Die Sprachelemente von VBA

Am gestrigen Tag haben Sie Datentypen, Variablen und den Einsatz von Konstanten kennen gelernt. Sie wissen nun, warum Sie Variablen einsetzen und wie Sie diese deklarieren können.

Die Themen heute:

- Arbeiten mit Verzweigungen
- Übersichtlichere Form mit Select Case
- Schleifen programmieren

Das Wesentliche einer Programmiersprache sind ihre Sprachelemente. In diesem Kapitel erfahren Sie, wie Sie mithilfe von Verzweigungen, Schleifen und anderen Anweisungen Ihre Programme flexibel gestalten können. Diese Sprachelemente lassen sich leider nicht mit dem Makrorekorder aufzeichnen und müssen von Ihnen selbst erstellt werden. Der richtige Einsatz der Sprachelemente macht letztendlich die Kunst der Programmierung aus.

Alle Makros dieses Tages finden Sie auf der mitgelieferten CD-ROM im Verzeichnis *Kap03* unter dem Namen *Sprachelemente.xls*.

3.1 Arbeiten mit Verzweigungen

Unter einer Verzweigung versteht man ganz allgemein eine Prüfung. Diese Prüfung kann entweder das eine oder das andere Ergebnis bringen. Je nach Ergebnis werden Sie einen bestimmten Weg einschlagen.

Verzweigungen im Leben

Dazu ein kleines Beispiel aus der Praxis:

»Wenn wir Geld auf dem Konto haben, dann fahren wir in den Urlaub. Wenn nicht, dann müssen wir zu Hause bleiben und sparen!«

Bildlich dargestellt könnte das wie folgt aussehen:

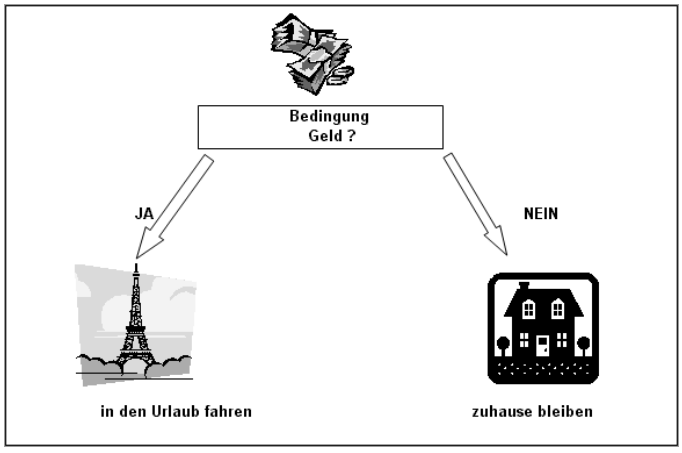

Abbildung 3.1:
Eine einfache Entscheidung

Oft sind Verzweigungen aber noch von weiteren Bedingungen abhängig, d.h. auf unser Beispiel angewendet könnte man noch weitere Kriterien anwenden, wie beispielsweise:

»Wenn wir Geld haben und ein gutes Angebot bekommen, fahren wir in den Urlaub!«

oder

»Wir fahren in den Urlaub, wenn wir Geld und die Kinder Ferien haben!«

Prüfungen können selbstverständlich auch geschachtelt werden, d.h. ein Zweig könnte weitergeführt werden und weitere Verzweigungen enthalten. Angewendet auf unser Beispiel würde das bedeuten:

»Also wir fahren jetzt in den Urlaub! Aber wohin? Sollen wir nach Frankreich oder nach Italien fahren?«

Eine weitere Schachtelung wäre dann: »Wir haben uns für Frankreich entschieden! Aber sollen wir nach Paris oder nach Nizza?«

Wie Sie sehen, können Sie auf diese Weise zig Bedingungen formulieren und zu Papier bringen.

Verzweigungen in Excel

Verzweigungen kennen Sie übrigens bereits aus der normalen Arbeitsoberfläche von Excel. Dort können Sie die Tabellenfunktion `WENN()` einsetzen, um entweder die eine oder die andere Aktion in Abhängigkeit von einem Zellenwert einzuleiten.

Verzweigungen in Excel-VBA

Mit Verzweigungen können Sie in Excel-VBA bestimmte Zustände abfragen und je nach Zustand anders reagieren. Die allgemeine Syntax für eine solche Verzweigung lautet:

```
If Bedingung Then [Anweisungen] [Else elseAnweisungen]
```

Alternativ können Sie die Block-Syntax verwenden:

```
If Bedingung Then
 [Anweisungen]
 [ElseIf Bedingung-n Then
 [elseifAnweisungen] ...
 [Else
 [elseAnweisungen]]
End If
```

Unter dem Argument `Bedingung` bzw. `Bedingung-n` geben Sie die Bedingung(en) an, die erfüllt werden muss.

Unter dem Argument `Anweisungen` werden jene Anweisungen aufgeführt, die durchgeführt werden sollen, wenn die erste Bedingung erfüllt wird.

Unter dem Argument `elseifBedingungen` können Sie weitere Bedingungen formulieren und somit die Abfrage weiter schachteln. Über das Argument `ElseIfAnweisungen` lassen Sie dabei die Anweisungen folgen, sofern die `elseIfBedingung` erfüllt ist.

Über das Argument `elseAnweisungen` können Sie eine oder mehrere Anweisungen folgen lassen, welche ausgeführt werden sollen, wenn die erste Bedingung nicht erfüllt wird.

Lassen Sie uns nun diese etwas theoretische Syntaxbeschreibung auf einige Beispiele aus der Praxis anwenden.

Zelleninhalte prüfen

Im folgenden Beispiel soll der Inhalt der momentan aktivierten Zelle geprüft werden. Im ersten Schritt werden Sie testen, ob überhaupt etwas in der Zelle steht. Das Makro für diese Aufgabe lautet:

Listing 3.1: Prüfen, ob Zelle leer oder gefüllt ist

```
Sub Verzweigung()
  If ActiveCell.Value = "" Then _
  MsgBox "Zelle ist leer" Else MsgBox "Zelle ist gefüllt!"
End Sub
```

Überprüfen Sie zuerst, ob die aktive Zelle gefüllt ist. Alternativ zu der im Listing 3.1 aufgeführten Abfrage könnten Sie auch die Zeile

```
If IsEmpty(ActiveCell.Value) Then
```

verwenden. Beide Bedingungen liefern das gewünschte Ergebnis. Für den Fall, dass die Zelle einen Inhalt enthält, wird der `Then`-Zweig ausgeführt, andernfalls der `Else`-Zweig.

Im nächsten Schritt möchten Sie prüfen, ob der Zelleninhalt, sofern die Zelle gefüllt ist, entweder numerisch oder alphanumerisch ist. Dazu müssen Sie die Verzweigung schachteln. Das könnte dann wie folgt aussehen:

Listing 3.2: Überprüfen des genauen Zelleninhalts

```
Sub VerzweigungDaten()

 If IsEmpty(ActiveCell.Value) Then
  MsgBox "Zelle ist leer"
   Else
   If IsNumeric(ActiveCell) Then
      MsgBox "Zelle enthält eine Zahl"
   Else
      MsgBox "Zelle enthält einen Text"
  End If
 End If

End Sub
```

In der ersten Verzweigung überprüfen Sie, ob die Zelle leer ist. Wenn ja, geben Sie eine Meldung auf dem Bildschirm aus. Wenn die Zelle gefüllt ist, müssen Sie eine zweite Verzweigung einbauen, die prüft, ob es sich um einen numerischen oder alphanumerischen Zellenwert handelt. Mithilfe der Funktion `IsNumeric` können Sie diese Aufgabe erledigen. Vergessen Sie bei beiden Verzweigungen nicht, diese mit `End if` abzuschließen.

