatieren()  
Alle markierten Zellen formatieren

```
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.

	Januar	Februar	März	April
<b>Kosten</b>	11.422	26.354	10.750	27.953
<b>Umsatz</b>	24.768	23.710	12.550	17.488
<b>Ergebnis</b>	13.346	-2.644	1.800	-10.465
	Mai	Juni	Juli	August
<b>Kosten</b>	13.498	13.798	31.011	32.000
<b>Umsatz</b>	35.301	36.687	17.896	20.816
<b>Ergebnis</b>	21.803	22.889	-13.115	-11.184

*Bild 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
<code>Bold</code>	Diese Eigenschaft liefert den Wert <code>True</code> , wenn ein Text fett formatiert ist.
<code>ColorIndex</code>	Diese Eigenschaft gibt die Farbe des Rahmens, der Schriftart oder des Innenraums zurück. Es existieren in Excel genau 56 Farben.
<code>FontStyle</code>	Diese Eigenschaft sagt aus, welcher Schriftschnitt verwendet wird. Möglich sind u. a. Fett- und Kursivdruck.
<code>Italic</code>	Diese Eigenschaft liefert den Wert <code>True</code> , wenn ein Text kursiv formatiert ist.
<code>OutlineFont</code>	Diese Eigenschaft liefert den Wert <code>True</code> , wenn ein Text als Konturschriftart formatiert wird.
<code>Shadow</code>	Diese Eigenschaft liefert den Wert <code>True</code> , wenn ein Text als schattierte Schriftart formatiert wird.
<code>Strikethrough</code>	Diese Eigenschaft liefert den Wert <code>True</code> , wenn ein Text als horizontal durchgestrichen dargestellt wird.
<code>Subscript</code>	Diese Eigenschaft liefert den Wert <code>True</code> , wenn ein Text tiefergestellt formatiert wird.
<code>Superscript</code>	Diese Eigenschaft liefert den Wert <code>True</code> , wenn ein Text hochgestellt formatiert wird.
<code>Underline</code>	Diese Eigenschaft liefert den Wert <code>True</code> , wenn ein Text unterstrichen formatiert wird. Dabei können Sie u. a. Text einfach oder doppelt unterstreichen.

### 3.2.3 Extremwerte kennzeichnen

Im nächsten Beispiel in 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:* Sub ExtremwerteHervorheben()  
 Alle Werte größer oder gleich 100 werden optisch hervorgehoben

```

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, 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 »Wertegrenze« wieder farblich zurückzusetzen. Es kann schließlich vorkommen, dass Werte im vorgegebenen Bereich geändert werden. 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.



	A	B	C	D	E	F
1	77	27	105	17	79	
2	65	17	9	9	67	
3	89	92	99	62	95	
4	43	16	7	58	104	
5	105	106	11	95	47	
6	10	78	109	89	85	
7	51	59	15	93	9	
8	22	98	85	91	68	
9	18	60	3	27	16	
10	7	66	86	61	73	
11	1	86	70	11	69	
12	40	91	61	41	89	
13						

*Bild 3.10: Extremwerte über 100 werden mit der Hintergrundfarbe Grün hinterlegt*

### 3.2.4 Bereiche einrahmen

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

*Listing 3.11:* Sub RahmenFestlegen()  
*Mehrere Bereiche werden eingerahmt*  
 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)

