3 Die wichtigsten Sprachelemente in Access

Das Wesentliche, was eine Programmiersprache ausmacht, sind ihre Sprachelemente. In diesem Kapitel erfahren Sie, wie Sie mithilfe von Abfragen, Schleifen und anderen Anweisungen Ihre Programme flexibel gestalten können. Diese Sprachelemente lassen sich nicht mit dem Makrorekorder aufzeichnen und müssen von Ihnen selbst erstellt werden. Der richtige Einsatz der Sprachelemente macht letztendlich die Kunst der Programmierung aus. Des Weiteren finden Sie in diesem Kapitel eine Auswahl der wichtigsten Funktionen von VBA sowie eine Beschreibung ihres Einsatzes.

Die Themen dieses Kapitels

Unter anderem geht es in diesem Kapitel um folgende Fragestellungen:

- Wie arbeite ich mit Verzweigungen?
- Wie kann ich die verzwickten Verzweigungen übersichtlicher darstellen?
- Wie programmiere ich Schleifen mit VBA?
- Wie kann ich Standard-VBA-Funktionen in meinen Modulen einsetzen?
- Wie setze ich Arrays in der Programmierung ein?
- Wie arbeite ich mit Operatoren?
- Wie schreibe ich meine eigenen Funktionen?

Sie finden alle Beispiele in diesem Kapitel auf der diesem Buch beiliegenden CD-ROM im Verzeichnis KAP03 *in der Datei* SPRACHELEMENTE.MDB.

3.1 Verzweigungen

Die Syntax für Verzweigungen

Mit Verzweigungen können Sie in Access bestimmte Zustände abfragen und je nach Zustand anders reagieren. Die 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` müssen Sie entweder einen numerischen Ausdruck oder einen Zeichenfolgenausdruck eingeben, der `True` oder `False` ergibt. Wenn die Bedingung den Wert `0` zurückmeldet, wird Bedingung als `False` interpretiert.

Unter dem Argument `Anweisungen` werden jene Anweisungen aufgeführt, die durchgeführt werden sollen, wenn `Bedingung` den Wert `True` liefert.

Unter dem Argument `elseifAnweisungen` sind eine oder mehrere Anweisungen gemeint, die ausgeführt werden, wenn die zugehörige `Bedingung` (bzw. `Bedingung-n`) den Wert `True` meldet.

Das Argument `elseAnweisungen` meint eine oder mehrere Anweisungen, die ausgeführt werden sollen, wenn keine der Bedingungen (`Bedingung`-Ausdruck oder `Bedingung-n`-Ausdruck) den Wert `True` meldet.

:-) TIPP

Verwenden Sie die erste Variante, wenn Sie kurze Abfragen durchführen, beispielsweise einen Datentyp kontrollieren. Die Blockvariante ist dann von Interesse, wenn Sie mehrere Aktionen in einem Zweig durchführen möchten. Dies erhöht die Übersichtlichkeit des Quellcodes.

Üben Sie nun diese Struktur anhand der folgenden Beispiele.

Eingaben auswerten

Im folgenden Beispiel überprüfen Sie die Eingabe eines Anwenders mithilfe einer Verzweigung. Sehen Sie sich dazu das Listing 3.1 an.

Listing 3.1: Eingaben abfragen mit Verzweigung

```
Sub InputboxAuswerten()
Dim s As String
s = InputBox _
("Geben Sie hier entweder Ja oder Nein ein!", _
 "Eingabe")
If s = "" Then Exit Sub
 If s = "Ja" Then
 MsgBox "Sie haben Ja eingegeben!"
 ElseIf s = "Nein" Then
  MsgBox "Sie haben Nein eingegeben!"
 Else
  MsgBox "Sie haben keinen gültigen Wert eingegeben!"
 End If
End Sub
```

Mithilfe der Funktion `InputBox` können Sie einen Dialog auf dem Bildschirm anzeigen, bei dem der Anwender die Möglichkeit hat, eine Eingabe zu machen. Das Ergebnis dieser Eingabe speichern Sie in der Variablen `s`. Prüfen Sie in der ersten Schleife (Variante 1), ob überhaupt ein Wert eingegeben wurde. Wenn nicht, dann beenden Sie das Makro, indem Sie die Anweisung `Exit Sub` einsetzen. Im anderen Fall läuft das Makro weiter und gelangt zur zweiten Verzweigung (Variante 2). Dort prüfen Sie, welche Eingabe erfolgt ist. Wenn die Eingabe JA vorgenommen wurde, geben Sie eine dem entsprechende Meldung über die Funktion `MsgBox` auf dem Bildschirm aus. Anderenfalls müssen Sie noch prüfen, ob der Wert NEIN oder gar ein anderer Wert eingegeben wurde. Setzen Sie dafür die `ElseIf`-Bedingung ein und fragen die Eingabe erneut ab.

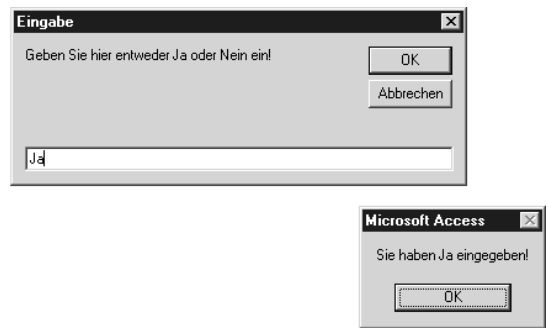


Abbildung 3.1: Eingabe und Auswertung

Eingaben prüfen und wandeln

Im zweiten Beispiel werden Sie überprüfen, ob ein Anwender einen Text oder einen numerischen Wert eingibt. Je nach Ergebnis werden Sie entsprechend reagieren. Sehen Sie sich dazu einmal das Listing 3.2 an.

Listing 3.2: Numerisch oder alphanumerisch?

```
Sub EingabenWandeln()
Dim s As String

s = InputBox _
("Geben Sie hier einen Text ein!", "Eingabe")
If s = "" Then Exit Sub
If Not IsNumeric(s) Then s = UCase(s) Else MsgBox _
 "Sie haben einen numerischen Wert eingegeben!"
MsgBox s
End Sub
```

Nützen Sie die Funktion `InputBox`, um einen Abfragedialog am Bildschirm anzuzeigen. Kontrollieren Sie danach wiederum, ob überhaupt eine Eingabe vorgenommen wurde. Wenn ja, dann prüfen Sie mithilfe der Funktion `IsNumeric`, ob ein numerischer Wert eingegeben wurde. Indem Sie das Wort `Not` vor diese Funktion setzen, prüfen Sie, ob ein Text eingegeben wurde. Wenn ja, dann wandeln Sie den eingegebenen Text über die Funktion `UCase` in Großbuchstaben um.

Abbildung 3.2: In Großbuchstaben konvertieren

:-) TIPP

Selbstverständlich haben Sie auch die Möglichkeit, einen Text in Kleinbuchstaben zu wandeln. Die dazu notwendige Funktion heißt `LCase`.

Sehr oft werden auch Datumsüberprüfungen in Access durchgeführt. Insbesondere wenn Berechnungen wie Liefertermine oder Zahlungsziele errechnet werden, müssen Sie als Entwickler sicherstellen, dass auch wirklich Datumseingaben vorgenommen wurden. Kontrollieren Sie dieses sofort nach der Eingabe, indem Sie das Listing 3.3 einsetzen.

Listing 3.3: Wurde ein gültiges Datum eingegeben?

```
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 01.01.02 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.

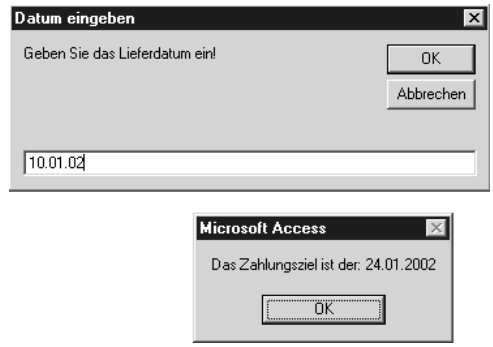


Abbildung 3.3: Aus einem Liefertermin wurde ein Zahlungsziel errechnet

Neben der Funktion `IsNumeric` *gibt es weitere Funktionen, mit denen Sie Ihre Daten prüfen können. Eine davon ist die Funktion* `IsDate`*. Die Funktion* `IsDate` *gibt den Wert* `True` *zurück, wenn der Ausdruck ein Datum ist oder in ein gültiges Datum umgewandelt werden kann. Andernfalls wird der Wert* `False` *zurückgegeben. In Windows liegen gültige Datumswerte im Bereich vom 1. Januar 100 n. Chr. bis 31. Dezember 9999 n. Chr. vor. Auf anderen Betriebssystemen können andere Bereiche gelten.*

INFO

Exkurs

Fehler abfangen

Wenn Sie sich noch einmal das Listing 3.3 ansehen, so finden Sie dort zu Beginn des Listings 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 Sprungmarke `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.

Tabelle 3.1: Die wichtigsten Prüffunktionen von 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

Eine Besonderheit

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

***Die Syntax der Funktion* `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.

Machen Sie sich die Wirkungsweise an folgendem Beispiel deutlich:

Listing 3.4: Erleichterung oder nicht?

```
Function Prüfen(Test1 As Integer)
  Prüfen = IIf(Test1 > 1000, "Groß", "Klein")
End Function
```

```
Sub Übergabe()
 MsgBox "Ihre Eingabe war: " & Prüfen(100)
End Sub
```

Über die Funktion `Prüfen` können Sie mit der Funktion `IIF` kontrollieren, ob ein übergebener Wert in einem bestimmten Wertbereich liegt. Die Funktion `Prüfen` rufen Sie auf, indem Sie in Ihrer Prozedur den Namen der Funktion angeben und dieser einen Wert mitgeben. In der Funktion `Prüfen` wird dieser übergebende Wert dann untersucht und an die aufrufende Prozedur zurückgemeldet.

3.2 Die Anweisung Select Case für mehr Übersicht

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 können. Im Argument `Ausdrucksliste-n` spezifizieren Sie den untersuchenden Ausdruck näher. Dabei können Sie Vergleichsoperatoren verwenden. So stehen Ihnen Vergleichsoperatoren wie `To`, `Is` oder `Like` zur Verfügung.

Unter dem Argument `Anweisungen-n` können Sie eine oder mehrere Anweisungen angeben, die ausgeführt werden sollen, wenn `Testausdruck` mit irgendeinem 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`.

Zahlenwerte prüfen mit Select Case

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 3.5: Zahlenwerte mit `Select Case` prüfen

```
Sub MehrfachAuswertung()
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, dann schlägt die Anweisung `Case Else` zu. 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` sein wird:

Tabelle 3.2: Die Vergleichsoperatoren in Access

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

Textwerte prüfen mit Select Case

Im letzten Beispiel aus Listing 3.5 haben Sie Zahlenwerte überprüft. Dasselbe funktioniert natürlich auch bei Texten. Im folgenden vereinfachten Beispiel werden anhand von eingegebenen Ortsnamen die dazugehörigen Postleitzahlen ermittelt.

Listing 3.6: Textwerte mit `Select Case` prüfen

```
Sub OrteInPLZ()
Dim s As String
Dim s_plz As String

s = InputBox _
("Geben Sie bitte einen Ortsnamen ein!")

If s = "" Then Exit Sub
Select Case s
 Case "Gerlingen"
  s_plz = "70839"
 Case "Stuttgart"
 s_plz = "70469, 70468, 70467"
 Case "Tübingen"
 s_plz = "72076"
 Case "Leinfelden"
 s_plz = "70771"
 Case "Sindelfingen"
 s_plz = "72076"
 Case Else
 MsgBox "Die Stadt " & s & " ist noch nicht erfasst!"
 Exit Sub
End Select

MsgBox "Für die Stadt " & s & _
    " sind folgende PLZ möglich: " & s_plz
End Sub
```

Die Abfragetechnik bei Texten funktioniert fast identisch mit jener bei den Zahlenwerten. Denken Sie jedoch daran, dass Sie die Texte in doppelte Anführungszeichen setzen.

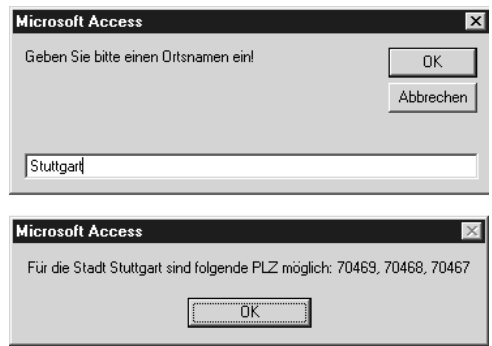


Abbildung 3.4: Ortsnamen in Postleitzahlen umwandeln

Das Beispiel aus Listing 3.6 ist natürlich jederzeit erweiterbar. Der Ablauf der Abfragen ist dabei wie folgt: Die Überprüfung erfolgt von oben nach unten. Sobald der Suchbegriff, in unserem Beispiel die richtige Stadt, gefunden wird, wird die dazugehörige Postleitzahl ermittelt und ans Ende der Anweisung gesprungen.

3.3 Schleifen in Access einsetzen

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

Die Syntax der For...Next-Schleife

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

Das Argument `Zähler` ist erforderlich und besteht aus einer numerischen Variable, 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 `eins`.

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 Möglichkeiten zum Verlassen der Schleife verwendet werden.

Im nächsten Beispiel soll eine Schleife genau 20-mal durchlaufen werden und die Zählerstände in den Direktbereich Ihrer Tastumgebung schreiben. Der Code hierfür lautet:

```
Sub For_Schleife()
Dim i As Integer

 For i = 1 To 20
 Debug.Print "Zählerstand: " & i
 Next i
End Sub
```

Listing 3.7: Die `For...Next`-Schleife schreibt Zahlen in den Direktbereich

Die Schleife wird genau 20-mal durchlaufen. Innerhalb der Schleife werden die Zählerstände über die Anweisung `Debug.Print` in das Direkt-Fenster geschrieben. Am Ende jeder Schleife wird der interne Zähler `i` um den Wert `1` erhöht.