Bei dieser Gelegenheit möchte ich Ihnen eine weitere Funktion vorstellen, die Ihnen dabei hilft, herauszufinden, ob ein Datumswert vorliegt. Sehr oft werden solche Datumsüberprüfungen in Excel durchgeführt. Insbesondere bei Berechnungen von Lieferterminen oder Zahlungszielen müssen Sie als Entwickler sicherstellen, dass auch wirklich Datumseingaben vorgenommen wurden. Im nächsten Beispiel werden Sie vom Anwender eine Datumseingabe über eine Eingabemaske verlangen. Das Makro für diesen Zweck lautet:

Listing 3.3: Datumsprüfung vornehmen

```
Sub Datumsprüfung()
Dim d As Date

On Error GoTo fehler
```

```
Beginn:
d = InputBox("Geben Sie das Lieferdatum ein!", _
 "Datum eingeben")

If IsDate(d) And d >= "01.01.2002" Then
 Else
 MsgBox "Nur Eingaben im aktuellen Jahr möglich"
 GoTo Beginn
End If

d = d + 14
 MsgBox "Das Zahlungsziel ist der: " & d
Exit Sub

fehler:
 MsgBox "Sie haben kein gültiges Datum eingegeben!"
 GoTo Beginn
End Sub
```

Im ersten Schritt fordern Sie den Anwender auf, ein Datum einzugeben. Danach kontrollieren Sie mithilfe einer Verzweigung, ob das Datum im gültigen Bereich liegt. Es werden nur Datumseingaben akzeptiert, die größer oder gleich dem Datum 01.01.2002 sind. Prüfen Sie zusätzlich, ob es sich überhaupt um einen gültigen Datumswert handelt. Dazu verwenden Sie die Funktion `IsDate`. Diese Funktion meldet den Wert `True`, wenn es sich um ein Datum handelt. Wurde ein gültiges Datum eingegeben, dann rechnen Sie mit diesem Datum. Dabei können Sie genauso vorgehen, wie Sie es auch bei numerischen Werten machen würden. Addieren Sie zum Liefertermin einfach die Zahl 14 (14 Tage), um einen gängigen Zahlungstermin zu errechnen. Geben Sie diesen Termin dann auf dem Bildschirm aus.

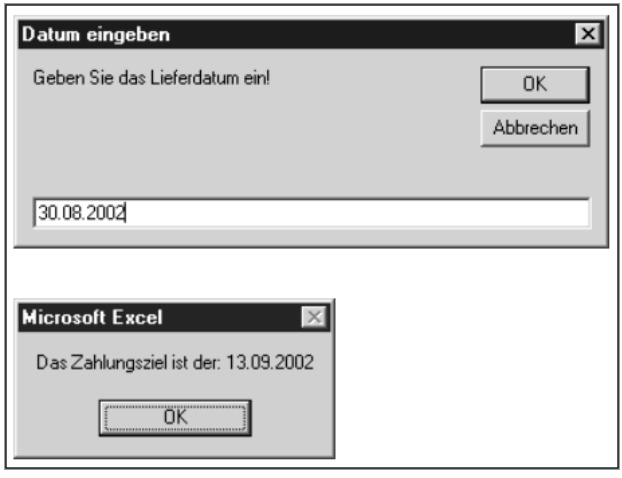

Abbildung 3.2:
Das Datum wurde korrekt eingegeben.

Sollte ein ungültiger Wert in der Eingabemaske erfasst werden, wird diese erneut aufgerufen.

Zu Beginn des Listings finden Sie eine `On Error`-Anweisung. Damit stellen Sie sicher, dass Ihr Makro nicht abstürzt, wenn ein Text eingegeben wird. Sollte ein Anwender in diesem Beispiel einen Text eingeben, wird die Marke `fehler` angesprungen. Dort erhält der Benutzer eine Nachricht, dass ihm bei der Eingabe ein Fehler unterlaufen ist. Mit dem Befehl `GoTo` geben Sie ihm aber die Möglichkeit, seine Eingabe zu wiederholen. Als Sprungziel geben Sie dort die Sprungmarke `Beginn` an.

Sie haben bereits mehrere typische VBA-Funktionen kennen gelernt, die häufig eingesetzt werden, um Eingaben zu überprüfen. In der folgenden Tabelle finden Sie die gängigsten Prüffunktionen in VBA.

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

Tabelle 3.1: Die wichtigsten Prüffunktionen von VBA

Auch eine Art von Verzweigung

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

Die Funktion `IIF` hat folgende Syntax:

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

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

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

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

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

In einem Beispiel angewendet, überprüfen Sie den Inhalt einer Zelle. Sofern der Inhalt numerisch ist, schreiben Sie in die Nebenzelle den Text »Numerisch«. Sollte es sich um einen Text handeln, dann schreiben Sie in die Nebenzelle den Text »Alphanumerisch«.

Listing 3.4: Numerisch oder alphanumerisch

```
Sub AlternativVerzweigung()
ActiveCell.Offset(0, 1).Value = _
IIF(IsNumeric(ActiveCell), "Numerisch", _
                           "Alphanumerisch")
End Sub
```

Das Ergebnis dieser Auswertung schreiben Sie über die Anweisung `Activecell.Offset(0, 1).Value` in die Nebenzelle. Dabei weist die Eigenschaft `Offset` zwei Argumente auf. Das erste Argument gibt die Zeilenverschiebung, von der aktiven Zelle aus gesehen, bekannt. Da Sie in derselben Zeile bleiben möchten, ist demnach die Verschiebung gleich 0. Das zweite Argument gibt die Spaltenverschiebung, von der aktiven Zelle aus gesehen, bekannt. Da das Ergebnis in die Nebenspalte geschrieben werden soll, wird dieses Argument mit dem Wert 1 bestückt.

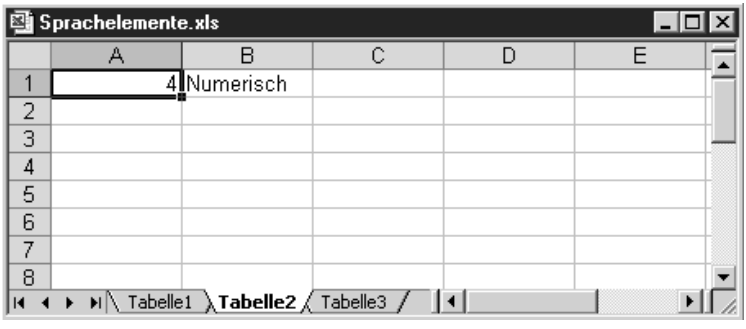

Abbildung 3.3: Zelleneintrag prüfen und Ergebnis in Nebenzelle ausgeben

3.2 Übersichtlichere Form mit Select Case

Wenn Sie mehrere Verzweigungen ineinander schachteln bzw. mehrere Verzweigungen hintereinander durchführen möchten, gibt es dafür eine bessere und übersichtlichere Lösung. Setzen Sie für solche Aufgaben die Anweisung `Select Case` ein.

Die Syntax für `Select Case` lautet:

```
Select Case Testausdruck
[Case Ausdrucksliste-n
      [Anweisungen-n]] ...
[Case Else
      [elseAnw]]
End Select
```

Unter dem Argument `Testausdruck` wird ein beliebiger numerischer Ausdruck oder Zeichenfolgenausdruck erfasst, den Sie auswerten möchten. Im Argument `Ausdrucksliste-n` spezifizieren Sie den zu untersuchenden Ausdruck näher. Dabei können Sie Vergleichsoperatoren, wie `To`, `Is` oder `Like`, verwenden.

Unter dem Argument `Anweisungen-n` können Sie eine oder mehrere Anweisungen angeben, welche ausgeführt werden sollen, wenn `Testausdruck` mit einem beliebigen Teil in `Ausdrucksliste-n` übereinstimmt.