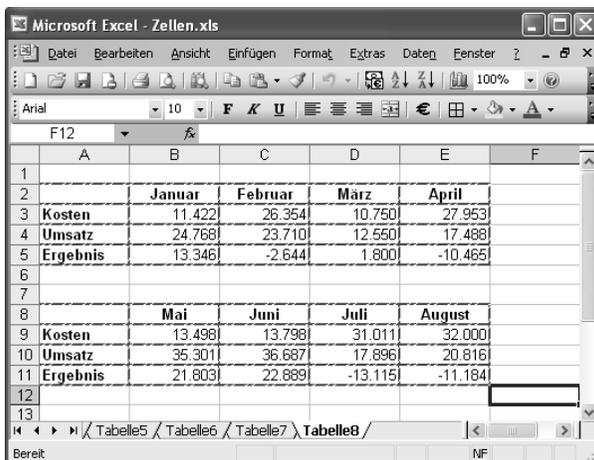
    With Gesamt.Borders
        .ColorIndex = 3
        .LineStyle = xlSlantDashDot
    End With
End With
```

End 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.

*Bild 3.11:*  
 Eine individuelle Rahmung von Bereichen durchführen



	Januar	Februar	März	April
Kosten	11.422	26.354	10.750	27.953
Umsatz	24.768	23.710	12.550	17.488
Ergebnis	13.346	-2.644	1.800	-10.465
	Mai	Juni	Juli	August
Kosten	13.498	13.798	31.011	32.000
Umsatz	35.301	36.687	17.896	20.816
Ergebnis	21.803	22.889	-13.115	-11.184

Mit dem Objekt `Borders` wird nun die Umrahmung 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.

Sehen Sie in der nächsten Tabelle weitere Möglichkeiten, um die Form eines Rahmens zu definieren.

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

*Tabelle 3.4:  
Die weiteren  
Möglichkeiten  
der Rahmen-  
formen*

## 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 Sie versuchen, die Daten weiter zu verarbeiten, zum Beispiel grafisch oder in Kalkulationen, entstehen Probleme.

Im folgenden Beispiel in 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:* Sub ZellenBereinigen()  
*Alle Zellen im* Dim Zelle As Range  
*benutzten Be-* For Each Zelle In ActiveSheet.UsedRange  
*reich werden* With Zelle  
*gesäubert* 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 Sie versuchen eine Summe über die Zahlenwerte zu bilden. Meldet die Funktion den Wert 0, können Sie davon ausgehen, dass Excel die Zahlenwerte als Text interpretiert. Üblicherweise werden Sie 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 Markieren der Zelle und Drücken der Taste `F2` und anschließend der `↵`-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 in Listing 3.13 dargestellt.

```
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
```

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

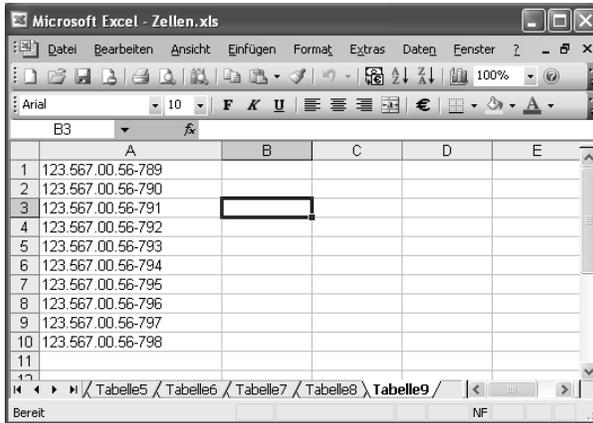
Zu Beginn wird eine Objektvariable vom Typ `Range` definiert. Danach werden mithilfe einer `For Each...Next`-Schleife alle markierten Zellen angesteuert. Innerhalb der Schleife erfolgt eine Prüfung, ob überhaupt ein numerischer Wert (`IsNumeric`) vorliegt oder ob sich in der jeweiligen Zelle eine Formel (`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

Um bestimmte Zeichen aus Zellen zu eliminieren, was in der Fachsprache als »parsen« bezeichnet wird, lässt sich mit dem folgenden Makro viel mühevoll Handarbeit ersparen.

Im folgenden Beispiel in 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.

**Bild 3.12:**  
Die Ausgangs-  
liste mit diver-  
sen Sonder-  
zeichen



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

*Listing 3.14:* Sub BestimmteZeichenEliminieren()  
*Definierte Zeichen werden aus allen markierten Zellen entfernt*

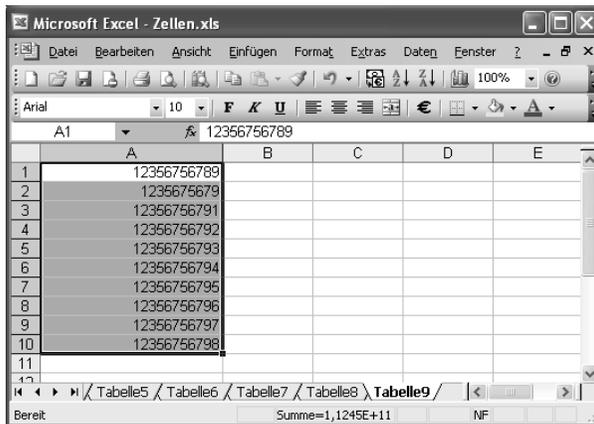
```

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
        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 innerhalb einer Zelle Zeichen für Zeichen mithilfe der Funktion Mid 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.