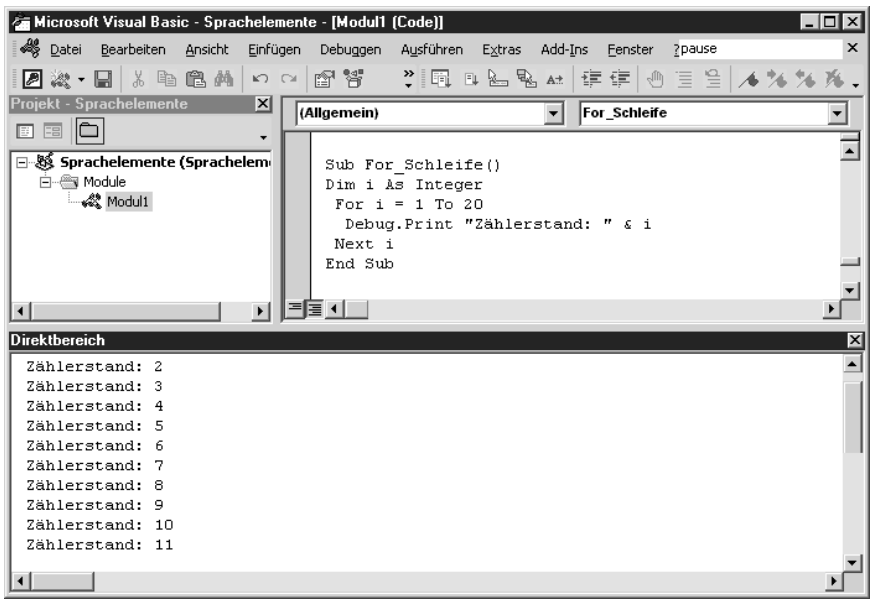


Abbildung 3.5: Die Zählerstände werden im Direktbereich ausgegeben

Wenn Sie eine andere Schrittweite haben möchten, dann ändern Sie das Makro aus Listing 3.7 wie folgt ab.

Listing 3.8: Die `For...Next`-Schleife mit veränderter Schrittweite

```
Sub For_Schleife02()
Dim i As Integer

 For i = 1 To 20 Step 5
 Debug.Print "Zählerstand: " & i
 Next i
End Sub
```

Möchten Sie Ihre Schleife noch übersichtlicher machen, dann können Sie dies mit sprechenden Variablen tun. Sehen Sie sich dazu das folgende Listing 3.9 an:

Listing 3.9: Eine `For...Next`-Schleife mit übersichtlichem Ende

```
Sub For_Schleife03()
Dim i As Integer
Dim maxWert As Integer

maxWert = 20

 For i = 1 To maxWert
 Debug.Print "Zählerstand: " & i
 Next i
End Sub
```

Die Vorgehensweise in Listing 3.9 ist zu empfehlen, da sie gerade bei größeren Makros übersichtlicher ist. Dabei setzen Sie den `maxWert` gleich zu Beginn des Makros auf den Höchstwert. Wenn Sie diesen Wert später ändern möchten, müssen Sie nicht das ganze Makro nach der richtigen Stelle durchsuchen. Außerdem haben Sie dabei auch die Möglichkeit, diese Variable für mehrere Schleifen zu verwenden. Die Änderung erfolgt in diesem Fall immer nur an einer Stelle, nämlich am Anfang des Makros.

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 der `For Each..Next`-Schleife

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 ist für `Element` nur eine Variable vom Typ `Variant`, eine allgemeine 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.

Im nachfolgenden Listing 3.10 werden alle Namen von Formularen in einer Access-Datenbank ausgegeben.

```
Sub NamenAllerFormulareAuslesen()
Dim obj As Form

For Each obj In Forms
 MsgBox obj.Name
Next obj
End Sub
```

Listing 3.10: Die `For Each... Next`-Schleife gibt die Namen aller Formulare aus

Bei dem Befehl `Forms` handelt es sich um ein so genanntes Auflistungsobjekt. In diesem Auflistungsobjekt sind alle momentan geöffneten Formulare einer Access-Datenbank verzeichnet. Diese können Sie über die Eigenschaft `Name` ermitteln und mit der Funktion `MsgBox` auf dem Bildschirm ausgeben.

Gehen Sie jetzt noch einen Schritt weiter und prüfen Sie, welches Formular in Ihrer Access-Datenbank geladen ist. Setzen Sie dazu das Listing 3.11 ein.

```
Sub IstFormularGeladen()
Dim obj As AccessObject
Dim dbs As Object

  Set dbs = Application.CurrentProject

  For Each obj In dbs.AllForms
    If obj.IsLoaded = True Then
      Debug.Print obj.Name
    End If
  Next obj
End Sub
```

Listing 3.11: Eine `For Each...Next`-Schleife zum Prüfen der geladenen Formulare

Deklarieren Sie zu Beginn Ihres Makros die verwendeten Variablen. Setzen Sie danach einen Objektverweis auf das Objekt `CurrentProject`. Dieses Objekt verfügt über mehrere Auflistungen, die bestimmte Objekte in der Access-Datenbank enthalten, unter anderem auch das Auflistungsobjekt `AllForms`. Entnehmen Sie die weiteren möglichen Auflistungsobjekte bitte der folgenden Tabelle:

Tabelle 3.3: Die Auflistungsobjekte des Objekts `CurrentProject`

Auflistungsobjekt	Bedeutung
`AllForms`	Enthält alle Formulare
`AllReports`	Enthält alle Berichte
`AllMacros`	Enthält alle Makros
`AllModules`	Enthält alle Module
`AllDataAccessPages`	Enthält alle Datenzugriffsseiten

Nachdem Sie den Objektverweis mithilfe der Anweisung `Set` eingesetzt haben, können Sie die Schleife beginnen. In der aktuellen Datenbank werden nun alle Formulare geprüft. Mit der Eigenschaft `IsLoaded` prüfen Sie, ob die jeweiligen Formulare geladen sind. Wenn ja, dann geben Sie die Namen der Formulare im Direktbereich Ihrer Entwicklungsumgebung aus.

Setzen Sie jetzt das Auflistungsobjekt `AllModules` ein, um alle Module eines VBA-Projektes zu ermitteln. Verwenden Sie dazu wiederum die `For Each...Next`-Schleife.

Listing 3.12: Eine `For Each...Next`-Schleife zum Auflisten aller Module

```
Sub AllModuleAuflisten()
Dim obj As AccessObject
Dim dbs As Object

Set dbs = Application.CurrentProject
 For Each obj In dbs.AllModules
   Debug.Print obj.Name
 Next obj
End Sub
```

Im folgenden Beispiel zur `For Each...Next`-Schleife werden Sie alle Steuerelemente auf einem bestimmten Formular ermitteln und in den Direktbereich schreiben. Der dazugehörige Code sieht wie folgt aus:

Listing 3.13: Eine `For Each...Next`-Schleife zum Ermitteln aller Steuerelemente eines Formulars

```
Sub SteuerelementeInFormAusgeben()
Dim obj As Control

For Each obj In Form_Personal.Form
 Debug.Print obj.Name
Next obj
End Sub
```

Definieren Sie im ersten Schritt eine Variable vom Typ `Control`. Dabei sprechen Sie ganz allgemein ein Steuerelement an, das sich z.B. auf einem Formular oder einem Bericht befinden kann. Setzen Sie danach die `For Each...Next`-Schleife auf und durchstreifen Ihr Formular. Über die Anweisung `Debug.Print` geben Sie die Namen der Steuerelemente aus, die sich in Ihrem Formular befinden.

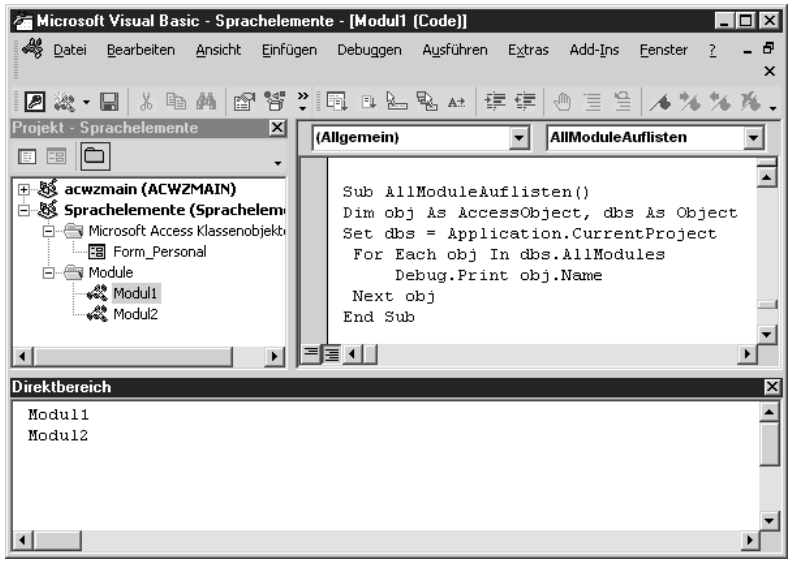


Abbildung 3.6: Alle Module im Direkt-Fenster ausgeben

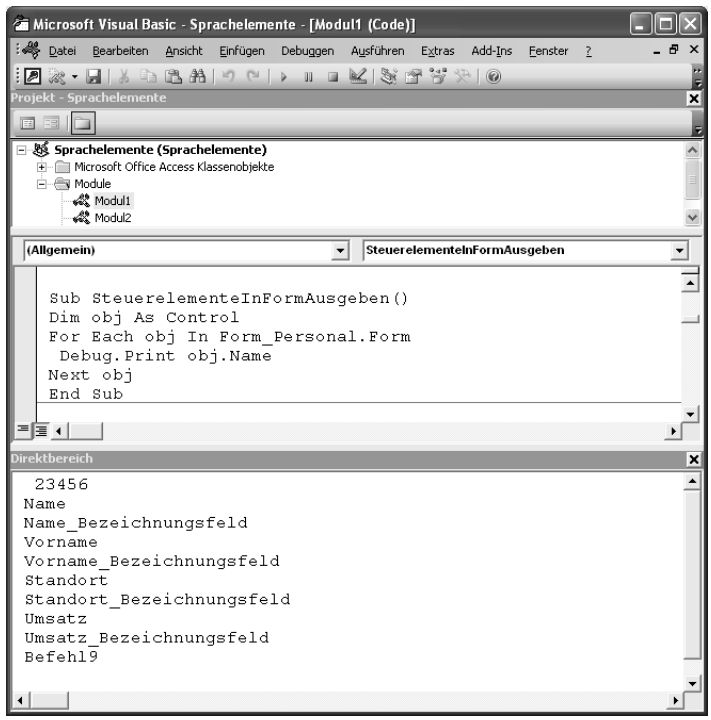


Abbildung 3.7: Alle Steuerelemente des Formulars werden ausgegeben

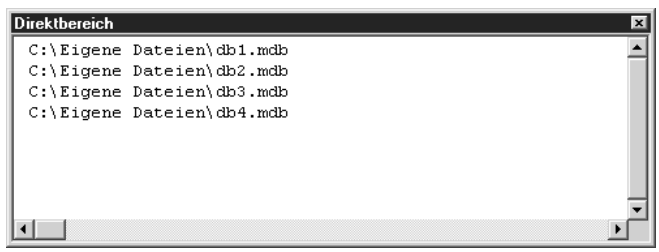
Im letzten Beispiel zur `For Each...Next`-Schleife sollen die Namen aller Access-Datenbanken eines Verzeichnisses und der darunter liegenden Verzeichnisse ermittelt und ausgegeben werden. Der Code für diese Aufgabe lautet:

Listing 3.14: Eine `For Each...Next`-Schleife zum Ermitteln aller Access-Datenbanken eines Verzeichnisses

```
Sub SchleifeForEach()
Dim obj As Variant

 Const verz = "C:\Eigene Dateien\"

 On Error GoTo fehler
 ChDir verz

  With Application.FileSearch
    .NewSearch
    .LookIn = verz
    .FileName = "*.mdb"
    .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` dann 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 Dateityp in einer Konstanten an, nach der gesucht werden soll. Möchten Sie beispielsweise nicht nur Access-Datenbanken suchen lassen, sondern alle Dateitypen, dann geben Sie die Konstante `msoFileTypeAllFiles` an. Über die Eigenschaft `FileCount` geben Sie dann den Namen der gefundenen Datei an und schreiben diesen in den Direktbereich Ihrer Entwicklungsumgebung.

Abbildung 3.8: Access-Datenbanken suchen



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.

Die Syntax dieser Schleife sieht wie folgt aus:

Die Syntax der `DoUntil...Loop` -Schleife

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

Die `Bedingung` stellt einen numerischen Ausdruck oder Zeichenfolgenausdruck dar, der entweder `True` oder `False` ergibt. 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...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.

Beim folgenden Beispiel wird eine Schleife genau 50 Mal durchlaufen. Das dazugehörige Makro sehen Sie in Listing 3.15.

Listing 3.15: `Do Until`-Schleife zum Bearbeiten eines Zählers

```
Sub SchleifeDoUntil()
Dim zz As Integer
Dim vorgabe As Integer

zz = 0
vorgabe = 60
 Do Until vorgabe = 10
  vorgabe = vorgabe - 1
  zz = zz + 1
 Loop

MsgBox " Die Schleife wurde " & zz & " mal durchlaufen."
End Sub
```

Initialisieren Sie zu Beginn des Makros Ihren Zähler und setzen diesen auf den Wert `0`. Legen Sie danach in der Variable `vorgabe` Ihren Vorgabewert fest. Setzen Sie die `Do Until`-Schleife auf und definieren als Ende-Bedingung

für die Schleife den Wert `10`. Die Schleife soll demnach so oft durchlaufen werden, bis der Wert `10` erreicht wird. Innerhalb der Schleife müssen Sie dafür sorgen, dass Ihr Countdown auch korrekt funktioniert. Dazu subtrahieren Sie jeweils den Wert `1` von der Zählvariablen `vorgabe`. Damit Sie am Ende ermitteln können, wie oft die Schleife durchlaufen wurde, zählen Sie innerhalb der Schleife die Zählvariable `zz` hoch und geben diese am Ende Ihres Makros aus.

Im nächsten Beispiel soll eine `Do Until`-Schleife verwendet werden, um die Namen aller Steuerelemente eines Formulars zu ermitteln. Den dazugehörigen Quellcode entnehmen Sie dem Listing 3.16.

Listing 3.16: Per `Do Until`-Schleife Steuerelemente in Formularen ermitteln