Das Argument `elseAnw` ist optional einsetzbar. Damit können Sie darauf reagieren, wenn `Testausdruck` mit keinem der Ausdrücke im `Case`-Abschnitt übereinstimmen sollte.

Sehen Sie nun ein paar typische Beispiele für den Einsatz von `Select Case`.

Excel-Version feststellen

In der ersten Aufgabe für dieses Sprachelement sollen Sie feststellen, mit welcher Excel-Version Sie arbeiten. Für diese Aufgabe können Sie die Eigenschaft `Version` auswerten, die über einen numerischen Wert Ihre Excel-Installation identifiziert.

Listing 3.5: Excel-Version ermitteln

```
Sub ExcelVersionFeststellen()
MsgBox Application.Version

    Select Case Left(Application.Version, 1)
      Case "5"
        MsgBox "Excel 5"
      Case "7"
        MsgBox "Excel 7/95"
```

```
    Case "8"
      MsgBox "Excel 8/97"
    Case "9"
      MsgBox "Excel 2000"
    Case "1"
      MsgBox "Excel 2002"
    Case Else
      MsgBox "Unbekannte Version von Excel"
  End Select

End Sub
```

Werten Sie über die Funktion `Left` die erste Ziffer der Versionsnummer aus, die Ihnen die Eigenschaft `Version` meldet. Innerhalb der `Select Case`-Anweisung überprüfen Sie die Versionsnummern. Trifft eine Bedingung zu, wird eine Meldung am Bildschirm ausgegeben, die Ihnen die Excel-Version meldet. Danach wird die `Select-Case`-Anweisung direkt beendet.

Zahlenwerte prüfen

Im nächsten Beispiel werden Eingaben geprüft. Dabei soll ermittelt werden, in welchem Wertbereich die Eingabe vorgenommen wurde. Sehen Sie sich dazu das folgende Listing an:

Listing 3.6: Zahlenwerte überprüfen

```
Sub ZahlAuswerten()
Dim i As Integer

i = InputBox _
("Geben Sie einen Wert zwischen 1 und 100 ein!")

Select Case i
 Case 1 To 5
  MsgBox "Wert liegt zwischen 1 und 5"
 Case 6, 7, 8
  MsgBox "Wert ist entweder 6, 7 oder 8"
 Case 9 To 15
  MsgBox "Wert liegt zwischen 9 und 15"
 Case 16 To 100
  MsgBox "Wert liegt zwischen 16 und 100"
 Case Else
  MsgBox "Es wurde kein gültiger Wert eingegeben!"
End Select

End Sub
```

Wenden Sie die `Select Case`-Anweisung an, um die eingegebenen Werte zu überprüfen. In der ersten Abfrage kontrollieren Sie, ob der eingegebene Wert zwischen 1 und 5 liegt. In diesem Fall können Sie den Vergleichsoperator `To` einsetzen. In der zweiten Abfrage haben Sie die Zahlenwerte durch Komma getrennt eingegeben. Wurde kein gültiger Zahlenwert eingegeben, kommt die Anweisung `Case Else` zum Tragen. Dort geben Sie eine Fehlermeldung auf dem Bildschirm aus.

Die folgende Tabelle enthält eine Liste der Vergleichsoperatoren und die Bedingungen, unter denen das Ergebnis `True`, `False` oder `0` wird:

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

Tabelle 3.2: Die Vergleichsoperatoren in Excel

Fensterstatus ermitteln

In der folgenden Aufgabe soll der Fensterstatus Ihrer Anwendung ermittelt werden. Dieser Status kann drei Werte aufweisen:

- Fenster normal
- Fenster maximiert
- Fenster minimiert

Überprüfen Sie nun den Fensterstatus Ihrer Anwendung, indem Sie das folgende Makro starten:

Listing 3.7: Fensterstatus der Anwendung ermitteln

```
Sub WindowStatusErmitteln()
Dim s As String

    Select Case Application.WindowState
        Case xlMaximized
            s = "Maximiert"
        Case xlMinimized
```

```
            s = "Minimiert"
        Case xlNormal
            s = "Normal"
    End Select

    MsgBox "Der augenblickliche Window-Status lautet: " & s
End Sub
```

Über die Eigenschaft `WindowState` können Sie den Status Ihres Anwendungsfensters feststellen. Werten Sie diesen anschließend über die Anweisung `Select Case` aus. Die Frage nach dem Fensterstatus wird beispielsweise dann interessant, wenn Sie Daten in einer Tabelle eingeben und dazu sicherstellen möchten, dass Sie die größtmögliche Ansicht haben.

3.3 Schleifen programmieren

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

Das war nun zugegeben etwas abstrakt: Stellen Sie sich vor, Sie müssten eine Flasche Wein (0,75 l) trinken. Dabei dürfen Sie den Wein nicht direkt aus der Flasche trinken, sondern müssen diesen in ein 0,25-l-Glas einschenken. Diesen Vorgang müssen Sie genau dreimal wiederholen, um die Flasche leer zu trinken. Dies ist ein typisches Beispiel für eine Schleife, die solange durchlaufen wird, bis die Endbedingung eintrifft. Die Geschwindigkeit bei Excel-Schleifen ist natürlich um ein 1000faches schneller als bei diesem Beispiel aus dem täglichen Leben.

Lernen Sie auf den nächsten Seiten die zur Verfügung stehenden Schleifen und einfache Beispiele für den Einsatz von Schleifen kennen.

For...Next-Schleifen

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

Die Syntax dieser Schleife lautet:

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

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

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

Mit dem Argument `Ende` legen Sie den Endwert von `Zähler` fest. Das Argument `Schritt` ist optional. Hier können Sie den Betrag bestimmen, um den Zähler bei jedem Schleifendurchlauf verändert wird. Falls kein Wert angegeben wird, ist die Voreinstellung 1.

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

Innerhalb einer Schleife kann eine beliebige Anzahl von `Exit For`-Anweisungen an beliebiger Stelle als alternative Möglichkeit zum Verlassen der Schleife verwendet werden.

Üben Sie diese Art von Schleife anhand der nächsten Aufgaben:

Zeilen abarbeiten

Im ersten Beispiel zur `For...Next`-Schleife sollen einige Zeilen abgearbeitet werden. Diese Aufgabe gehört zu den wichtigsten Aufgaben in Excel überhaupt. In vielen Fällen müssen Sie in einer Tabelle alle Datensätze nacheinander abarbeiten, wie z.B. Artikellisten, Preislisten oder Kundenlisten.

In der nächsten Aufgabe sollen die Werte aus Spalte A in Spalte B übertragen und um den Faktor 15% erhöht werden. Sehen Sie sich vorab einmal folgende Abbildung an:

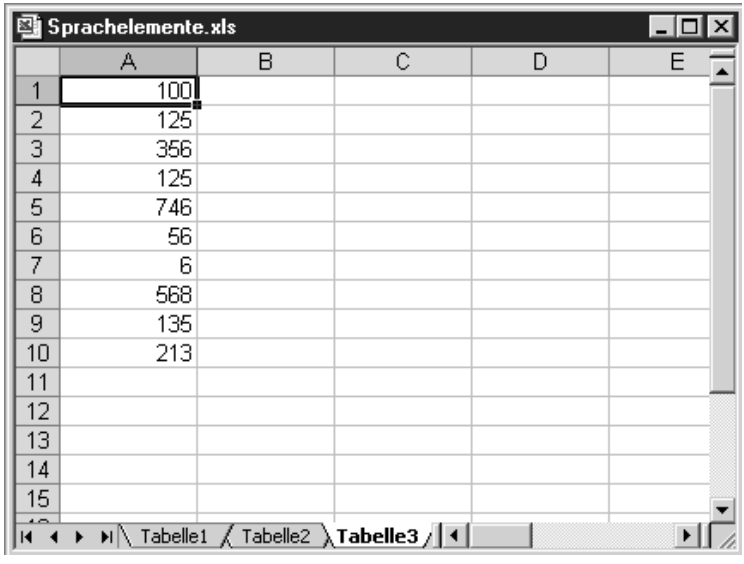

Abbildung 3.4: Eine Zahlenkolonne in Spalte A

Erfassen Sie nun das Makro, welches die Werte um 15% erhöht und in Spalte B schreibt:

Listing 3.8: Zahlen um einen bestimmten Prozentsatz erhöhen

```
Sub Schleife01()
Dim i As Long

Sheets("Tabelle3").Activate
Range("A1").Select

For i = 1 To 10
 ActiveCell.Offset(0, 1).Value = _
 ActiveCell.Value * 1.15
 ActiveCell.Offset(1, 0).Select
Next i

End Sub
```

Aktivieren Sie im ersten Schritt die Tabelle, auf der Sie die prozentuale Erhöhung durchführen möchten. Selektieren Sie danach die Startzelle, bei der die Aktion beginnen soll. Setzen Sie nun eine `For...Next`-Schleife auf, die genau zehnmal durchlaufen wird. Innerhalb der Schleife multiplizieren Sie den Inhalt der jeweils aktiven Zelle und fügen diesen Wert über die Eigenschaft `OffSet` in die Nebenzelle ein. Die Nebenzelle erreichen Sie, indem Sie das erste Argument von `OffSet`, die Zeilenverschiebung, auf den Wert `0` setzen, und das zweite Argument, die Spaltenverschiebung, auf den Wert `1`.

Vergessen Sie nicht, nach jedem Schleifendurchlauf den Mauszeiger über die Anweisung `ActiveCell.Offset(1, 0).Select` eine Zeile weiter nach unten zu setzen, da Sie sonst eine Endlosschleife produzieren.

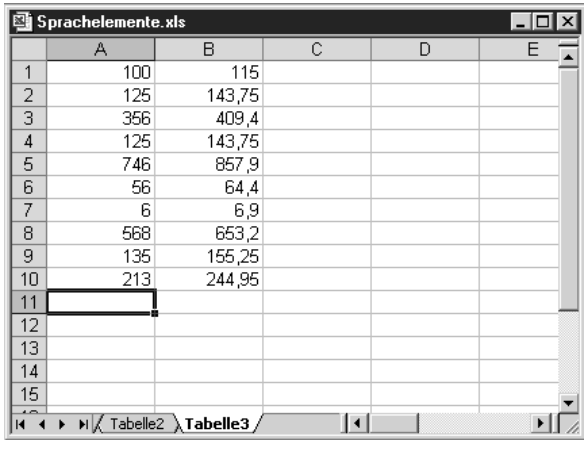

Abbildung 3.5:
Das Ergebnis der Operation sehen Sie in Spalte B

An dieser Stelle möchte ich Sie noch auf eine Schwachstelle in Listing 3.8 hinweisen. Diese Schleife ist leider nicht dynamisch, d.h. wenn Sie Ihre Tabelle erweitern, müssen Sie auch immer die Zählvariable `i` anpassen.

In diesem Fall ist es besser, zu ermitteln, wie viele Zeilen in der Tabelle benötigt werden und in Abhängigkeit davon die Schleife aufzubauen. Dies wurde im folgenden Makro in Listing 3.9 umgesetzt:

Listing 3.9: Dynamische Schleife

```
Sub Schleife01Dynamisch()
Dim i As Long

Sheets("Tabelle3").Activate
Range("A1").Select

For i = 1 To ActiveSheet.UsedRange.Rows.Count
 ActiveCell.Offset(0, 1).Value = _
 ActiveCell.Value * 1.15
 ActiveCell.Offset(1, 0).Select
Next i

End Sub
```

Mit der Anweisung `ActiveSheet.UsedRange.Rows.Count` ermitteln Sie die Anzahl der belegten Zeilen in Ihrer Tabelle. Dies bildet auch das Endkriterium für Ihre Schleife.

Zeilen einfärben

Vielleicht haben Sie schon einmal einen Blick in ein Rechenzentrum geworfen. Dort gibt es riesige Papierrollen, auf denen Papier auf die Drucker geleitet wird. Dieses Papier ist oft liniert, d.h. eine Zeile weiß und eine Zeile grün. Stellen Sie im folgenden Makro ein solches Papier her:

Listing 3.10: Liniertes Papier herstellen

```
Sub Schleife02()
Dim i As Long

Sheets("Tabelle4").Activate
Range("A1").Select

 For i = 1 To 20 Step 2
  ActiveCell.Offset(i, 0).EntireRow.Interior.ColorIndex = 36
 Next i
End Sub
```

Die Schrittweite in Listing 3.9 ist dieses Mal nicht 1, sondern 2. Dabei verweisen Sie mithilfe der Eigenschaft `OffSet`, bei der Sie das Zeilenargument über die Variable `i` angeben, dynamisch. Während der ganzen Verarbeitung bleibt der Mauszeiger immer auf Zelle A1. Sie verweisen lediglich immer auf die gewünschte Zelle. Über die Eigenschaft `EntireRow` gewinnen Sie Zugriff auf die komplette Zeile, die Sie über die Eigenschaft `ColorIndex` einfärben. Jeder Farbwert in Excel hat einen eindeutigen Farbindex. So hat die Farbe HELLGELB den Farbindex 36.

In Kapitel 8 erfahren Sie noch, wie Sie die einzelnen Farben und deren Farbindexe auslesen können.

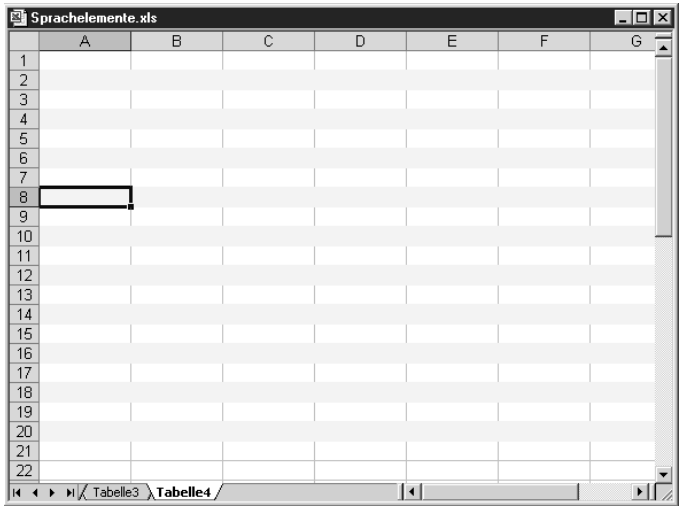

Abbildung 3.6: Das Papier im Rechenzentren-Look

Tabellennamen ermitteln

Im nächsten Beispiel sollen Sie die Namen der Tabellen ermitteln, die sich in der aktiven Arbeitsmappe befinden. Dies ist wichtig, damit Sie die einzelnen Tabellen aktivieren und ansprechen können. Das Makro für diese Aufgabe lautet:

Listing 3.11: Tabellennamen sammeln und ausgeben

```
Sub Schleife03()
Dim i As Integer
Dim s As String

For i = 1 To ThisWorkbook.Worksheets.Count
```

```
 s = s & Chr(13) & Worksheets(i).Name
Next i

MsgBox "In dieser Mappe befinden sich folgende Tabellen: " & _
Chr(13) & s, vbInformation + vbOKOnly
End Sub
```

Die Schleife in Listing 3.11 ist dynamisch. Über die `ThisWorkbook.Worksheets.Count`-Anweisung zählen Sie alle Tabellen, die sich in der Arbeitsmappe befinden. Diese Anzahl bauen Sie als Endkriterium in Ihre Schleife ein. Innerhalb der Schleife sammeln Sie die einzelnen Namen der Tabellen in einer `String`-Variablen. Die Namen der einzelnen Tabellen bekommen Sie über die Eigenschaft `Name`, welche Sie auf das jeweilige Tabellenblatt anwenden.