**Bild 3.13:**  
Alle definierten Zeichen wurden entfernt

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

```
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)
```

**Listing 3.15:**  
Alle alphanumerischen Zeichen aus den markierten Zellen entfernen

```
Case Else
```

```
End Select
```

```
Next intz
```

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

```
StrNeu = ""
```

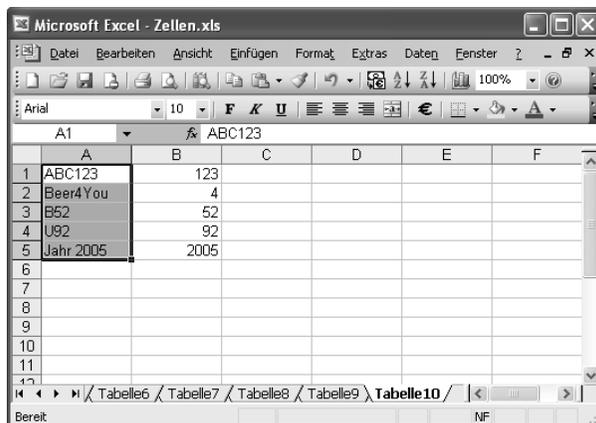
```
Next Zelle
```

```
End Sub
```

Das Makro in Listing 3.15 entspricht weitestgehend dem Makro in 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, tragen Sie in diesem Argument den Wert 1 ein.

*Bild 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 folgende Makro in Listing 3.16:

```
Sub NamenDrehen()
Dim Zelle As Range

For Each Zelle In Selection

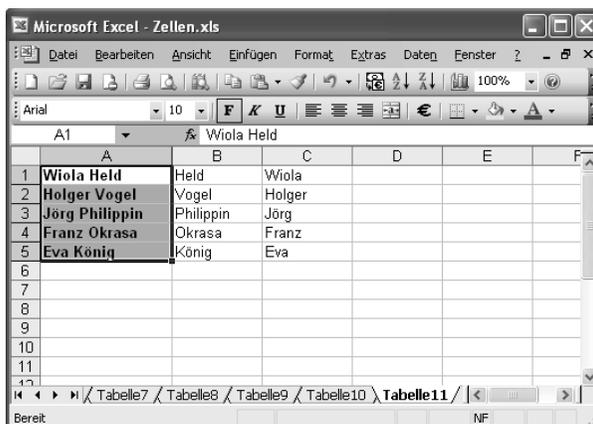
    Zelle.Offset(0, 1).Value =
    Mid(Zelle.Value, InStr(Zelle.Value, " ") + 1)
    Zelle.Offset(0, 2).Value =
    Left(Zelle.Value, InStr(Zelle.Value, " ") - 1)

Next Zelle

End Sub
```

*Listing 3.16:  
Der Vor- sowie  
Nachname  
wurde gedreht*

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 Nachname 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.



*Bild 3.15:  
Eine Zelle zer-  
legen und  
wieder in  
anderer Form  
zusammen-  
setzen*

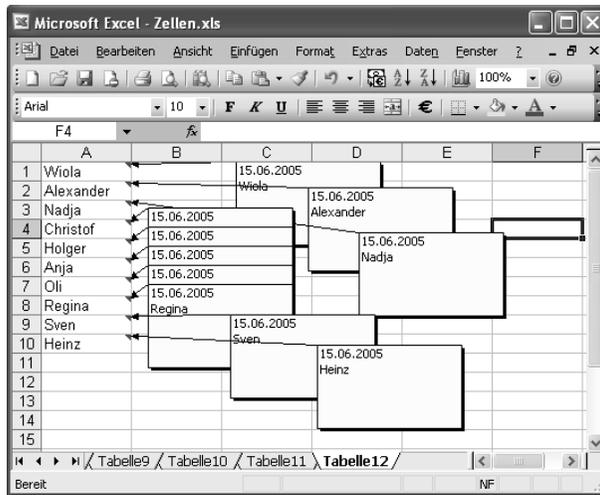
Den Vornamen können Sie direkt über die Funktion `Left` aus der jeweiligen Zelle der Spalte A extrahieren. Der Vorname muss bis zu einem Zeichen vor dem ermittelten Leerzeichen übertragen werden. Auch zu diesem Zweck wird die Funktion `Instr` verwendet und von der gefundenen Position der Wert 1 subtrahiert.