```
Sub SchleifeDoUntil02()
Dim obj As Control
Dim i As Integer
Dim frm As Form

i = 1
Set frm = Form_Personal

 With frm
 Do Until i = frm.Form.Count
  Debug.Print frm.Controls(i).Name
  i = i + 1
 Loop
 End With
End Sub
```

In Listing 3.16 sehen Sie, wie sich einiges an Schreibarbeit sparen lässt. Über die Anweisung `Set` weisen Sie dem Formular `Form_Personal` einen etwas kürzeren Namen zu, mit welchem Sie das Formular zukünftig ansprechen werden. Nun kommt die Anweisung `With` ins Spiel. Über diese Anweisung machen Sie bekannt, dass das Formular nun unter der Bezeichnung `frm` angesprochen werden soll. Achten Sie am Ende darauf, dass Sie diese Anweisung mit `End With` wieder abschließen. Setzen Sie im Anschluss daran die `Do Until`-Schleife auf. Als Ende-Bedingung setzen Sie den Wert ein, den Sie über die Eigenschaft `Count` ermittelt haben. Gezählt werden in diesem Fall die sich auf dem Formular befindlichen Steuerelemente. Innerhalb der Schleife geben Sie die Namen der Steuerelemente, die Sie über die Eigenschaft `Name` bekommen, im Direkt-Fenster aus.

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 `True` oder `False` ergibt. 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.

Im folgenden Beispiel sollen in einem bestimmten Verzeichnis alle Access-Datenbanken gezählt werden. Den dafür notwendigen Code sehen Sie in Listing 3.17.

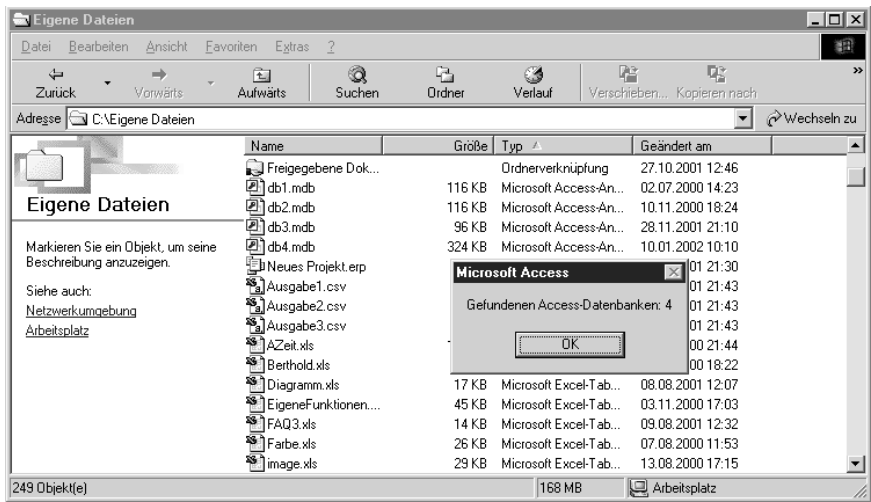
Listing 3.17: `Do While... Loop`-Schleife zum Ermitteln der Anzahl von Datenbanken eines Verzeichnisses

```
Sub DateienZählen()
Dim Verzeichnis As String
Dim s As String
Dim i As Integer

  Verzeichnis = "c:\Eigene Dateien\"
  s = Dir(Verzeichnis & "*.mdb")
  While s <> ""
    s = Dir
    i = i + 1
  Wend
  MsgBox "Gefundene Access-Datenbanken: " & i
End Sub
```

Zu Beginn des Makros legen Sie fest, auf welchem Laufwerk bzw. in welchem Verzeichnis nach Access-Datenbanken gesucht werden soll. Danach setzen Sie die Funktion `Dir` ein. Diese Funktion gibt eine Zeichenfolge (String) zurück, die den Namen einer Datei, eines Verzeichnisses oder eines Ordners darstellt, der mit einem bestimmten Suchmuster, einem Dateiattribut oder

mit der angegebenen Datenträger- bzw. Laufwerksbezeichnung übereinstimmt. Geben Sie dieser Funktion noch als Information mit, wo sie suchen und nach welchen Dateien sie Ausschau halten soll. Speichern Sie den ersten gefundenen Wert in der Variable `s`. Setzen Sie danach die `Do While...Loop`-Schleife auf, die so lange durchlaufen werden soll, bis keine weiteren Access-Datenbanken mehr gefunden werden. Innerhalb der Schleife suchen Sie jeweils die nächste Datenbank im angegebenen Verzeichnis.

Abbildung 3.9: Dateien zählen und ausgeben



3.4 VBA-Funktionen einsetzen

Neben den typischen Access-Funktionen gibt es eine ganze Reihe von VBA-Funktionen und -Anweisungen, die Sie übrigens auch in anderen Office-Programmen wie Excel, Word oder PowerPoint einsetzen können. In diesem Abschnitt lernen Sie diese wichtigen Funktionen anhand von Beispielen aus der Praxis kennen.

Laufwerk und Verzeichnis einstellen

Wenn Sie Access-Datenbanken oder auch andere Dateien über ein Makro öffnen oder speichern möchten, sollten Sie vorher sicherstellen, dass das richtige Laufwerk bzw. das gewünschte Verzeichnis eingestellt ist. Dazu können Sie die Funktionen `ChDrive` und `ChDir` einsetzen. Mithilfe der Funktion `ChDrive` wechseln Sie auf das angegebene Laufwerk, mit der Funktion `ChDir` springen Sie direkt in das gewünschte Verzeichnis.

Listing 3.18: Laufwerk und Verzeichnis einstellen

```
Sub LaufwerkUndVerzeichnisEinstellen()
Dim s As String
Const LW = "c:\"
Const Verzeichnis = "c:\Eigene Dateien\"
```

```
   On Error GoTo fehler1
 ChDrive LW
 On Error GoTo fehler2
 ChDir Verzeichnis
 MsgBox "Sie befinden sich jetzt auf dem Laufwerk: " _
 & LW & Chr(13) & " im Verzeichnis " & Verzeichnis
'weitere Aktionen
 Exit Sub

fehler1:
 MsgBox "Das Laufwerk " & LW & " ist nicht verfügbar!"
 Exit Sub
fehler2:
 MsgBox "Das Verzeichnis " & Verzeichnis & _
 " existiert nicht!"
 Exit Sub
End Sub
```

Es empfiehlt sich, die Verzeichnisse nicht irgendwo »hart« im Code zu editieren. Definieren Sie stattdessen das gewünschte Laufwerk sowie das Verzeichnis, auf das Sie zugreifen möchten, zu Beginn des Makros in einer Konstanten. Setzen Sie auch die `On Error`-Anweisung an, bevor Sie auf das Laufwerk bzw. das Verzeichnis positionieren. Für den Fall, dass das Laufwerk bzw. das Verzeichnis nicht vorhanden ist, können Sie somit eine saubere Fehlerbehandlung durchführen.

Textdateien einlesen

Oft kommt es vor, dass Ihnen Informationen nicht in Access-Datenbanken, sondern in Textdateien geliefert werden. Sie haben dann die Aufgabe, diese Textdateien einzulesen. Das folgende Beispiel liest eine Textdatei in das Direkt-Fenster Ihrer Entwicklungsumgebung ein. Dabei wird eine sehr wichtige VBA-Funktion gebraucht, um das Ende der Textdatei festzustellen. Diese Funktion heißt `EOF` (von engl. »End Of File«). Sehen Sie die Lösung dieser Aufgabe in Listing 3.19.

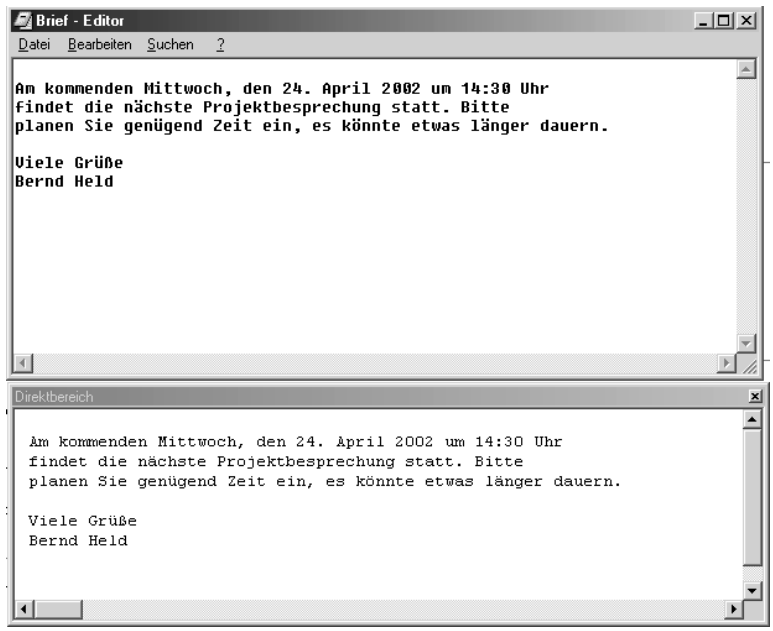
Listing 3.19: Textdatei einlesen

```
Sub TextdateiInDirektbereich()
Dim textzeile As String

 Open "c:\Eigene Dateien\Brief.txt" _
  For Input As #1
  Do While Not EOF(1)
   Line Input #1, textzeile
   Debug.Print textzeile
  Loop
  Close #1
End Sub
```

Bevor Sie das Makro ausführen, legen Sie im Ordner C:\EIGENE DATEIEN eine Textdatei mit dem Namen BRIEF.TXT an und tippen Sie ein paar beliebige Textzeilen ein.

Öffnen Sie mit der Methode `Open` die angegebene Textdatei. Setzen Sie danach die `Do While`-Schleife auf und geben Sie als Endkriterium die Funktion `EOF` an. Über die Anweisung `Line Input` lesen Sie die einzelnen Zeilen der Textdatei in die Variable `textzeile` ein. Ebenfalls in der Schleife geben Sie die einzelnen Zeilen über die Anweisung `Debug.Print` im Direktbereich aus. Vergessen Sie nicht, am Ende des Makros die Textdatei über die Methode `Close` zu schließen.

Abbildung 3.10: Textdatei auslesen und im Direktbereich ausgeben



Eingegebene E-Mail-Adressen prüfen

Eine sehr interessante und oft eingesetzte Funktion heißt `InStr`. Mit dieser Funktion können Sie prüfen, ob ein Text eine bestimmte Zeichenfolge enthält oder nicht.

Im folgenden Makro wird geprüft, ob ein eingegebener Text eine gültige E-Mail-Adresse enthält.

Listing 3.20: Wurde eine gültige E-Mail-Adresse eingegeben?

```
Sub TextePrüfen()
Dim s As String

 s = InputBox _
("Geben Sie eine E-Mail-Adresse ein!")
 If s = "" Then Exit Sub
 If InStr(s, "@") > 0 Then _
 MsgBox "E-Mail-Adresse!" Else _
 MsgBox "Keine gültige E-Mail-Adresse!"
End Sub
```

Die Funktion `InStr` gibt einen Wert vom Typ `Variant` zurück, der die Position des ersten Auftretens einer Zeichenfolge innerhalb einer anderen Zeichenfolge angibt. Es handelt sich in diesem vereinfachten Beispiel demnach um eine gültige E-Mail-Adresse, wenn das Sonderzeichen @ darin vorkommt. Wenn Sie zum Beispiel die E-Mail-Adresse `machero@aol.com` eingeben, meldet Ihnen die Funktion `InStr` den Wert `8`. Das achte Zeichen von links entspricht dem Sonderzeichen @.

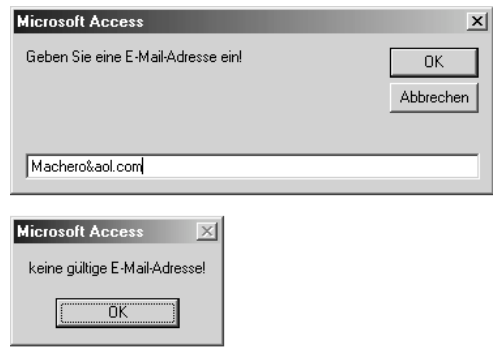


Abbildung 3.11: E-Mail-Adressen checken

Textteile extrahieren

Möchten Sie einen bestimmten Textteil aus einem Gesamttext extrahieren, dann suchen Sie standardmäßig über die Funktion `Instr` beginnend vom linken Ende des Textes nach einer bestimmten Zeichenfolge, ab der die gewünschten Zeichen in einen anderen String übertragen werden sollen. Sehen Sie sich nun einmal folgenden String an:

```
"//Daten/Beispiele/Held/Datenbank.mdb"
```

In diesem Fall soll nach dem letzten Zeichen »/« gesucht werden, um letztendlich den Dateinamen aus dem Text heraus zu extrahieren. In diesem Fall wäre es ein wenig ungeschickt, über die Funktion `instr` zu arbeiten, da diese Funktion, wie schon gesagt, vom linken Beginn des Textes her das Zeichen »/« sucht. Hier müssten Sie dann eine Schleife programmieren und den Text solange nach links durchlaufen, bis Sie das letzte Zeichen »/« gefunden haben. Das muss aber nicht sein. Setzen Sie stattdessen die Funktion `InstrRev` ein. Mithilfe dieser Funktion erhalten Sie die genaue Position des gesuchten Zeichens bzw. die Position des ersten Zeichens der gesuchten Zeichenfolge. Dabei stellt diese Angabe jedoch wieder die gezählten Zeichen von links dar. Zusammenfassend ist zu sagen, dass die Suche bei der Funktion `InstrRev` demnach von rechts stattfindet, das Ergebnis, also die Position des Zeichens, wird aber von links angegeben.

Lassen Sie uns diese Funktion nun an einem Beispiel üben. Aus dem Text

```
"//Daten/Beispiele/Held/Datenbank.mdb"
```

soll der Name der Datei extrahiert werden. Dazu sind folgende Einzelschritte notwenig:

- Ermittlung des letzten Zeichens »/« vom linken Rand des Textes gesehen, bzw. Ermitteln des ersten Zeichens »/« vom rechten Rand des Textes her gesehen
- Feststellen der Gesamtlänge des Textes
- Ermittlung der Länge des gesuchten Textteiles
- Ausgabe des gesuchten Textteiles

Schreiben Sie für diesen Zweck folgendes Makro aus dem hier gezeigten Listing.

Listing 3.21: Textteile extrahieren

```
Sub TextteilSuchen()
Dim s As String
Dim i As Integer
Const Suchtext = "//Daten/Beispiele/Held/Datenbank.mdb"

i = InStrRev(Suchtext, "/")
Debug.Print "Position des letzten /-Zeichens " & i

l = Len(Suchtext)
Debug.Print "Länge des Gesamttextes " & l

l = Len(Suchtext) - i
Debug.Print "Länge des gesuchten Textteils " & l

s = Mid(Suchtext, i + 1, l)
Debug.Print "Der gesuchte Textteil lautet: " & s
End Sub
```

Der Gesamttext ist in diesem Beispiel in einer Konstanten angegeben. Über die Funktion `InstrRev` ermitteln Sie die Position des ersten Aufretens der Zeichenfolge »/« vom rechten Rand her gesehen.

Mithilfe der Funktion `Len` ermitteln Sie die Gesamtlänge des Textes. Die Länge des gesuchten Textes erfragen Sie, indem Sie die ermittelte Position des Zeichens »/« von der Länge des Gesamttextes subtrahieren.

Im letzten Schritt setzen Sie die Funktion `Mid` ein, um den gesuchten Teil des Gesamttextes in der Variablen `s` zu speichern. Dazu übergeben Sie dieser Funktion die vorher ermittelte Position des Zeichens »/« und addieren den Wert `1`, damit dieses Zeichen nicht mit übertragen wird. Das zweite Argument für diese Funktion beinhaltet die Länge der Zeichen, die übertragen werden müssen. Auch diese Information haben Sie bereits ermittelt.

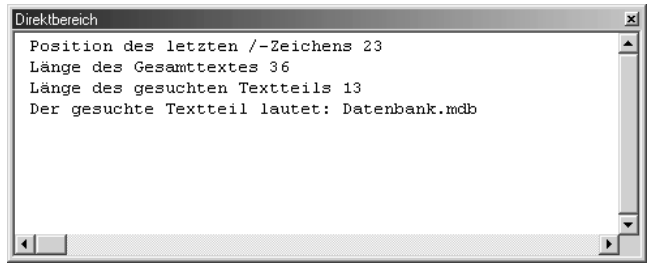


Abbildung 3.12: Textteile extrahieren

Dateiendungen prüfen

Mit einer ähnlichen Funktion können Sie die Suche nach bestimmten Zeichen von rechts nach links durchlaufen lassen. Diese Funktion heißt `InStrRev`. Sie gibt die Position einer Zeichenfolge in einer anderen Zeichenfolge vom Ende der Zeichenfolge gesehen an.

Im nächsten Makro prüfen Sie die Eingabe einer Internetseite. Diese Eingabe erklären Sie dann für gültig, wenn sie mit der Endung .HTM bzw. .HTML eingegeben wurde. Sehen Sie sich dazu das folgende Listing an.

Listing 3.22: Wurde eine gültige Internetseite eingegeben?