Geben Sie am Ende den Inhalt der Variablen `s` am Bildschirm aus.

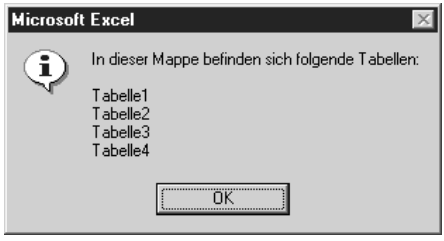

Abbildung 3.7:
Auflisten der Tabellennamen einer Mappe

For Each...Next-Schleifen

Die Schleife `For Each...Next` wiederholt eine Gruppe von Anweisungen für jedes Element in einem Datenfeld oder einer Auflistung.

Die Syntax dieser Schleife lautet:

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

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

Auch dieser wichtige Schleifentyp wird nun anhand praxisnaher Aufgaben geübt.

Zellen auswerten

Im ersten Beispiel für die `For Each...Next`-Schleife sollen Sie Zellen eines vorher definierten Bereichs auswerten. Alle Zellen in diesem Bereich, die einen Wert > 80 aufweisen, sollen mit dem Schriftschnitt FETT belegt werden. Sehen Sie sich vorher die Ausgangssituation in der folgenden Abbildung an:

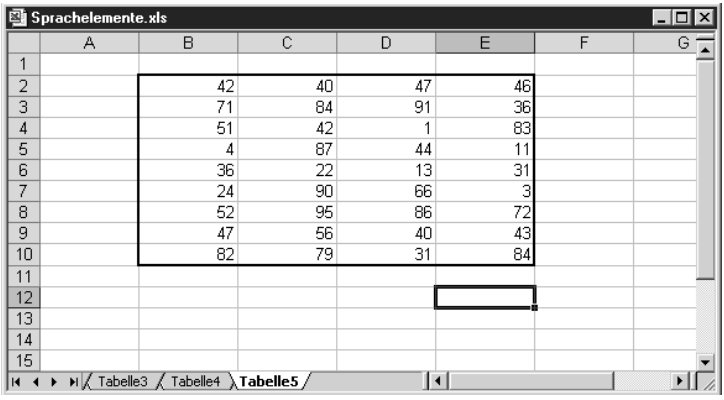

Abbildung 3.8: Die Ausgangslage – ein Bereich mit Zahlenwerten

Erfassen Sie jetzt das Makro, welches den Zellenbereich `B2:E10` untersucht und die Werte `> 80` mit dem Schriftschnitt FETT belegt:

Listing 3.12: Zellen in Zielbereich untersuchen

```
Sub Schleife04()
Dim Zelle As Range
Dim Bereich As Range

Set Bereich = Sheets("Tabelle5").Range("B2:E10")

For Each Zelle In Bereich
 If Zelle.Value > 80 Then Zelle.Font.Bold = True
Next Zelle
End Sub
```

Deklarieren Sie im ersten Schritt zwei Objektvariablen vom Typ `Range`. Die Objektvariable `Zelle` symbolisiert jeweils eine Zelle. Die Objektvariable `Bereich` steht für mehrere Zellen, also einen ganzen Zellenbereich. Diesen Zellenbereich geben Sie über die Anweisung `Set` bekannt. Geben Sie dabei sowohl den Namen der Tabelle als auch die korrekten Koordinaten des Bereichs an.

Danach durchlaufen Sie eine `For Each...Next`-Schleife, bei der alle Zellen (`Zelle`) des Zielbereichs (`Bereich`) durchlaufen werden. Innerhalb dieser Schleife prüfen Sie über eine

`If`-Abfrage, ob der jeweilige Zellenwert über 80 liegt. Wenn ja, dann wenden Sie die Eigenschaft `Bold` auf das Objekt `Font` an, um der Zelle den Schriftschnitt FETT zuzuweisen.

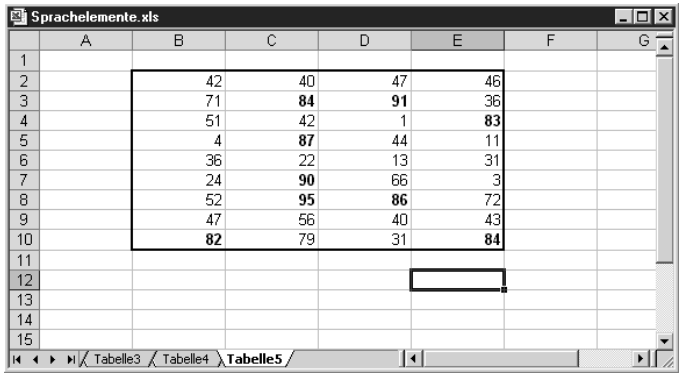

Abbildung 3.9:
Alle Zellenwerte > 80 wurden mit dem Schriftschnitt Fett formatiert

Einheitliche Kopf- und Fußzeilen

Im nächsten Beispiel werden Sie für eine einheitliche Gestaltung der Kopf- und Fußzeilen sorgen. Dabei arbeiten Sie alle Tabellen einer Arbeitsmappe ab, rufen die Seitenansicht auf und stellen die gewünschten Kopf- und Fußzeilen ein. Dabei werden folgende Informationen gewünscht:

- Kopfzeile links: der Firmenname
- Kopfzeile Mitte: der Name des Tabellenblattes
- Kopfzeile rechts: das aktuelle Datum
- Fußzeile links: der komplette Speicherpfad inkl. Dateinamen
- Fußzeile Mitte: bleibt leer
- Fußzeile rechts: Seitennummerierung

Erfassen Sie nun das Makro aus Listing 3.13:

Listing 3.13: Einheitliche Kopf- und Fußzeilen erzeugen

```
Sub Schleife05()
Dim Tabelle As Worksheet

    Application.ScreenUpdating = False

    For Each Tabelle In ActiveWorkbook.Worksheets
```

```
        With Tabelle.PageSetup
            .LeftHeader = "Firmennamen"
            .CenterHeader = "Tabellenname &A"
            .RightHeader = "&D"
            .LeftFooter = "Pfad : " & ActiveWorkbook.FullName
            .CenterFooter = ""
            .RightFooter = "Seite &P von &N"
        End With
    Next Tabelle

    Set Tabelle = Nothing
    Application.ScreenUpdating = True
End Sub
```