### 3.3.5 Kommentare aus Zellentexten generieren

Gerade haben Sie gelernt, wie Zellinhalte überprüft werden und 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: Sub KommentareAusZellenInhaltBilden()  
Zelleninhalte Dim Kom As Comment  
als Kommentar- Dim Zelle As Range  
artext über- For Each Zelle In Selection  
nehmen     On Error Resume Next  
           Set Kom = Zelle.AddComment  
           Kom.Text Date & vbLf & Zelle.Value  
  
Next Zelle  
  
End Sub
```

Das Makro in Listing 3.17 verschiebt innerhalb der Markierung den Inhalt der einzelnen Zellen jeweils in ein Kommentarfenster. Jeder Kommentar wird zusätzlich noch mit dem Datum der Verschiebung ausgestattet.



**Bild 3.16:**  
Aus den  
Zelleninhalten  
wurden Kommen-  
tare mit  
dem aktuellen  
Datum gebil-  
det

### 3.3.6 Formeln durch Festwerte ersetzen

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

```
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
```

**Listing 3.18:**  
Alle Formel-  
zellen werden  
durch Fest-  
werte ersetzt

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 als Ergebnis aktuell darstellt.

### 3.3.7 Bestimmte Zelleninhalte löschen

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

*Bild 3.17:  
Die Ausgangs-  
tabelle mit  
negativen so-  
wie positiven  
Werten*

	A	B	C	D	E	F	G
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 in Listing 3.19.

*Listing 3.19:* 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 Objektvariablen vom Typ `Range` deklariert. Danach wird mithilfe der Anweisung `Set` der Variablen `Bereich` bekannt gegeben, in welcher Tabelle sich der Bereich befindet, der korrigiert werden soll. Nun werden über eine Schleife alle Zellen dieses Bereichs 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.



	A	B	C	D	E	F	G
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							

**Bild 3.18:**  
Die negativen Werte wurden entfernt

### 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.

```
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
```

**Listing 3.20:**  
Zellenbezüge umwandeln

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 (`x1Absolute` bzw. `X1Relativ`).

## 3.4 Bereiche auswerten

Standardmäßig werden Funktionen wie `SUMME`, `SVERWEIS`, `ZÄHLENWENN` usw. 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. Dafür können ganze Bereiche diesen Funktionen 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

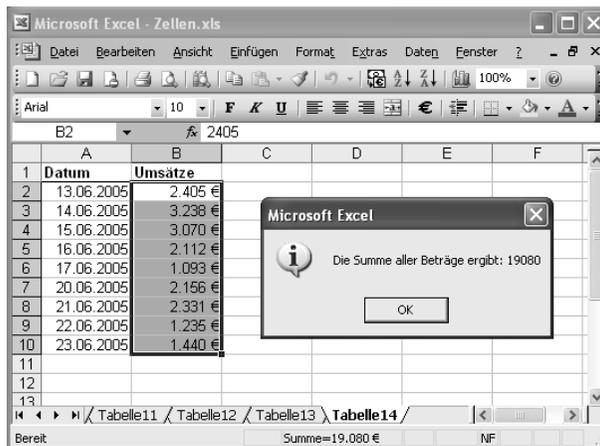
Im ersten Beispiel aus diesem Abschnitt soll ein Datenbereich aufsummiert werden. Dazu werden in einer Tabelle ein paar Umsätze in Spalte B erfasst. In Spalte A wird das dazugehörige Datum abgelegt.

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