```
Sub DateiEndungenPrüfen()
Dim s As String

 s = InputBox _
("Geben Sie eine gültige Internetseite an!")
 If s = "" Then Exit Sub
 If InStrRev(s, "htm") > 0 Then _
 MsgBox "Gültige Internetseite!" Else _
 MsgBox "Keine gültige Internetseite!"
End Sub
```

Wenn Sie die Internetseite INDEX.HTML erfassen, meldet Ihnen die Funktion `InStrRev` den Wert `7`. Die Funktion `InStr` würde ebenso den Wert `7` melden.

Sie fragen sich jetzt, warum man dann einen Unterschied gemacht hat. Der Unterschied wird im folgenden Listing deutlich.

Listing 3.23: Der Unterschied zwischen `InStr` und `InStrRev`

```
Sub Unterschied()
Dim s As String

s = "C:\Eigene Dateien\Test.txt"
 MsgBox s & Chr(13) & Chr(13) & _
 "InStrRev: " & InStrRev(s, "\") _
 & Chr(13) & "InStr  : " & InStr(s, "\")
End Sub
```

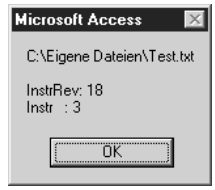


Abbildung 3.13: Unterschiedliche Ergebnisse bei `InStr` und `InStrRev`

Die Funktion `InStrRev` meldet die Position 18, was dem letzten Backslash entspricht, während die Funktion `InStr` den Wert `3` liefert, was den ersten Backslash im Pfad darstellt. Je nachdem, wie Sie also einen String durchsuchen möchten, müssen Sie entweder die Funktion `InStr` oder die Funktion `InStrRev` einsetzen.

:-) TIPP

Im letzten Listing haben Sie die Funktion `Chr(13)` *verwendet, die einen Zeilenvorschub erzeugt. Dafür gibt es als Alternative auch VBA-Konstanten, die Sie der folgenden Tabelle entnehmen können.*

Tabelle 3.4: Die Konstanten für die Steuerzeichen

Konstante	Beschreibung
`vbCrLf`	Kombination aus Wagenrücklauf und Zeilenvorschub
`vbCr`	Wagenrücklaufzeichen
`vbLf`	Zeilenvorschubzeichen
`vbNewLine`	Plattformspezifisches Zeilenumbruchzeichen; je nachdem, welches für die aktuelle Plattform geeignet ist
`vbNullChar`	Zeichen mit dem Wert `0`
`vbNullString`	Zeichenfolge mit dem Wert `0`; nicht identisch mit der Null-Zeichenfolge (»«); wird verwendet, um externe Prozeduren aufzurufen
`vbTab`	Tabulatorzeichen
`vbBack`	Rückschrittzeichen

Texte kürzen und extrahieren

VBA stellt Ihnen einige Funktionen zur Verfügung, mit denen Sie Texte bearbeiten können. So können Sie mithilfe der beiden Funktionen `Left` und `Right` einen Text je nach Bedarf kürzen. Mit der Funktion `Mid` können Sie einen Teil des Textes aus einem Gesamttext heraus extrahieren.

Im folgenden Beispiel soll ein eingegebener Text auf zehn Zeichen gekürzt werden. Um diese Aufgabe zu lösen, starten Sie das Makro aus dem folgenden Listing.

Listing 3.24: Texte kürzen mit der Funktion `Left`

```
Sub TextKürzen()
Dim s As String

 s = InputBox("Geben Sie einen Text ein!")
 If s = "" Then Exit Sub
 If Len(s) > 10 Then s = Left(s, 10)
 MsgBox s
End Sub
```

Mit der Funktion `Len` können Sie prüfen, wie viele Zeichen eingegeben wurden. Sollten mehr als zehn Zeichen eingegeben worden sein, so übertragen Sie mithilfe der Funktion `Left` zeichenweise genau zehn Zeichen und speichern diese in der Variablen `s`.

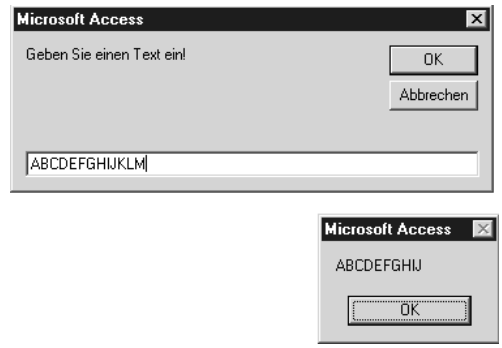


Abbildung 3.14: Den Text nach zehn Zeichen abschneiden

Die Funktion `Right` kommt beispielsweise dann zum Einsatz, wenn Sie eine Zeichenfolge eines Textes von der rechten Seite aus übertragen möchten.

So wird im folgenden Beispiel aus einer Datumseingabe das Jahr extrahiert und weiterverarbeitet. Sehen Sie sich dazu das nächste Makro an.

Listing 3.25: Texte kürzen mit der Funktion `Right`

```
Sub TextKürzen02()
Dim s As String

 s = InputBox _
 eben Sie ein Datum *01.01.2002* ein!")
 If s = "" Then Exit Sub
 s = Right(s, 4)
 MsgBox s
End Sub
```

Geben Sie bei der Funktion `Right` an, welchen Text Sie bearbeiten und wie viele Zeichen davon, von rechts beginnend, Sie übertragen möchten.

Gehen Sie noch einen Schritt weiter und extrahieren Sie bei folgendem Makro den Dateinamen aus einer kompletten Pfadangabe.

Listing 3.26: Textteile extrahieren mit der Funktion `Mid`

```
Sub TextExtrahieren()
Dim s As String
Dim s2 As String

s = Application.CurrentDb.Name
s2 = Mid(s, InStrRev(s, "\") + 1)
MsgBox "Aus " & s & Chr(13) & " wird " & Chr(13) & s2
End Sub
```

Speichern Sie zunächst den Dateinamen der aktuell geöffneten Datenbank mithilfe der Methode `CurrentDb` sowie der Eigenschaft `Name` in der Variablen `s`. Nun wird neben dem Dateinamen auch der komplette Speicherpfad der Datenbank angezeigt. Über die Funktion `Mid` können Sie jetzt den Dateinamen extrahieren. Um die Position des letzten Backslashs herauszufinden, setzen Sie die Funktion `InStrRev` ein. Damit dieser Backslash nicht mit übertragen wird, addieren Sie den Wert `1`.

Abbildung 3.15: Die Funktion `Mid` in der praktischen Anwendung

Texte splitten

Seit der Access-Version 2000 gibt es die VBA-Funktion `Split`. Mithilfe dieser Funktion können Sie Zeichenfolgen in Einzelteile zerlegen.

Bei der nächsten Aufgabe wird ein Text, der durch Semikola getrennt ist, in einzelne Felder aufgeteilt und anschließend im Direkt-Fenster der Entwicklungsumgebung ausgegeben.

Listing 3.27: Texte splitten mithilfe der Funkton `Split`

```
Sub TexteZerlegen()
Dim sText As String
Dim Varray As Variant
Dim i As Integer
Dim iGesamt As Integer

sText = "Bernd;Held;Teststr. 15;70839;Gerlingen"
Varray = Split(sText, ";")
iGesamt = UBound(Varray)
For i = 0 To iGesamt
 Debug.Print "Feld " & i + 1 & ": " & Varray(i)
Next i
End Sub
```

In der String-Variablen `sText` wird zunächst der komplette Datensatz erfasst. Die einzelnen Felder sind hier durch ein Semikolon voneinander getrennt.

Danach wenden Sie die Funktion `Split` an, um die einzelnen Feldinhalte in ein Datenfeld zu lesen. Dabei geben Sie im ersten Argument an, welcher Text gesplittet werden soll. Im zweiten Argument geben Sie das Seperatorzeichen, hier das Semikolon, an. Das Datenfeld `Varray` wird vorher mit dem Datentyp `Variant` deklariert. Nachdem Sie das Datenfeld gefüllt haben, ermitteln Sie mithilfe der Funktion `UBound`, wie viele einzelne Daten im Datenfeld eingelesen wurden. Dabei muss darauf geachtet werden, dass das erste Feld im Datenfeld mit dem Index `0` beginnt. Um also den Vornamen aus dem Datenfeld zu ermitteln, könnte man `Msgbox Varray(1)` schreiben.

In einer Schleife lesen Sie ein Feld nach dem anderen aus und geben es über die Anweisung `Debug.Print` im Direkt-Fenster der Entwicklungsumgebung aus.

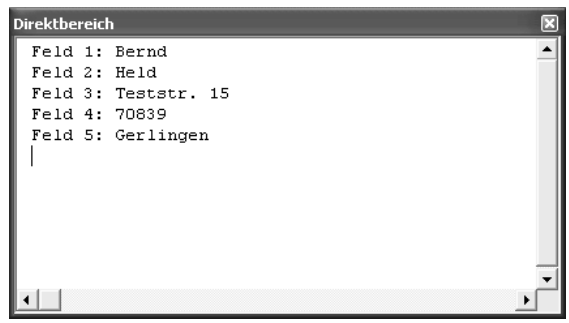


Abbildung 3.16: Der komplette Text wurde auf einzelne Felder aufgeteilt

Texte zerlegen, konvertieren und wieder zusammensetzen

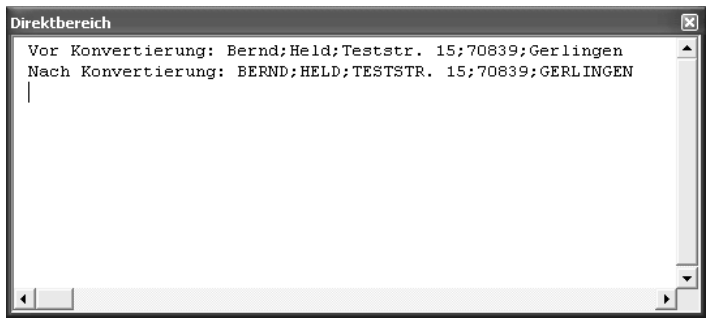
Im nächsten Beispiel wird ein Text komplett zerlegt, dann werden Änderungen vorgenommen und anschließend wird er wieder zusammengesetzt. Zu Beginn liegt ein Text vor, bei dem die einzelnen Informationen durch Semikola getrennt sind. Nach der Zerlegung der einzelnen Teile werden diese in Großbuchstaben gewandelt und wieder zu einem Gesamttext zusammengesetzt. Das Makro für diese Aufgabe lautet:

Listing 3.28: Texte zerlegen, konvertieren und wieder zusammensetzen

```
Sub TexteZerlegenUndVerbinden()
Dim sText As String
Dim Varray As Variant
Dim i As Integer
Dim iGesamt As Integer

sText = "Bernd;Held;Teststr. 15;70839;Gerlingen"
Debug.Print "Vor Konvertierung: " & sText
Varray = Split(sText, ";")
iGesamt = UBound(Varray)
For i = 0 To iGesamt
 Varray(i) = UCase(Varray(i))
Next i
Varray = Join(Varray, ";")
Debug.Print "Nach Konvertierung: " & Varray
End Sub
```

Der Text liegt zunächst in einer `String`-Variablen vor. Die einzelnen Textinformationen werden durch Semikola voneinander getrennt. Über den Einsatz der Funktion `Split` zerlegen Sie diesen Gesamttext in einzelne Teile. Danach durchlaufen Sie in einer Schleife alle einzelnen Felder, die jetzt im Datenfeld `Varray` sind. Innerhalb der Schleife wenden Sie die Funktion `Ucase` an, um die einzelnen Informationen in Großbuchstaben zu wandeln. Am Ende des Makros kommt dann die Funktion `Join` zum Einsatz, um die einzelnen Felder wieder zusammenzusetzen. Übergeben Sie dazu dieser Funktion im ersten Argument das Datenfeld, das die konvertierten Texte enthält. Im zweiten Argument geben Sie das Separatorzeichen bekannt, das verwendet werden soll.

Abbildung 3.17: Texte zerlegen und wieder zusammensetzen



Texte bereinigen

Wenn Sie Transfers von Texten in Ihre Datenbanken vornehmen, sollten Sie vorher dafür sorgen, dass keine Leerzeichen mit übertragen werden. Für diesen Zweck stellt Ihnen VBA drei Funktionen zur Verfügung. Die Funktion `LTrim` entfernt führende Leerzeichen (linker Rand), die Funktion `RTrim` eliminiert nachgestellte Leerzeichen (rechter Rand) und die Funktion `Trim` entfernt vor- und nachgestellte Leerzeichen.

Der folgende Quellcode gibt dafür ein Beispiel:

Listing 3.29: Leerzeichen entfernen mit der Funktion `Trim`

```
Sub LeerzeichenEntfernen()
Const s = _
"  da stimmt es hinten wie vorne nicht "

MsgBox "Aus dem Text: " & Chr(13) & s & _
Chr(13) & " wird: " & Chr(13) & Trim(s)
End Sub
```

Zahlenwerte runden

Für das Runden von Zahlenwerten steht Ihnen in VBA eine eigene Funktion zur Verfügung. Diese Funktion heißt `Round`. Sehen Sie in Listing 3.30, was Sie bei dieser Funktion beachten müssen.

Abbildung 3.18: Texte bereinigen

```
Sub Runden()
Const Zahl1 = 1.456

 MsgBox Round(Zahl1, 2)
End Sub
```

Listing 3.30: Zahlen runden mit der Funktion `Round`

Geben Sie bei der Funktion `Round` an, welchen Wert Sie runden möchten sowie die Anzahl der Stellen hinter dem Komma.

Dateien löschen

VBA stellt Ihnen eine Anweisung zur Verfügung, die Ihnen erlaubt, Dateien zu löschen, ohne vorher den Explorer aufzurufen. Diese Anweisung heißt `Kill`.

Im folgenden Beispiel werden in einem vorher angegebenen Verzeichnis alle Textdateien gelöscht.