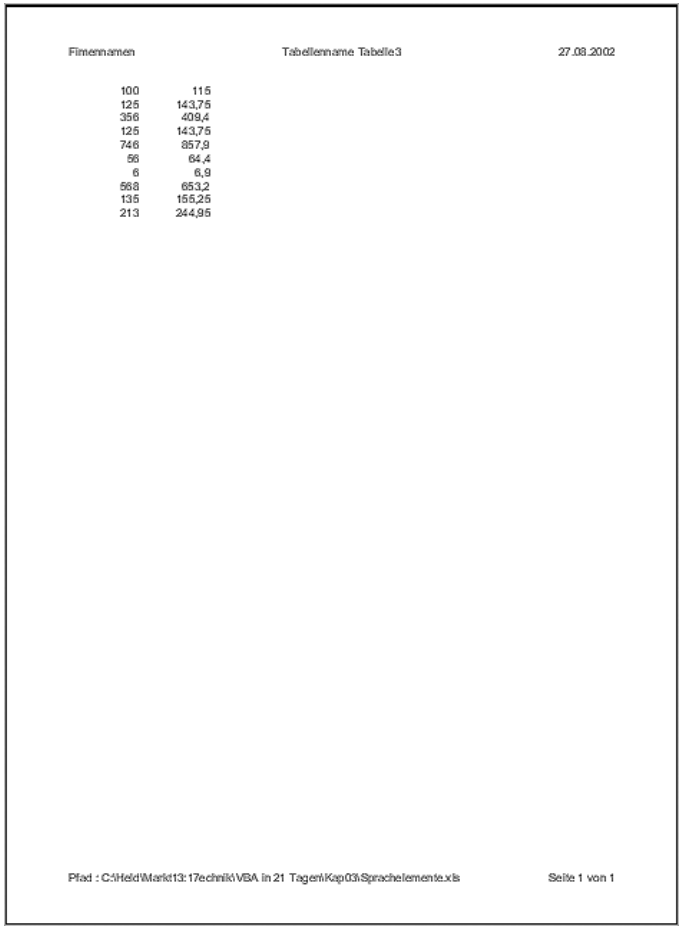

Abbildung 3.10:
Die Kopf- und Fußzeilen sind in allen Tabellen einheitlich.

Definieren Sie im ersten Schritt eine Objektvariable vom Typ `Worksheet`. Schalten Sie danach die Bildschirmaktualisierung aus. Dies tun Sie, indem Sie die Anweisung `Application.ScreenUpdating = False` einsetzen. Setzen Sie danach eine Schleife auf, die alle Tabellen der aktiven Arbeitsmappe durchläuft und über das Objekt `PageSetup` einheitliche Kopf- und Fußzeilen einstellt. Heben Sie am Ende des Makros den Objektverweis wieder auf, um Speicher freizugeben, und schalten Sie die Bildschirmaktualisierung wieder ein.

Excel-Arbeitsmappen zählen

Im letzten Beispiel zur `For Each...Next`-Schleife sollen die Namen aller Excel-Arbeitsmappen eines Verzeichnisses und der darunter liegenden Verzeichnisse ermittelt und ausgegeben werden. Der Code dafür lautet:

Listing 3.14: Alle Excel-Arbeitsmappen eines Verzeichnisses werden gezählt

```
Sub Schleife06()
Dim obj As Variant
Const verz = "C:\Eigene Dateien\"

 On Error GoTo fehler
 ChDir verz

    With Application.FileSearch
        .NewSearch
        .LookIn = verz
        .Filename = "*.xls"
        .SearchSubFolders = True

        If .Execute() > 0 Then
           For Each obj In .FoundFiles
             Debug.Print obj
           Next obj
        End If

        MsgBox .FoundFiles.Count
    End With
  Exit Sub

fehler:
MsgBox "Es gibt kein Verzeichnis mit dem Namen " & verz
End Sub
```

Zu Beginn des Makros können Sie das zu durchsuchende Verzeichnis in einer Konstanten angeben. Wechseln Sie über die Anweisung `ChDir` direkt in dieses Verzeichnis. Danach

starten Sie die Suche und verwenden dabei das Objekt `FileSearch`, um die einzelnen Dateien im Verzeichnis zu ermitteln. Auf dieses Objekt können Sie einige Eigenschaften anwenden:

Die Eigenschaft `NewSearch` setzt die Einstellungen aller Suchkriterien auf die Standardeinstellungen zurück.

Mithilfe der Eigenschaft `LookIn` geben Sie bekannt, in welchem Verzeichnis die Suche beginnen soll.

Die Eigenschaft `SearchSubFolders` bestimmt, ob die Suche auch in Unterverzeichnissen fortgesetzt werden soll. In diesem Fall müssen Sie diese Eigenschaft auf den Wert `True` setzen.

Die Eigenschaft `FileType` gibt den Typ der Datei in einer Konstanten an, nach der gesucht werden soll. Möchten Sie beispielsweise nicht nur Excel-Arbeitsmappen suchen lassen, sondern alle Dateitypen, dann geben Sie die Konstante `msoFileTypeAllFiles` an.

Über die Anweisung `FoundFiles.Count` geben Sie dann den Namen der gefundenen Datei an und schreiben diesen in den Direktbereich Ihrer Entwicklungsumgebung.

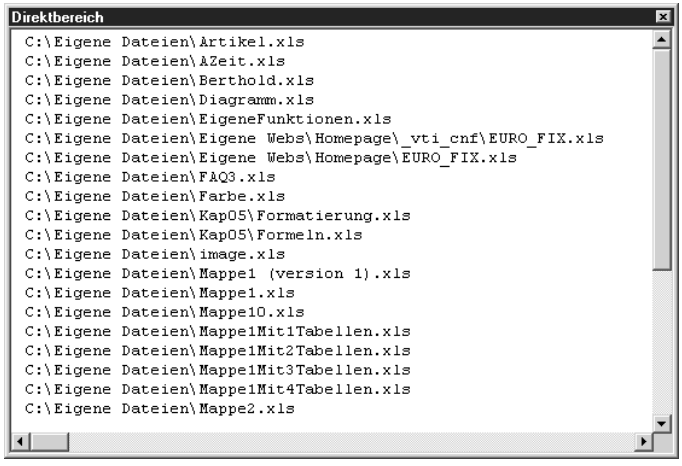

Abbildung 3.11: Die Namen der Excel-Arbeitsmappen werden im Direktbereich ausgegeben.

Den Direktbereich können Sie einblenden, indem Sie in Ihre Entwicklungsumgebung wechseln und aus dem Menü ANSICHT den Befehl DIREKTFENSTER wählen.

Die Schleife Do Until...Loop

Die `Do Until...Loop`-Schleife wiederholt einen Block mit Anweisungen, solange eine Bedingung den Wert `True` erhält. Die Bedingung wird jeweils am Ende der Schleife geprüft. Als Abbruchbedingung können Sie alles Mögliche abfragen; so können Sie z.B. eine Abbruchbedingung festlegen, wenn ein bestimmter Wert erreicht ist oder eine Zelle einen bestimmten Text aufweist. Beispielsweise könnten Sie eine solche Schleife so oft wiederholen, solange die Zellenformatierung der Zellen sich nicht ändert.

Die Syntax dieser Schleife sieht wie folgt aus:

```
Do [{Until} Bedingung]
 [Anweisungen]
 [Exit Do]
 [Anweisungen]
Loop
```

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

Innerhalb einer `Do Until...Loop`-Anweisung kann eine beliebige Anzahl von `Exit Do`-Anweisungen an beliebiger Stelle als Alternative zum Verlassen einer `Do Until...Loop`-Anweisung verwendet werden. `Exit Do` wird oft in Zusammenhang mit der Auswertung einer Bedingung (zum Beispiel `If...Then`) eingesetzt und hat zur Folge, dass die Ausführung mit der ersten Anweisung im Anschluss an `Loop` fortgesetzt wird.

Üben Sie den Einsatz dieser Schleife mittels einiger ausgesuchter Beispiele.

Jede x-te Zeile löschen

Im ersten Beispiel für die `Do Until...Loop`-Schleife werden Sie in einer Tabelle jede dritte Zeile löschen. Dabei gehen Sie von der in Abbildung 3.12 dargestellten Ausgangstabelle aus.