```
Listing 3.21: Sub SummeBilden()
Alle Umsätze Dim Bereich As Range
aus Spalte B Set Bereich = Sheets("Tabelle14").Columns(2)
summieren MsgBox "Die Summe aller Beträge ergibt: " & _
Application.WorksheetFunction.Sum(Bereich), vbInformation

End Sub
```

Deklariert Sie im ersten Schritt des Makros in 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 Objekts `WorksheetFunction` aufgerufen. Dieses Objekt enthält nahezu alle Tabellenfunktionen, die standardmäßig in Excel eingesetzt werden können. Unter anderem auch die Funktion `Sum`, die den vorher definierten Bereich übergeben bekommt. Nun wird am Ende des Makros das Ergebnis der Formel am Bildschirm in einem extra Fenster dargestellt.



**Bild 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 mit den deutschen Tabellenfunktionen in der normalen Arbeitsfläche arbeiten. Microsoft bietet Ihnen aber eine Unterstützung durch die Auslieferung einer extra Tabelle an, in der eine Gegenüberstellung der deutschen und englischen Funktionen enthalten ist. Diese Arbeitsmappe heißt `VBAListe.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 in Listing 3.22 gelöst werden:

*Listing 3.22:* Sub SummeBildenGrößer2000()  
 Eine bedingte Summierung durchführen

```

Dim Bereich As Range

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

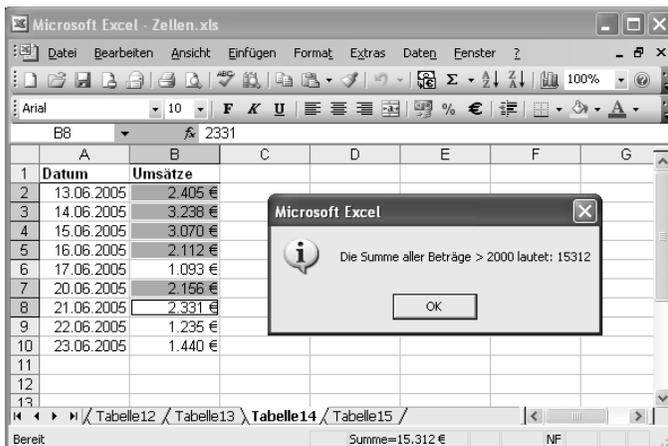
MsgBox "Die Summe aller Beträge ergibt: " & _
    Application.WorksheetFunction.SumIf(Bereich, ">2000"), _
    vbInformation

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.

*Bild 3.20:*  
 Es werden nur Umsätze summiert, die größer als 2.000 Euro sind



### 3.4.3 Numerische Werte zählen

Im folgenden Beispiel in 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)

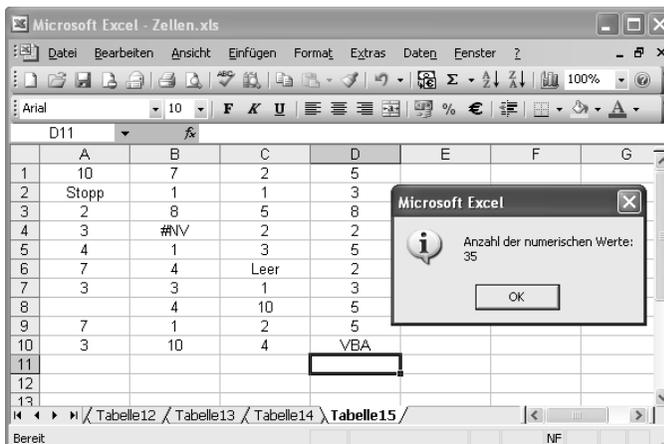
MsgBox "Anzahl der numerischen Werte:" & _
vbLf & intz, vbInformation

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.



*Bild 3.21:  
Die Zellen mit numerischem Inhalt werden gezählt*

### 3.4.4 Eine bedingte Zählung durchführen

Beim nächsten Beispiel greifen Sie auf die TABELLE 14 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 in Listing 3.24 an.