```
Sub DateienLöschen()
On Error GoTo fehler
 ChDir ("C:\Temp\")
 Kill "*.txt"
 Exit Sub

fehler:
 MsgBox "Das Verzeichnis konnte nicht gefunden werden!"
End Sub
```

Listing 3.31: Dateien löschen mit `Kill`

Beim Einsatz der Anweisung `Kill` können Sie so genannte Platzhalter einsetzen oder den Namen der zu löschenden Datei komplett angegeben. Die Dateien werden direkt gelöscht, ohne dass vorher noch einmal nachgefragt wird. Möchten Sie eine Sicherheitsabfrage einbauen, dann vervollständigen Sie den Code aus Listing 3.31 wie folgt:

```
Sub DateienLöschenMitRückfrage()
Dim i As Integer

On Error GoTo fehler
 ChDir ("C:\Temp\")
 i = MsgBox _
("Möchten Sie die Dateien wirklich löschen?", vbYesNo)
 If i = 6 Then Kill "*.txt" Else _
 MsgBox "Sie haben die Aktion abgebrochen!"
 Exit Sub
```

```
fehler:
 MsgBox "Das Verzeichnis konnte nicht gefunden werden!"
End Sub
```

Abbildung 3.19: Ja oder Nein?

Mit der Konstanten `vbYesNo` zeigen Sie das Meldungs-Fenster mit den beiden Schaltflächen JA und NEIN an. Um zu ermitteln, welche Schaltfläche der Anwender geklickt hat, müssen Sie diese abfragen. Klickt der Anwender die Schaltfläche JA AN, wird der Wert `6` gemeldet. Klickt er hingegen auf die Schaltfläche NEIN, wird der Wert `7` zurückgegeben. In Abhängigkeit davon führen Sie dann Ihre Löschaktion durch oder nicht.

Verzeichnisse erstellen

Um ein neues Verzeichnis anzulegen, brauchen Sie nicht extra den Windows-Explorer zu starten, sondern Sie können dies direkt mit einer VBA-Anweisung erledigen. Diese Anweisung lautet `MkDir`.

Im nächsten Beispiel in Listing 3.32 wird unterhalb des Verzeichnisses C:\EIGENE DATEIEN ein weiteres Verzeichnis angelegt.

Listing 3.32: Verzeichnisse anlegen mit `MkDir`

```
Sub VerzeichnisAnlegen()
Const Verz = "Sammlung"

 ChDir "C:\Eigene Dateien"
 On Error GoTo fehler
 MkDir Verz
 Exit Sub

fehler:
 MsgBox "Es ist ein Problem aufgetreten!"
End Sub
```

Wird übrigens kein Laufwerk angegeben, so wird das neue Verzeichnis bzw. der neue Ordner auf dem aktuellen Laufwerk erstellt.

Analog dazu gibt es selbstverständlich auch eine VBA-Anweisung, um ein Verzeichnis zu entfernen. Diese Anweisung lautet `RmDir`.

Sehen Sie im Code von Listing 3.33, wie Sie diese Anweisung einsetzen können.

```
Sub VerzeichnisLöschen()
Const Verz = "Sammlung"

 ChDir "C:\Eigene Dateien"
 On Error GoTo fehler
 RmDir Verz
 Exit Sub

fehler:
 MsgBox "Es ist ein Problem aufgetreten!"
End Sub
```

Listing 3.33: Verzeichnisse löschen mit `RmDir`

Ein Fehler tritt auf, wenn Sie `RmDir` *für ein Verzeichnis oder einen Ordner ausführen, in dem Dateien enthalten sind. Löschen Sie zuerst alle Dateien mit der* `Kill`*-Anweisung, bevor Sie ein Verzeichnis oder einen Ordner entfernen.*

Arbeitsverzeichnis ermitteln

Möchten Sie das aktuelle Arbeitsverzeichnis Ihrer Access-Installation ermiteln, dann wählen Sie aus dem Menü EXTRAS den Befehl OPTIONEN. Wechseln Sie danach auf die Registerkarte ALLGEMEIN und werfen einen Blick in das Eingabefeld STANDARDDATENBANKORDNER.

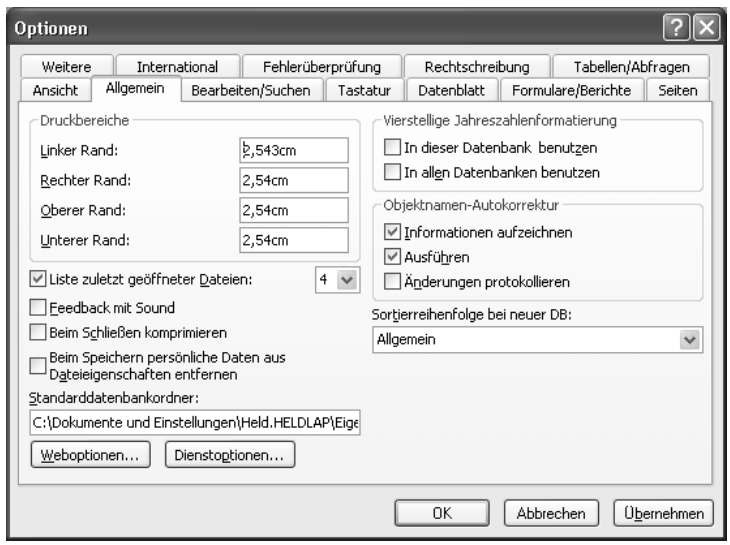


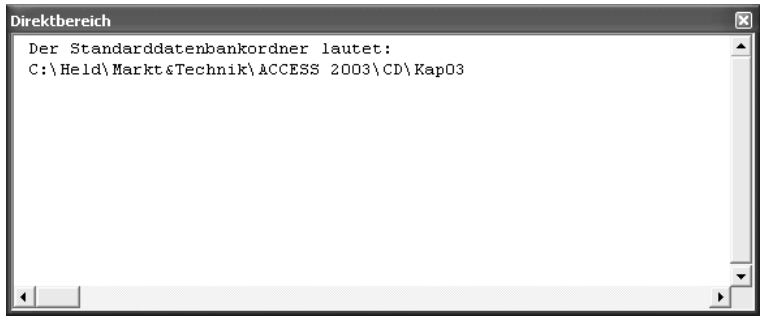
Abbildung 3.20: Das Standardarbeitsverzeichnis ermitteln

Dieselbe Information können Sie auch gewinnen, indem Sie das folgende Makro starten.

```
Sub StandardVerzeichnisAusgeben()
 Debug.Print _
 "Der Standarddatenbankordner lautet: " & vbLf & CurDir
End Sub
```

Listing 3.34: Den Standarddatenbankordner ermitteln

Abbildung 3.21: Das Standardarbeitsverzeichnis ausgeben



In der Online-Hilfe von Access steht allerdings, dass Sie mit der Funktion `CurDir` *das aktuelle Verzeichnis ausgeben können, in dem Sie sich befinden. Dies ist aber offensichtlich nicht korrekt. Möchten Sie wirklich das aktuelle Verzeichnis ermitteln, in dem sich Ihre gerade geöffnete Datenbank befindet, dann können Sie sich beispielsweise eine eigene Funktion schreiben, die Sie im folgenden Listing sehen.*

Listing 3.35: Den Speicherort der geöffneten Datenbank ermitteln

```
Function AktOrd() As String
Dim s As String

 s = CurrentDb().Name
 AktOrd = Left$(s, Len(s) - Len(Dir$(s)))
End Function


Sub AktuellesSpeicherVerzeichnis()
Dim ordner As String

ordner = AktOrd
Debug.Print "Der Speicherpfad der Datenbank lautet: " _
& vblf & ordner
End Sub
```

Mithilfe der Eigenschaft `Name`, welche Sie auf das Objekt `CurrentDb` anwenden, bekommen Sie den kompletten Pfad sowie den Namen der momentan geöffneten Datenbank. Wenden Sie die Funktion `Dir` nun direkt auf diesen ermittelten Gesamtpfad (`s`) an, um den Namen der geöffneten Datenbank zu finden. Danach erfragen Sie die Längen der beiden Teilstrings über die Funktion `Len` und erhalten durch eine Subtraktion beider Strings das gewünschte Ergebnis, welches Sie im Direktbereich von Access ausgeben.

Dateien kopieren

Möchten Sie Ihre Datenbanken regelmäßig sichern, dann können Sie sie mithilfe der Anweisung `FileCopy` kopieren.

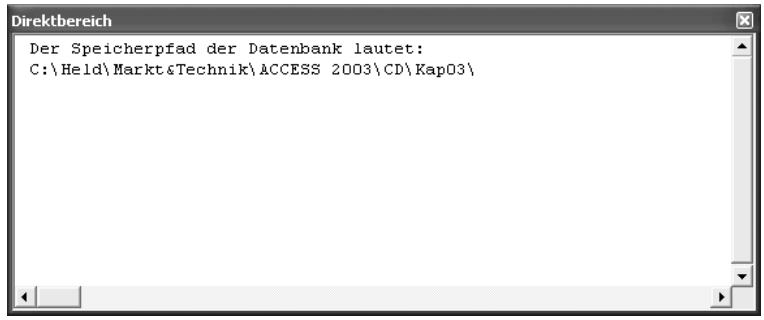


Abbildung 3.22: Das Speicherverzeichnis der geöffneten Datenbank ermitteln

Im nächsten Makro sehen Sie, wie Sie diese Aufgabe bewerkstelligen können.

Listing 3.36: Datenbank kopieren mit `FileCopy`

```
Sub DatenbankKopieren()
Dim s As String
Const LW = "c:\"
Const Verzeichnis = "c:\Eigene Dateien\"
Const DB = "DB4.mdb"
Const S_DB = "Sicherung_DB4.mdb"

 On Error GoTo fehler1
 ChDrive LW
 On Error GoTo fehler2
 ChDir Verzeichnis
 FileCopy DB, S_DB
 Exit Sub

fehler1:
 MsgBox "Das Laufwerk " & LW & " ist nicht verfügbar!"
 Exit Sub

fehler2:
 MsgBox "Das Verzeichnis " & Verzeichnis & _
 " existiert nicht!"
 Exit Sub
 End Sub
```

Die Anweisung `FileCopy` benötigt lediglich den Namen der Quelldatei sowie den gewünschten Namen des Duplikats.

Wochentag ermitteln

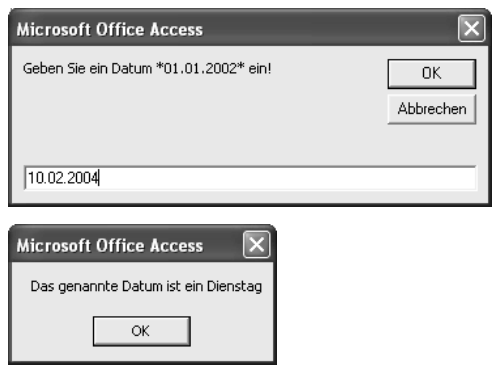
Wenn Sie ein Datum vor sich haben, dann wissen Sie nicht immer, um welchen Tag in der Woche es sich dabei handelt. Die Funktion `Weekday` meldet Ihnen einen Wert zwischen `1` und `7` zurück, wenn Sie diese Funktion mit einem gültigen Datumswert füttern. Wie das konkret aussieht, sehen Sie im folgenden Makro.

Listing 3.37: Wochentag ermitteln mit `Weekday`

```
Sub WochentagErmitteln()
Dim s As String
Dim WT As String

 s = InputBox _
("Geben Sie ein Datum *01.01.2002* ein!")
 If s = "" Then Exit Sub

  Select Case Weekday(s)
   Case 1
   WT = "Sonntag"
   Case 2
   WT = "Montag"
   Case 3
   WT = "Dienstag"
   Case 4
   WT = "Mittwoch"
   Case 5
   WT = "Donnerstag"
   Case 6
   WT = "Freitag"
   Case 7
   WT = "Samstag"
   End Select
  MsgBox "Das genannte Datum ist ein " & WT
 End Sub
```

Wenn Sie das Listing 3.37 betrachten, dann fällt auf, dass die Funktion `Weekday` den Wert `1` für den Sonntag meldet. Diese Besonderheit beruht höchstwahrscheinlich auf dem jüdischen Kalender, bei dem jede neue Woche mit dem Sonntag beginnt und daher den Index `1` bekommt.

Abbildung 3.23: Vom Datum den Wochentag ermitteln



Möchten Sie das Makro aus Listing 3.37 weiter verkürzen, können Sie eine zusätzliche Funktion einsetzen, die aus dem Zahlenwert, den Ihnen die Funktion `Weekday` meldet, automatisch den richtigen Wochentag angibt. Die Funktion hierfür lautet `WeekdayName`.

Listing 3.38: Wochentag ermitteln mit `WeekdayName`

```
Sub WochentagErmitteln_Kürzer()
Dim s As String

 s =InputBox _
("Geben Sie ein Datum *01.01.2002* ein!")

 If s = "" Then Exit Sub
 s = Weekday(s) - 1
 s = WeekdayName(s)
 MsgBox "Der genannte Tag ist ein: " & s
End Sub
```

Achten Sie darauf, dass Sie den Wert `1` *vom Ergebnis subtrahieren, bevor Sie den Wert an die Funktion* `WeekdayName` *übergeben, da es sonst zu einem fehlerhaften Ergebnis kommt.*

Monat ermitteln

Verwandt mit der gerade beschriebenen Funktion ist auch die Funktion `Month`. Diese Funktion meldet aufgrund eines Datums den dazugehörenden Monat. Diese Funktion gibt einen Wert vom Typ `Variant` zurück, der den Monat im Jahr als ganze Zahl im Bereich von 1 bis 12 angibt.

Sehen Sie im folgenden Beispiel aus Listing 3.39, wie Sie dieser Zahlenvergabe aussagefähige Namen geben können.

Listing 3.39: Monatsnamen ermitteln mit den Funktionen `Month` und `MonthName`

```
Sub MonatErmitteln()
Dim s As String
Dim s2 As String
Dim MN As String

 s = InputBox("Geben Sie ein Datum *01.01.2002* ein!")