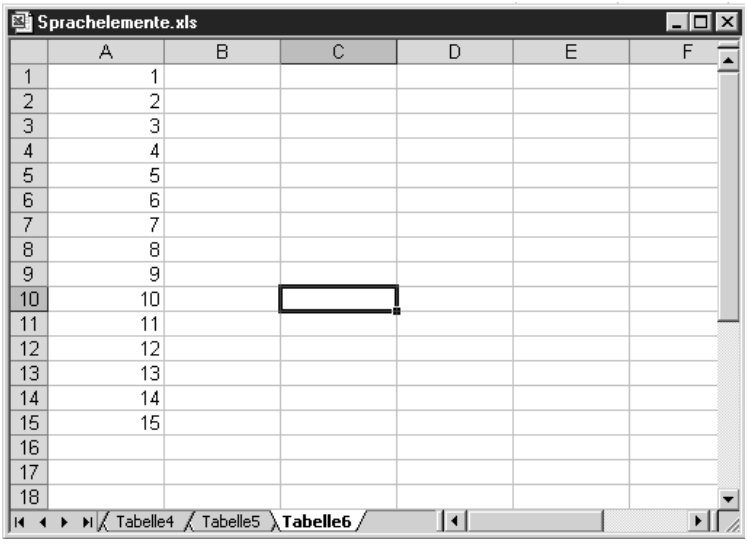

Abbildung 3.12: Die Ausgangstabelle

Erfassen Sie nun das Makro, welches jede dritte Zeile löscht:

Listing 3.15: Jede dritte Zeile löschen

```
Sub Schleife07()
Const zz = 2
Dim i As Integer

Sheets("Tabelle6").Activate
Range("A1").Select
i = 1

Do Until ActiveCell.Value = ""
    For i = 1 To zz
      ActiveCell.Offset(1, 0).Select
    Next i
    Selection.EntireRow.Delete
    i = 1
Loop
End Sub
```

Durchlaufen Sie in einer `Do Until...Loop`-Schleife alle gefüllten Zellen der Spalte A. Als Endkriterium für die Schleife gilt die erste leere Zelle, auf die Excel stößt. Innerhalb dieser Schleife können Sie mit einer weiteren Schleife arbeiten, welche die Zeilen überspringt, die nicht gelöscht werden sollen. Über die Methode `Delete` löschen Sie jeweils jede dritte Zeile.

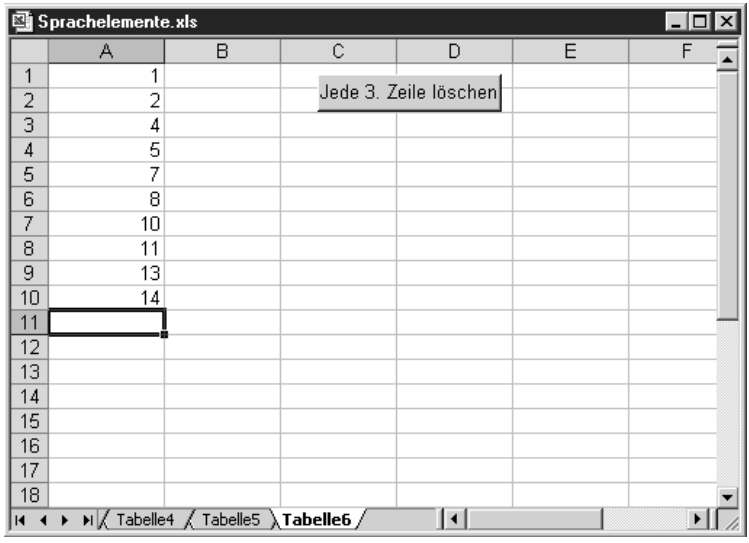

Abbildung 3.13: Jede dritte Zeile wurde gelöscht.

Bestimmte Zeichen austauschen

Im nächsten Beispiel werden in einer Tabelle bestimmte Zeichen ausgetauscht. So sollen alle Bindestriche durch Slashes ersetzt werden. Sehen Sie sich vorher die Ausgangssituation für diese Aufgabenstellung an.

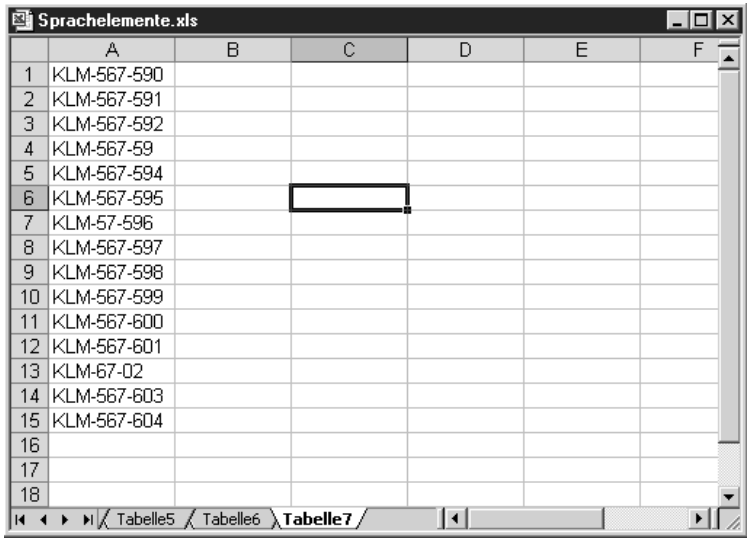

Abbildung 3.14: Die Bindestriche sollen durch das Zeichen Slash ersetzt werden.

Erfassen Sie dafür das folgende Makro:

Listing 3.16: Alle Bindestriche wurden ersetzt

```
Sub Schleife08()
Sheets("Tabelle7").Activate
Range("A1").Select

Do Until ActiveCell.Value = ""
 If InStr(ActiveCell.Value, "-") Then
    ActiveCell.Value = _
    Application.Substitute(ActiveCell.Value, "-", "/")
 End If
ActiveCell.Offset(1, 0).Select
Loop
End Sub
```

Setzen Sie eine `Do Until...Loop`-Schleife ein, um alle Zellen der Spalte A abzuarbeiten. Innerhalb der Schleife prüfen Sie mithilfe der Funktion `InStr`, ob ein Bindestrich in der jeweiligen Zelle überhaupt vorkommt. Wenn ja, wenden Sie die Funktion `Substitute` an, um dieses Zeichen durch ein anderes auszutauschen. Vergessen Sie nicht, den Mauszeiger bei jedem Schleifendurchlauf über die Eigenschaft `Offset` eine Zeile weiter nach unten zu setzen.

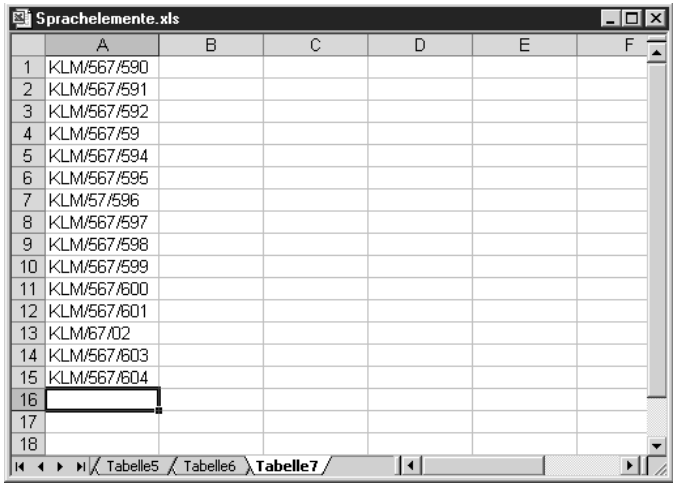

Abbildung 3.15: Alle Bindestriche wurden entfernt.

Die Schleife Do While...Loop

Die `Do While...Loop`-Schleife wiederholt einen Block mit Anweisungen, solange eine Bedingung den Wert `True` erhält. Die Prüfung der angegebenen Bedingung erfolgt immer zu Beginn der Schleife. Als Abbruchbedingung können Sie alles Mögliche abfragen; so können Sie z.B. eine Abbruchbedingung festlegen, wenn ein bestimmter Wert erreicht ist oder eine Zelle einen bestimmten Text aufweist.