*Listing 3.24:* Sub AnzahlUmsätzeGrößer2000()  
 Eine bedingte Zählung durchführen

```

Dim Bereich As Range

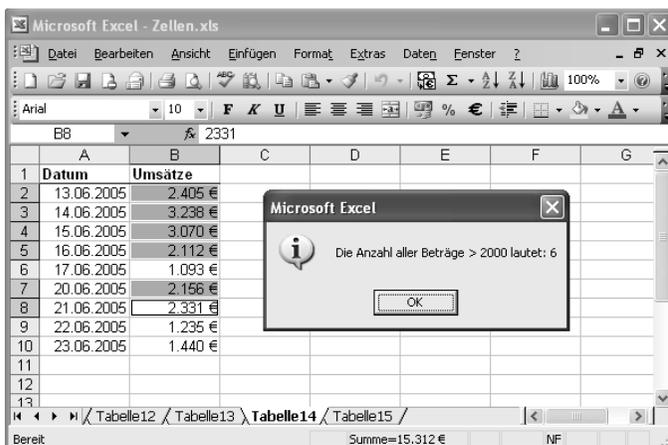
Set Bereich = Sheets("Tabelle14").Columns(2)

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 soll, die größer als 2.000 Euro sind, der Funktion CountIf übergeben. Die Bedingung >2000 formulieren Sie gleich im Anschluss daran.

*Bild 3.22:*  
 Alle Umsätze  
 ab einem be-  
 stimmten Wer-  
 tebereich wer-  
 den gezählt



## 3.5 Zellen suchen

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

### 3.5.1 Suche in einem bestimmten Bereich durchführen

Im folgenden Beispiel in 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.

```
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
```

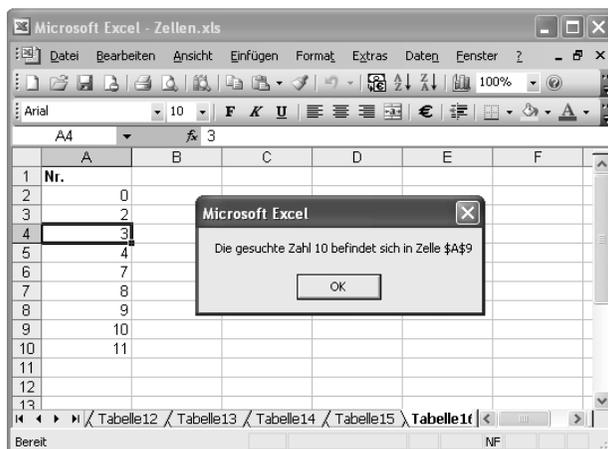
*Listing 3.25:  
Eine eindeutige  
Nummer in  
einem Bereich  
finden*

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 definierten Bereich direkt vor die Methode. Das Ergebnis der Suche wird in der Objektvariablen `zelle` gespeichert. Kann keine Zelle gefunden werden, die dem Suchkriterium entspricht, 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.

Bild 3.23:  
Die Fundstelle  
wurde er-  
mittelt



### 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 in Listing 3.26.

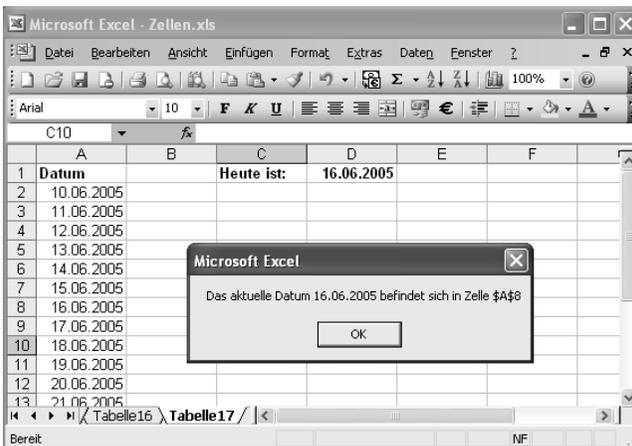
```
Listing 3.26: Sub DatumSuche()
Das aktuelle Dim zelle As Range
Tagesdatum Dim Bereich As Range
finden

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 Beschreibungen entnehmen Sie bitte aus den Erläuterungen des Listings 3.25.



*Bild 3.24:  
Die Zellen-  
adresse des  
aktuellen  
Tagesdatums  
finden*