 If s = "" Then Exit Sub
 s = Month(s)
 s2 = s
 s2 = Weekday(s2) - 1
 s2 = WeekdayName(s2)
 MsgBox "Das eingegeben Datum entspricht einem " _
 & s2 & " im Monat " & MonthName(s)
End Sub
```

Mithilfe der Funktion `Month` ermitteln Sie zunächst den aktuellen Monatsindex, den Sie an die Funktion `MonthName` übergeben. Diese Funktion bildet dann aus diesem Monatsindex den dazugehörigen Monatsnamen (`1`=`Januar`, `2`=`Februar` usw.).

Möchten Sie ganz flexibel aus einem Datum einen bestimmten Teil, sei es nun das Jahr, den Monat oder den Tag extrahieren, steht Ihnen für diese

Abbildung 3.24: Den Monatsnamen ausgeben

Aufgabenstellung eine weitere Funktion zur Verfügung. Diese Funktion heißt `DatePart`.

Wie Sie diese Funktion einsetzen, können Sie dem Listing 3.40 entnehmen.

Listing 3.40: Das Quartal aus einem Datum ermitteln

```
Sub TeilVomDatum()
Dim Datum1 As Date

Datum1 = InputBox("Geben Sie ein Datum ein:")
MsgBox "Quartal: " & DatePart("q", Datum1)
End Sub
```

Wie Sie sehen, müssen Sie der Funktion `DatePart` ein Kürzel hinzufügen, das angibt, welche Information Sie genau möchten. Der folgenden Tabelle können Sie weitere Möglichkeiten entnehmen, die Sie bei dieser Funktion haben.

Tabelle 3.5: Die Kürzel für die Funktion `DatePart`

Kürzel	Bedeutung
`yyyy`	Jahr
`q`	Quartal
`m`	Monat
`y`	Tag des Jahres
`d`	Tag
`w`	Wochentag
`ww`	Woche
`h`	Stunde

Kürzel	Bedeutung
n	Minute
s	Sekunde

Tabelle 3.5: Die Kürzel für die Funktion `DatePart` (Forts.)

Welche weiteren Argumente Sie bei dieser Funktion noch einstellen können, sehen Sie in der Online-Hilfe nach.

Datumsberechnungen durchführen

Für Datumsberechnungen steht Ihnen die Funktion `DatDiff` zur Verfügung. Sehen Sie in Listing 3.41, wie Sie beispielsweise die Differenz vom aktuellen Tag zu einem in der Zukunft liegenden Tag ermitteln können.

```
Sub Datumsberechnung()
Dim Datum1 As Date

 Datum1 = InputBox _
 ("Geben Sie ein zukünftiges Datum ein")
 MsgBox "Tage von heute an: " & _
 DateDiff("d", Now, Datum1)
End Sub
```

Listing 3.41: Datumsberechnungen mit der Funktion `DateDiff` ausführen

Auch hierbei gelten dieselben Kürzel wie schon in Tabelle 3.5 aufgeführt.

Abbildung 3.25: Das aktuelle Datum ist hier der 11.01.2002

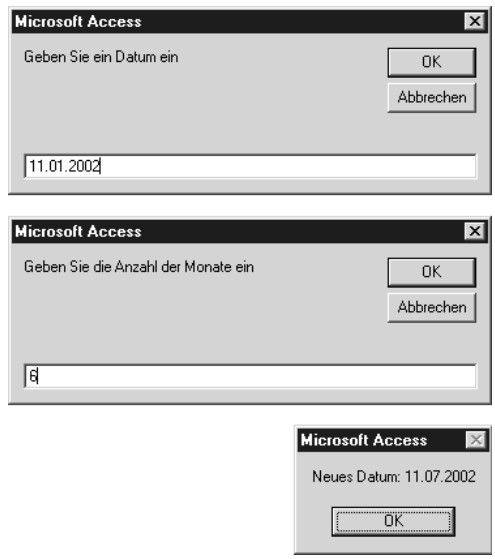
Verwandt mit der gerade beschriebenen Funktion ist auch die Funktion `DateAdd`. Damit können Sie Termine in der Zukunft bzw. in der Vergangenheit errechnen. Auch hier gelten wieder die Kürzel aus Tabelle 3.5.

Im nächsten Beispiel aus Listing 3.42 wird der Termin vom aktuellen Datum aus bestimmt. In einem Eingabefeld tragen Sie die Anzahl der Monate ein.

Listing 3.42: Datumsberechnungen mit der Funktion `DateAdd`

```
Sub Datumsberechnung02()
Dim Datum1 As Date
Dim s As String
Dim Zahl As Integer

s = "m"
Datum1 = InputBox("Geben Sie ein Datum ein")
Zahl = InputBox _
("Geben Sie die Anzahl der Monate ein")
MsgBox "Neues Datum: " & DateAdd(s, Zahl, Datum1)
End Sub
```

Abbildung 3.26: Vom aktuellen Datum sechs Monate in die Zukunft



Datumsangaben formatieren

Möchten Sie eine Datumsangabe in ein bestimmtes Format bringen, dann können Sie dazu die Eigenschaft `Format` verwenden.

Im folgenden Beispiel in Listing 3.43 liegt ein Datum im Format `30.10.02` vor. Ihre Aufgabe besteht nun darin, diese Eingabe in das Format `31.10.2002` zu formatieren.

Listing 3.43: Datumsangaben formatieren (vierstelliges Jahr)

```
Sub DatumFormatieren()
Const EDatum = "30.10.02"

MsgBox Format(EDatum, "DD.MM.YYYY")
End Sub
```

Geben Sie das Kürzel für die Jahresangabe viermal (`YYYY`) an, um das Jahr vierstellig anzuzeigen.

Abbildung 3.27: Ein Datum vierstellig ausgeben

Im nächsten Beispiel gehen Sie einen Schritt weiter und geben zusätzlich noch den Wochentag an. Das dazugehörige Makro können Sie im Listing 3.44 sehen.

```
Sub DatumFormatieren02()
Const EDatum = "30.10.02"

MsgBox Format(EDatum, "DD.MM.YYYY (DDDD)")
End Sub
```

Listing 3.44: Datumsangaben formatieren (Wochentag)

Indem Sie das Kürzel für den Tag (`DDDD`) viermal angeben, wird der Tag in ausgeschriebener Form ausgegeben.

Abbildung 3.28: Den Wochentag des Datums ermitteln

Im folgenden Beispiel soll die Kalenderwoche eines eingegebenen Datums ermittelt werden. Das Makro für diese Aufgabe sehen Sie in Listing 3.45.

```
Sub DatumFormatieren03()
Const EDatum = "30.10.02"

MsgBox "Wochennummer: " & Format(EDatum, "ww")
End Sub
```

Listing 3.45: Datumsangaben formatieren (Kalenderwoche)

Möchten Sie zusätzlichen Text mit ausgeben, müssen Sie mit dem Zeichen `&` arbeiten und die beiden Befehlsfolgen miteinander verknüpfen. Um die Kalenderwoche zu ermitteln, geben Sie in der Eigenschaft `Format` das Kürzel (`ww`) an.

Abbildung 3.29: Die Kalenderwoche ermitteln

In den letzten Beispielen haben Sie bereits einige Datumskürzel kennen gelernt. Sehen Sie in der Tabelle 3.6 weitere mögliche Kürzel.

Tabelle 3.6: Die Datums- und Zeitkürzel von Access

Kürzel	Bedeutung
c	Vordefiniertes Standarddatum
d	Monatstag mit einer oder zwei Ziffern (1 bis 31)
dd	Monatstag mit zwei Ziffern (01 bis 31)
ddd	Die ersten drei Buchstaben des Wochentags (Son bis Sam)
dddd	Vollständiger Name des Wochentags (Sonntag bis Samstag)
w	Wochentag (1 bis 7)
ww	Kalenderwoche (1 bis 53)
m	Monat des Jahres mit einer oder zwei Ziffern (1 bis 12)
mm	Monat des Jahres mit zwei Ziffern (01 bis 12)
mmm	Die ersten drei Buchstaben des Monats (Jan bis Dez)
mmmm	Vollständiger Name des Monats (Januar bis Dezember)
q	Datum als Quartal angezeigt (1 bis 4)
y	Kalendertag (1 bis 366)
yy	Die letzten zwei Ziffern der Jahreszahl (01 bis 99)
yyyy	Vollständige Jahreszahl (0100 bis 9999)
h	Stunde mit einer oder zwei Ziffern (0 bis 23)
hh	Stunde mit zwei Ziffern (00 bis 23)
n	Minute mit einer oder zwei Ziffern (0 bis 59
nn	Minute mit zwei Ziffern (00 bis 59)
S	Sekunde mit einer oder zwei Ziffern (0 bis 59)
SMS	Sekunde mit zwei Ziffern (00 bis 59)
AM/PM	Zwölf-Stunden-Format mit den Großbuchstaben AM oder PM
am/pm	Zwölf-Stunden-Format mit den Kleinbuchstaben am oder pm
A/P	Zwölf-Stunden-Format mit den Großbuchstaben A oder P
a/p	Zwölf-Stunden-Format mit den Kleinbuchstaben a oder p

Zeitfunktionen einsetzen

In VBA stehen Ihnen einige Zeitfunktionen zur Verfügung, die Sie flexibel einsetzen können. Die wichtigsten Zeit-Funktionen sind:

- `Now`: Diese Funktion liefert neben der Uhrzeit auch gleich das aktuelle Tagesdatum.
- `Time`: Diese Funktion gibt die aktuelle Uhrzeit aus. Dabei holt sich Access diese Information aus der eingestellten Zeit in der Systemsteuerung von Windows.
- `Hour`: Diese Funktion liefert die Stunde einer angegebenen Uhrzeit.
- `Minute`: Diese Funktion liefert die Minuten einer angegebenen Uhrzeit.
- `Second`: Diese Funktion liefert die Sekunden einer angegebenen Uhrzeit.

Im folgenden Beispiel werden diese Zeitfunktionen demonstriert.

Listing 3.46: Zeitfunktionen einsetzen

```
Sub MinutenExtrahieren()
Dim s As String

Debug.Print "Funktion Now:    " & Now
Debug.Print "Funktion Time:    " & Time
s = InputBox("Geben Sie eine Uhrzeit ein!", _
"Uhrzeit", Time)
Debug.Print "Eingegebene Uhrzeit: " & s
Debug.Print "Funktion Minutes:  " & Minute(s)
Debug.Print "Funktion Hour:    " & Hour(s)
Debug.Print "Funktion Seconds:  " & Second(s)
End Sub
```

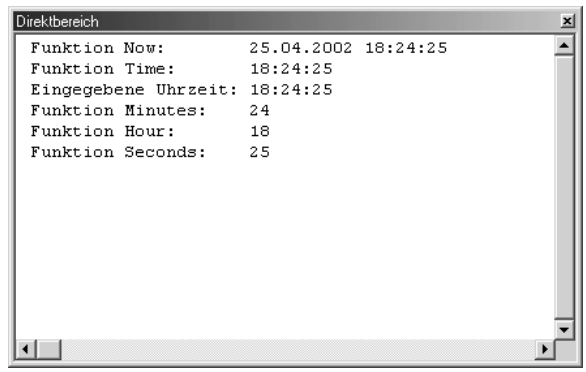


Abbildung 3.30: Die Ergebnisse der Zeitfunktionen

Farbfunktionen verwenden

Um einem Formularfeld eine bestimmte Farbe zuzuweisen, können Sie mit der Funktion `QBColor` arbeiten. Dieser Funktion müssen Sie einen Farbwert, der über einen eindeutigen Index festgelegt ist, zuweisen. Die Zuordnung von Farbindex zu Farbe können Sie der Tabelle 3.7 entnehmen.

Tabelle 3.7: Die Farbindizes der Funktion `QBColor`

Index	Farbe
0	Schwarz
1	Blau
2	Grün
3	Cyan
4	Rot
5	Magenta
6	Gelb
7	Weiß
8	Grau
9	Hellblau
10	Hellgrün
11	Hellcyan
12	Hellrot
13	Hellmagenta
14	Hellgelb
15	Leuchtend weiß

Um diese Funktion an einem praktischen Beispiel zu demonstrieren, werden Sie in einem Formular ein bestimmtes Feld (UMSATZ) überwachen. Dabei wechseln Sie die Farbe der Schrift bzw. die Farbe des Feldhintergrundes, je nach Umsatzhöhe.

Folgende Definitionen treffen Sie dabei:

- Umsatz > 35.000: Hintergrundfarbe Hellgelb und Schriftfarbe Blau
- Umsatz < 35.000: Hintergrundfarbe Weiß und Schriftfarbe Schwarz

Um diese automatische Einfärbung zu hinterlegen, befolgen Sie die nächsten Arbeitsschritte:

1. Öffnen Sie das Formular PERSONAL aus der Datenbank SPRACHELEMENTE.MDB in der Entwurfsansicht.
2. Klicken Sie mit der rechten Maustaste auf eine freie Fläche im Formular und wählen den Befehl EIGENSCHAFTEN aus dem Kontextmenü.

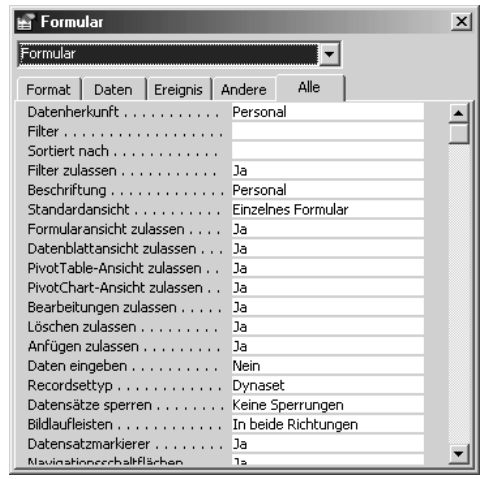


Abbildung 3.31: Den Eigenschaftendialog aufrufen

3. Wählen Sie aus dem Dropdown-Feld den Eintrag FORMULAR.
4. Wechseln Sie auf die Registerkarte EREIGNIS.
5. Klicken Sie im Feld BEIM ANZEIGEN ganz rechts auf das Symbol mit den drei Punkten.
6. Wählen Sie im Dialog GENERATOR WÄHLEN den Eintrag CODE-GENERATOR.
7. Bestätigen Sie Ihre Wahl mit OK.
8. Erfassen Sie jetzt im Code-Bereich folgendes Ereignismakro:

```
Private Sub Form_Current()
If Me!Umsatz > 35000 Then
 Me!Umsatz.ForeColor = QBColor(1) 'Schrift Blau
 Me!Umsatz.BackColor = QBColor(14) 'Hintergrund HellGelb
 Debug.Print Me!Umsatz.Value
Else
 Me!Umsatz.ForeColor = QBColor(0) 'Schrift Schwarz
 Me!Umsatz.BackColor = QBColor(7) 'Hintergrund Weiß
 Debug.Print Me!Umsatz.Value
End If
End Sub
```

Listing 3.47: Die Funktion `QBColor` anwenden

Abbildung 3.32: Ein Ereignis einstellen

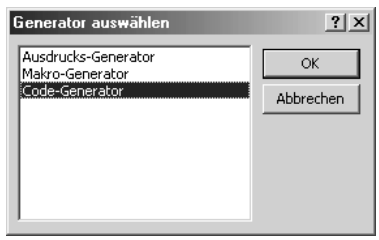


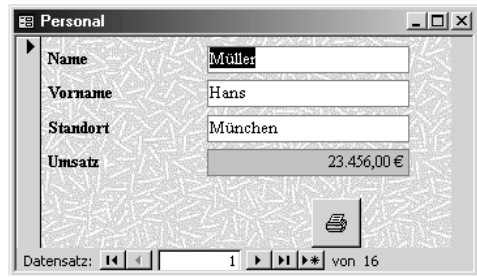
Abbildung 3.33: Den Code-Generator starten

9. Speichern Sie diese Änderung und schließen das noch geöffnete Formular.

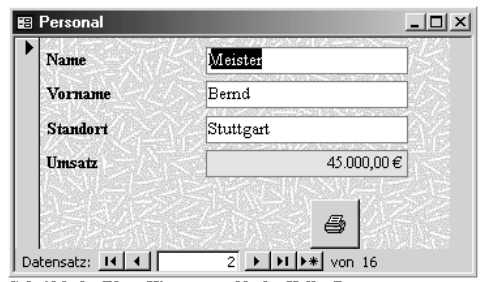
Mit der Eigenschaft `ForeColor` können Sie die Farbe für Text in einem Steuerelement angeben.

Mit der Eigenschaft `BackColor` können Sie die Farbe im Inneren des Steuerelements eines Formulars oder Bereichs angeben.

Rufen Sie nun das Formular mit einem Doppelklick auf und blättern nacheinander ein paar Sätze durch.



Schriftfarbe Schwarz, Hintergrundfarbe Weiß



Schriftfarbe Blau, Hintergrundfarbe Hellgelb

Abbildung 3.34: Eingabefelder je nach Wert färben

Erfahren Sie mehr zu den Themen Formulare und Ereignisse in den Kapiteln 7 und 9 dieses Buches.

Werte aus Liste auswählen

Mithilfe der Funktion `Choose` können Sie einen Wert aus einer Liste über einen Index ermitteln und ausgeben.

Im folgenden Beispiel haben Sie sich einige typische Sätze, die Sie jeden Tag in Briefen und E-Mails schreiben, zusammen in eine Funktion geschrieben.

```
Function Auswahl(I As Integer)
 Auswahl = Choose(I, "Sehr geehrte Damen und Herren,", _
"Liebe Anwender,", "Hallo,", _
"Mit freundlichen Grüßen",
"Viele Grüße", "Gruß")
End Function
```

Listing 3.48: Die Funktion `Choose` einsetzen

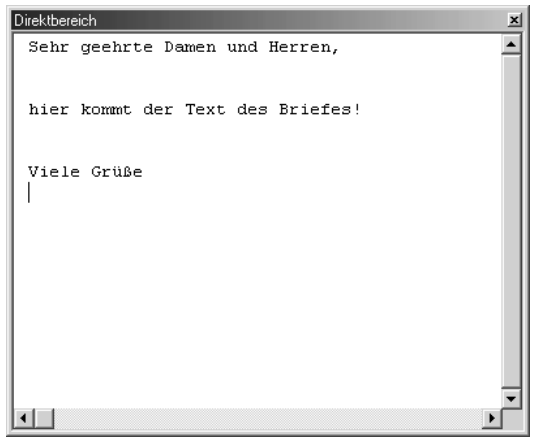
Rufen Sie nun die Funktion auf, indem Sie den gewünschten Index übergeben. So liefert Ihnen der Index `1` den Text »Sehr geehrte Damen und Herren«, der Index `2` den Text »Liebe Anwender«, usw.

```
Sub Aufruf()
s = Auswahl(1)
Debug.Print s
Debug.Print Chr(13)
Debug.Print "hier kommt der Text des Briefes!"
```

Listing 3.49: Den Index an die Funktion `Choose` übergeben

```
Debug.Print Chr(13)
s = Auswahl(5)
Debug.Print s
End Sub
```

Abbildung 3.35: Das Ergebnis im Direktbereich



Ganzzahligen Wert extrahieren

Um einen ganzzahligen Wert aus einer Zahl herauszufiltern, können Sie die Funktion `Fix` bzw. `Int` einsetzen. Bei negativen Zahlen ermittelt die Funktion `Int` die erste ganze negative Zahl, die kleiner oder gleich der übergebenen Zahl ist, während die `Fix`-Funktion die erste negative ganze Zahl liefert, die größer oder gleich der übergebenen Zahl ist.

Sehen Sie sich dazu das folgende Beispiel an.

Listing 3.50: Die Funktionen `Int` und `Fix`

```
Sub GanzzahligeWerteErmitteln()
Const Zahl1 = 12.67
Const Zahl2 = 49.9
Const Zahl3 = -58.78

Debug.Print "Zahl1 " & Zahl1
Debug.Print "Zahl2 " & Zahl2
Debug.Print "Zahl3 " & Zahl3
Debug.Print Chr(13)

Debug.Print "FIX - Zahl1 " & Fix(Zahl1)
Debug.Print "INT - Zahl1 " & Int(Zahl1)
Debug.Print Chr(13)

Debug.Print "FIX - Zahl2 " & Fix(Zahl2)
Debug.Print "INT - Zahl2 " & Int(Zahl2)
Debug.Print Chr(13)

Debug.Print "FIX - Zahl3 " & Fix(Zahl3)
Debug.Print "INT - Zahl3 " & Int(Zahl3)
End Sub
```

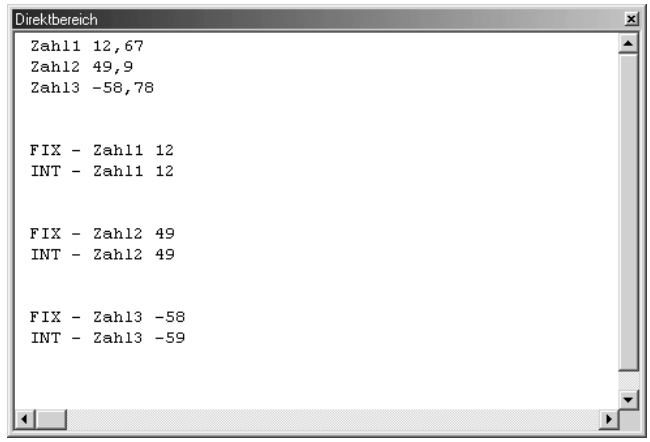


Abbildung 3.36: Das Ergebnis im Direktbereich

Zinsbelastung errechnen

Wenn Sie beabsichtigen, einen Kredit aufzunehmen und dabei Ihre Zinsbelastung errechnen möchten, dann hilft Ihnen dabei die Funktion `Ipmt`. Diese Funktion hat folgende Syntax:

```
IPmt(rate, per, nper, pv, fv, type)
```

Im Argument `rate` geben Sie den Zinssatz ein, den Sie in diesem Zeitraum aufbringen müssen. Rechnen Sie diese Angabe auf den monatlichen Zinssatz herunter.

Im Argument `per` bzw. `nper` geben Sie den Zeitraum in Monaten an, über den Sie den Kredit zurückbezahlen.

Das Argument `pv` steht für die Kreditsumme, die Sie aufnehmen möchten.

Im Argument `fv` geben Sie den Endstand des Kredits an. Diesen setzen Sie standardmäßig auf den Wert `0`, da dies die Kredithöhe nach der letzten Zahlung ist.

Im letzten Argument `Type` geben Sie an, wann Zahlungen fällig sind. Bei `0` sind die Zahlungen am Ende eines Zahlungszeitraums fällig, bei `1` zu Beginn des Zahlungszeitraums. Wird der Wert nicht angegeben, so wird `0` angenommen.

Im nächsten Beispiel gehen Sie von folgenden Ausgangsvoraussetzungen aus:

Kreditsumme: 10.000 €

Zinssatz/Jahr: 10%

Anzahl der Abzahlungsmonate: 24

Bauen Sie diese Informationen in das folgende Makro ein.

Listing 3.51: Die Zinsbelastung errechnen

```
Sub ZinsSummeErrechnen()
Dim EndWert As Currency
Dim Kreditsumme As Currency
Dim JahresZins As Single
Dim Anz_Zahlungen As Integer
Dim Wann As Boolean
Dim Zeitraum As Integer
Dim Zinszahlung As Currency
Dim GesZins As Currency
Const Beginn = 0, Ende = 1

EndWert = 0
Kreditsumme = InputBox("Wie hoch ist die Kreditsumme?")
JahresZins = InputBox("Wie hoch ist der Jahreszins?")
If JahresZins > 1 Then JahresZins = JahresZins / 100
Anz_Zahlungen = InputBox("Wie viele Monatszahlungen?")
Wann = MsgBox _
("Erfolgen Zahlungen am Monatsende?", vbYesNo)

If Wann = vbNo Then Wann = Ende Else Wann = Beginn

For Zeitraum = 1 To Anz_Zahlungen
 Zinszahlung = IPmt(JahresZins / 12, Zeitraum, _
 Anz_Zahlungen, -Kreditsumme, EndWert, Wann)
 GesZins = GesZins + Zinszahlung
Next Zeitraum

MsgBox "Sie zahlen insgesamt " & GesZins & Chr(13) & _
 "Zinsen für diesen Kredit."
End Sub
```

Internen Zinsfuß errechnen

Zur Beurteilung der Wirtschaftlichkeit von Investitionen wird sehr oft die interne Zinsfuß-Methode angewendet. Darunter versteht man die Rendite, die durch Investitionen in einem Unternehmen erzielt wird.

Diese Methode können Sie mithilfe der VBA-Funktion `IRR` durchführen. Diese Methode hat folgende Syntax:

```
IRR(values()[, guess])
```

Im Argument `values` geben Sie die Cash-Flow-Werte (Aus- und Einzahlungen) an. Unter diesen Werten muss mindestens ein negativer Wert (Zahlungsausgang) und ein positiver Wert (Zahlungseingang) enthalten sein.

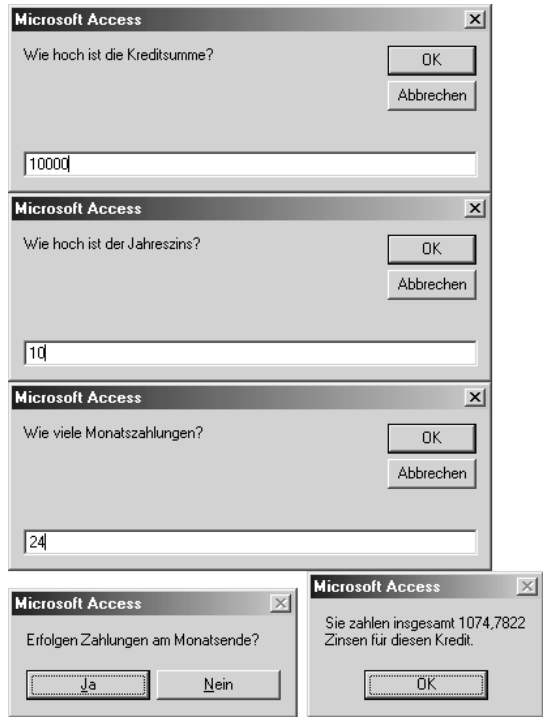


Abbildung 3.37: Die Gesamtbelastung der Zinsen

Unter dem Argument `guess` geben Sie einen von Ihnen geschätzten Wert ein. Wird der Wert nicht angegeben, so ist `guess` gleich 0,1 (10 Prozent).

Beim nächsten Beispiel gehen Sie von folgenden Ausgangsdaten aus:

- Anfangskosten von –90.000 €
- Cash-Flows der Folgejahre: 18.000 €, 20.000 €, 25.500 €, 29.600 € und 31.980 €
- Ihr geschätzter interner Zinsfuß lautet: 10%.

Setzen Sie diese Informationen in das folgende Makro ein.

Listing 3.52: Den internen Zinsfuß berechnen

```
Sub InternerZinsfuss()
Dim Schätz As Single
Dim IntZins As Single
Static Werte(6) As Double

Schätz = 0.
Werte(0) = -90000
Werte(1) = 18000: Werte(2) = 20000
Werte(3) = 25500: Werte(4) = 29600
Werte(5) = 31980
```

```
IntZins = IRR(Werte(), Schätz) * 100
MsgBox _
"Der interne Zinsfuß beträgt " & _
 Format(IntZins, "#.00") & " Prozent."
End Sub
```

Abbildung 3.38: Der interne Zinsfuß lautet: 10,83%

Abschreibungen berechnen

Möchten Sie die Abschreibung eines Vermögenswertes über einen bestimmten Zeitraum mithilfe der arithmetischen Abschreibungsmethode ermitteln, dann setzen Sie die VBA-Funktion `SLN` ein.

Diese Funktion hat folgende Syntax:

```
SLN(cost, salvage, life)
```

Im Argument `cost` geben Sie die Anschaffungskosten des Anlageguts an.

Im Argument `salvage` geben Sie den Wert des Anlagegutes am Ende der Abschreibung an. Diesen Wert können Sie auf 0 setzen.

Unter dem Argument `life` geben Sie die Anzahl der Abschreibungsmonate an.

Im nächsten Beispiel gehen Sie von folgenden Ausgangsvoraussetzungen aus:

- Anschaffungskosten des Anlagegutes: 1.000 €
- Wert am Ende der Abschreibung: 0
- Dauer der Abschreibung: 3 Jahre = 36 Monate

Setzen Sie diese Informationen in das folgende Makro ein.

Listing 3.53: Die lineare Abschreibung berechnen

```
Sub AbschreibungBerechnen()
Dim InvestSumme As Currency
Dim Restwer As Currency
Dim Afa As Currency
Dim AfaMonate As Integer

InvestSumme = InputBox("Investsumme angeben")
Restwert = InputBox _
("Wert des Vermögensgegenstands am Ende " & _
   "der Nutzungsdauer?")
AfaMonate = InputBox("Nutzungsdauer in Monaten?")
Nutzung = AfaMonate / 12
```