Die Syntax dieser Schleife sieht wie folgt aus:

```
Do [{While} Bedingung]
 [Anweisungen]
 [Exit Do]
 [Anweisungen]
Loop
```

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

Innerhalb einer `Do While...Loop`-Anweisung kann eine beliebige Anzahl von `Exit Do`-Anweisungen an beliebiger Stelle als Alternative zum Verlassen einer `Do...Loop`-Anweisung verwendet werden. `Exit Do` wird oft in Zusammenhang mit der Auswertung einer Bedingung (zum Beispiel `If...Then`) eingesetzt und hat zur Folge, dass die Ausführung mit der ersten Anweisung im Anschluss an `Loop` fortgesetzt wird. Üben Sie diese Art von Schleife:

Zellenlänge überprüfen

Im ersten Beispiel zur Schleife `Do While...Loop` überprüfen Sie in einer Tabelle, ob in Spalte A gültige Eingaben vorgenommen wurden. So darf jeder Eintrag nur 8 Zeichen lang sein. Sehen Sie zur besseren Verständlichkeit die folgende Abbildung an:

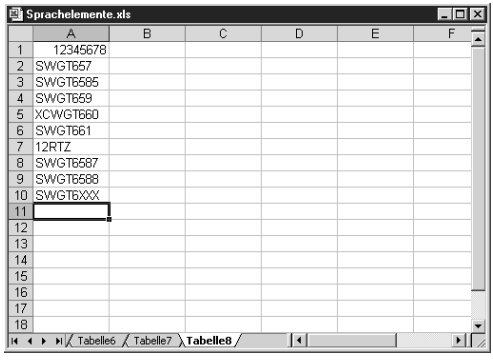

Abbildung 3.16: Welche Eingaben sind korrekt?

Schreiben Sie jetzt das folgende Makro, welches die Längen der Zellen in Spalte A überprüft. Wird eine falsche Länge gefunden, wird ein Eintrag in Spalte B vorgenommen.

Listing 3.17: Länge von Eingaben messen und reagieren

```
Sub Schleife09()
Sheets("Tabelle8").Activate
Range("A1").Select

 Do While ActiveCell.Value <> ""
   If Len(ActiveCell.Value) <> 8 Then _
      ActiveCell.Offset(0, 1).Value = "Falsche Länge!"
   ActiveCell.Offset(1, 0).Select
 Loop
End Sub
```

Durchlaufen Sie über die Schleife `Do While...Loop` alle Zellen der Spalte A, solange Excel auf keine leere Zelle stößt. Innerhalb der Schleife prüfen Sie über die Funktion `Len`, ob die richtige Länge der Zelle, nämlich 8, vorliegt. Wenn nicht, dann schreiben Sie in die Nebenzelle in Spalte B einen Vermerk.

Abbildung 3.17: Falsche Eingaben wurden gekennzeichnet.

Bestimmte Zahlen summieren

Im folgenden Beispiel sollen in einer Tabelle mit Zahlen nur diejenigen summiert werden, die mit der Schriftfarbe ROT und dem Schriftschnitt FETT formatiert sind. Das Makro für diese Aufgabe lautet:

Listing 3.18: Alle roten und fetten Zahlen sollen summiert werden

```
Sub Schleife10()
Dim Betrag As Single

Sheets("Tabelle9").Activate
Range("A1").Select

 Do While ActiveCell.Value <> ""
   If ActiveCell.Font.ColorIndex = 3 And _
      ActiveCell.Font.Bold = True And _
      IsNumeric(ActiveCell) Then _
      Betrag = Betrag + ActiveCell.Value
      ActiveCell.Offset(1, 0).Select
 Loop
MsgBox "Die Summe der roten Zahlen lautet: " & Betrag
End Sub
```

In einer `Do While...Loop`-Schleife durchlaufen Sie alle gefüllten Zellen der Spalte A. Überprüfen Sie dabei für jede Zelle deren Formatierung. Nur numerische Zellen, die mit der Schriftfarbe ROT und dem Schriftschnitt FETT formatiert sind, sollen summiert werden.

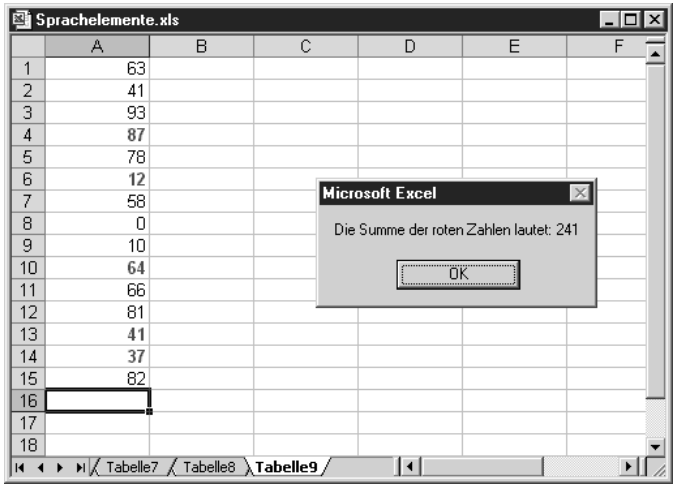

Abbildung 3.18:
Alle roten, fetten Zahlen wurden summiert

3.4 Workshop

3.4.1 Fragen & Antworten

F *Wie können Sie überprüfen, welche Arbeitsmappen gerade geöffnet sind?*

A Diese Fragestellung können Sie über eine `For Each...Next`-Schleife beantworten. Im folgenden Makro werden die Namen der geöffneten Arbeitsmappen auf dem Bildschirm ausgegeben.

Listing 3.19: Alle geöffneten Arbeitsmappen ausgeben

```
Sub MappenNamenErmitteln()
Dim Mappe As Workbook

For Each Mappe In Workbooks
 MsgBox Mappe.Name
Next Mappe
End Sub
```

Deklarieren Sie im ersten Schritt eine Objektvariable vom Typ `Workbook`. Danach greifen Sie in einer Schleife auf das Auflistungsobjekt `Workbooks` zu. In diesem Auflistungsobjekt sind alle geöffneten Arbeitsmappen verzeichnet, die Sie über die Eigenschaft `Name` auslesen und über die Funktion `MsgBox` auf dem Bildschirm anzeigen können.

F *Wie können Sie mithilfe einer Schleife alle installierten, aktivierten Add-Ins ermitteln?*

A Erfassen Sie das Makro aus Listing 3.20:

Listing 3.20: Alle aktivierten Add-Ins ausgeben

```
Sub AddInsAusgeben()
Dim ADDI As AddIn

    For Each ADDI In Application.AddIns
        If ADDI.Installed = True Then MsgBox ADDI.Name
    Next
End Sub
```

Deklarieren Sie zu Beginn des Makros eine Objektvariable vom Typ `AddIn`. Danach greifen Sie in einer Schleife auf die `AddIns`-Auflistung zu und ermitteln die Namen der installierten Add-Ins über die Eigenschaft `Name`.

3.4.2 Quiz

1. *Über welche Funktion kann man prüfen, ob eine Zelle ein Datum enthält?*
2. *Wie kann man aus einer* `For Each...Next`*-Schleife springen?*

3.4.3 Übung

Zum Abschluss dieses Tages üben Sie das heute Gelernte. Dabei soll folgende Aufgabe gelöst werden:

Die Namen der Tabellen einer Arbeitsmappe sollen auf der ersten Tabelle dieser Arbeitsmappe in Spalte A eingefügt werden. Lösen Sie diese Aufgabenstellung mithilfe einer `For Each...Next`-Schleife.

Die Demodatei *Sprschelemente.xls* finden Sie auf der CD-ROM im Verzeichnis *Kap03*. Eine Musterlösung finden Sie im Anhang.