```
Afa = SLN(InvestSumme, Restwert, Nutzung)
MsgBox "Die Abschreibung ist " & Format(Afa, "#.00") & _
" pro Jahr."
End Sub
```

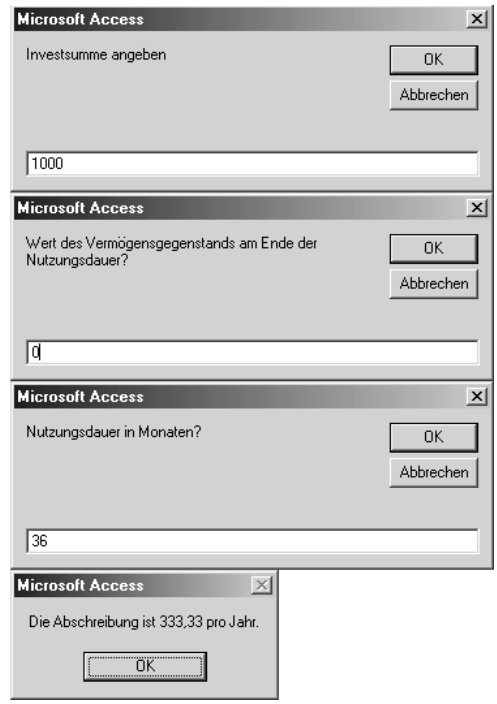


Abbildung 3.39: Lineare Abschreibung ermitteln

Neben der Funktion `SLN` *gibt es eine weitere Funkton namens* `DDB`*, mit deren Hilfe Sie weitere Abschreibungstypen wie die degressive Abschreibung abhandeln können. Allerdings scheint diese Funktion einen Bug zu haben, da die degressive Abschreibung nicht den zu erwartenden Ergebnissen entspricht.*

3.5 Umwandlungsfunktionen

Hin und wieder müssen Sie einzelne Datentypen in andere Datentypen umwandeln. Für disen Zweck gibt eine ganze Reihe von Umwandlungsfunktionen.

Funktion	Rückgabetyp	Bereich des Arguments *Ausdruck*
`CBool`	`Boolean`	Eine gültige Zeichenfolge oder ein gültiger numerischer Ausdruck
`CByte`	`Byte`	0 bis 255

Tabelle 3.8: Typumwandlungsfunktionen (Quelle: Online-Hilfe)

Tabelle 3.8: Typumwandlungsfunktionen (Quelle: Online-Hilfe) (Forts.)

Funktion	Rückgabetyp	Bereich des Arguments *Ausdruck*
CCur	Currency	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807
CDate	Date	Ein beliebiger gültiger Datumsausdruck.
CDbl	Double	-4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte
CDec	Decimal	Für skalierte Ganzzahlen, d.h. Zahlen ohne Dezimalstellen. Für Zahlen mit 28 Dezimalstellen gilt der Bereich +/-7,9228162514264337593543950335.
CInt	Integer	-32.768 bis 32.767; Nachkommastellen werden gerundet.
CLng	Long	-2.147.483.648 bis 2.147.483.647; Nachkommastellen werden gerundet.
CSng	Single	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte
CVar	Variant	Numerische Werte im Bereich des Typs Double. Nichtnumerische Werte im Bereich des Typs String
CStr	String	Die Rückgabe für CStr hängt vom Argument Ausdruck ab.

Es folgen nun ein paar typische Beispiele, in denen Typumwandlungsfunktionen eingesetzt werden.

Die Typumwandlungsfunktion CBool

Die Typumwandlungsfunktion CBool wird verwendet, um einen Ausdruck in einen Wert vom Typ Boolean umzuwandeln. Liefert der Ausdruck einen Wert ungleich Null, so gibt CBool den Wert True zurück, andernfalls False.

Im nächsten Beispiel soll geprüft werden, ob in der Datenbank SPRACHELEMENTE.MDB das Formular PERSONAL gerade geöffnet ist. Diese Aufgabe lösen Sie, indem Sie zuerst eine Funktion schreiben, die Sie in der folgenden Auflistung einsehen können.

```
Function FormularGeöffnet(FormularName As String) _
 As Boolean

 FormularGeöffnet = _
 CBool(SysCmd(acSysCmdGetObjectState, acForm, _
 FormularName))
End Function
```

Listing 3.54: Funktion zum Prüfen eines geöffneten Formulars

Mit der Methode `SysCmd` können Sie eine Statusanzeige oder optional angegebenen Text in der Statusleiste anzeigen, Informationen zu Microsoft Access und den zugehörigen Dateien oder den Status eines angegebenen Datenbankobjekts zurückgeben.

Diese Methode hat folgende Syntax:

```
Ausdruck.SysCmd(Aktion, Argument2, Argument3)
```

Unter dem Argument `Aktion` übergeben Sie der Methode eine fest definierte Konstante. Einige mögliche Konstanten sind dabei:

- `AcSysCmdAccessDir` gibt den Namen des Verzeichnisses zurück, in dem sich MSACCESS.EXE befindet.
- `AcSysCmdAccessVer`: Diese Konstante gibt die Versionsnummer von Microsoft Access zurück.
- `AcSysCmdClearStatus`: Diese Konstante gibt Informationen zum Status eines Datenbankobjekts zurück.
- `AcSysCmdGetObjectState`: Sie gibt den Status des angegebenen Datenbankobjekts zurück. Sie müssen die Argumente `Argument2` und `Argument3` angeben, wenn Sie für die Aktion diesen Wert verwenden.
- `AcSysCmdGetWorkgroupFile`: Sie gibt den Pfad zur Arbeitsgruppendatei (SYSTEM.MDW) zurück.
- `AcSysCmdIniFile`: Die Konstante gibt den Namen der von Microsoft Access verwendeten INI-Datei zurück.
- `AcSysCmdInitMeter`: Sie initialisiert die Statusanzeige. Sie müssen die Argumente `Argument2` und `Argument3` angeben, wenn Sie diese Aktion verwenden.
- `AcSysCmdProfile`: Sie gibt die Einstellung von `/profile` zurück, die der Benutzer angegeben hat, wenn er Microsoft Access von der Befehlszeile aus gestartet hat.
- `AcSysCmdRemoveMeter`: Die Konstante entfernt die Statusanzeige.
- `AcSysCmdRuntime`: Sie gibt den Wert `True (-1)` zurück, wenn eine Laufzeitversion von Microsoft Access ausgeführt wird.

- `AcSysCmdSetStatus`: Sie legt den Text in der Statusleiste auf das Argument `Text` fest.
- `AcSysCmdUpdateMeter`: Die Konstante aktualisiert die Statusanzeige mit dem angegebenen Wert. Sie müssen das Argument `Text` angeben, wenn Sie diese Aktion verwenden.

Beim `Argument2` handelt es sich um einen optionalen `Variant`-Wert. Dieses Argument steht für einen Zeichenfolgenausdruck, der den Text angibt, der linksbündig in der Statusleiste angezeigt wird. Dieses Argument ist erforderlich, wenn das Argument `Aktion` auf `acSysCmdInitMeter`, `acSysCmdUpdateMeter` oder `acSysCmdSetStatus` festgelegt wurde.

In unserem Beispiel haben Sie die Einstellung `acSysCmdGetObjectState` verwendet. In diesem Fall müssen Sie beispielsweise eine der vorgegebenen integrierten Konstanten verwenden:

- `AcTable`: eine Tabelle
- `AcQuery`: eine Abfrage
- `AcForm`: ein Formular
- `AcReport`: ein Bericht
- `AcMacro`: ein Makro
- `AcModule`: ein Modul
- `AcDataAccessPage`: eine Datenzugriffsseite
- `AcDiagram`: ein Diagramm

Beim `Argument3` handelt es sich um einen optionalen `Variant`-Wert. Dahinter verbirgt sich ein numerischer Ausdruck, der die Statusanzeige steuert. Dieses Argument ist erforderlich, wenn das Argument `Aktion` auf `acSysCmdInitMeter` festgelegt wurde. Für andere Werte des Arguments `Aktion` ist es nicht anzugeben.

So können Sie beispielsweise über die Methode `SysCMD` ermitteln, ob ein Formular geöffnet ist. Rufen Sie die `SysCmd`-Methode mit dem auf `acSysCmdGetObjectState` festgelegten Argument `Aktion` und den Argumenten `Objekttyp` und `Objektname` auf, um den Status des Formulars DATENBANKOBJEKT zurückzugeben.

Ein Objekt kann einen von vier möglichen Status aufweisen: nicht geöffnet bzw. nicht vorhanden, geöffnet, neu sowie geändert, jedoch noch nicht gespeichert.

Was jetzt noch fehlt, ist die Prozedur, die unsere Funktion `FormularGeöffnet` aufruft. Erfassen Sie jetzt diese Prozedur.

Listing 3.55: Formular öffnen, wenn noch nicht geöffnet

```
Sub FormularCheck()
Dim b As Boolean

b = FormularGeöffnet("Personal")
If b = True Then
MsgBox "Das Formular ist geöffnet!"
Else
MsgBox "Das Formular ist noch nicht geöffnet!"
DoCmd.OpenForm "Personal", acViewNormal
End If
End Sub
```

Die Funktion `FormularGeöffnet` liefert Ihnen einen Wahrheitswert `True`, wenn das Formular PERSONAL bereits geöffnet ist. Wenn nicht, dann liefert die Funktion den Wert `False`. In diesem Fall wenden Sie die Methode `OpenForm` an, um das Formular zu öffnen. Dieser Methode müssen Sie sowohl den Objekttyp als auch den Objektnamen bekannt geben.

Die Typumwandlungsfunktion CDbl

Mithilfe der Typumwandlungsfunktion `CDbl` können Sie beispielsweise aus einem String, der einen Wert enthält, alle Punkte entfernen. Dieser String wird dann in einen Ausdruck des Datentyps `Double` umgewandelt.

Im nächsten Beispiel werden aus einem String, der Tausenderpunkte enthält, diese Punkte entfernt und in einer Variablen vom Typ `Double` gespeichert.

Listing 3.56: Tausenderpunkte aus einem String entfernen

```
Sub TausenderpunkteRaus()
Dim zk As String
Dim Betrag As Double

zk = "128.235.311,45"
Debug.Print "Zeichenkette vorher: " & zk
Betrag = CDbl("128.235.311,45")
Debug.Print "Zeichenkette nachher: " & Betrag
End Sub
```

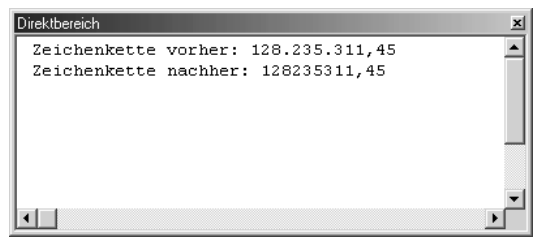


Abbildung 3.40: Einen String in einen `Double`-Datentyp umwandeln

Die Typumwandlungsfunktion CDate

Mithilfe der Typumwandlungsfunktion `CDate` können Sie Datumsangaben in Zeichenfolgen in echte Datumsangaben wandeln. So wird beispielsweise aus der Zeichenfolge `13. Oktober 2002` das Datum `13.10.2002`.

Sehen Sie weitere Beispiele im folgenden Listing.

Listing 3.57: Datumswerte aus Strings herstellen

```
Sub DatümerWandeln()
Const Datum1 = "11/25/1998 8:30 AM"
Const Datum2 = "12. Februar 2002"
Const Datum3 = "Januar, 2002"


Debug.Print CDate(Datum1)
Debug.Print CDate(Datum2)
Debug.Print CDate(Datum3)
End Sub
```

Abbildung 3.41: Einen String in korrekte Datumsangaben wandeln

Die Typumwandlungsfunktion CLng

Mithilfe der Funktion `CLng` können Sie einen Wert in einen Datentyp `Long` umwandeln. Dabei werden eventuell existierende Nachkommastellen gerundet.

Im folgenden Beispiel werden genau 20 Zufallszahlen gebildet. Diese werden mithilfe der Funktion `CLng` in den richtigen Datentyp gewandelt.

Listing 3.58: Zufallszahlen bilden

```
Sub ZufallszahlenErzeugen()
Dim i As Integer
Dim l As Long

For i = 1 To 20
 l = CLng(99999 * Rnd + 1)
 Debug.Print "Zufallszahl " & Format(i, "00") & _
 " lautet " & Format(l, "0,##")
Next i
End Sub
```

Über die Anweisung `CLng(99999 * Rnd + 1)` bilden Sie eine Zufallszahl zwischen 1 und 9999 und weisen dieser Zahl den Datentyp `Long` zu. Über die Funktion `Format` bringen Sie die Werte in die gewünschte Form.

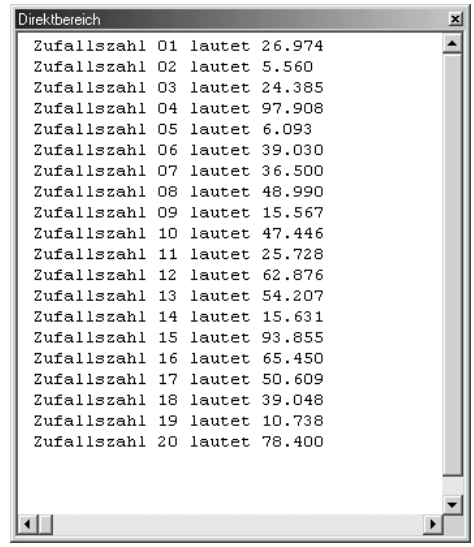


Abbildung 3.42: Zufallszahlen bilden

Die Typumwandlungsfunktion CStr

Die Funktion `CStr` kommt dann zum Einsatz, wenn Sie einen numerischen Datentyp in einen `String`-Datentyp umwandeln möchten.

Im folgenden Beispiel zerlegen Sie ein Datum in seine Einzelteile, also in die Angaben Tag, Monat und Jahr und setzen es danach wieder in einer etwas anderen Form zusammen. So soll aus der Datumsangabe `1.1.2002` die Datumsangabe `01.01.2002` gemacht werden. Es sollen demnach Nullen an den Stellen eingefügt werden, an denen die Tagesangabe bzw. die Monatsangabe einstellig ist. Außerdem soll die Jahresanzeige auf zwei Stellen gekürzt werden. Wie Sie diese Aufgabe lösen können, erfahren Sie im folgenden Makro.

```
Sub UmwandlungsfunktionInStr()
Dim s_Tag As String
Dim s_Monat As String
Dim S_Jahr As String

Eingabe = "1.1.2002"
```

Listing 3.59: Datum zerlegen und ins gewünschte Format überführen

```
Debug.Print _
"Das Datum vor der Umwandlung lautet: " & Eingabe
Tag = Day(¢)
Monat = Month(¢)
Jahr = Year(¢)


If Tag < 10 Then
 s_Tag = "0" & CStr(Tag)
Else
 s_Tag = CStr(Tag)
End If


If Monat < 10 Then
 s_Monat = "0" & CStr(Monat)
Else
 s_Monat = CStr(Monat)
End If


S_Jahr = Right(CStr(Jahr), 2)


Eingabe = CStr(s_Tag & "." & s_Monat & "." & S_Jahr)
Debug.Print _
"Das Datum nach der Umwandlung lautet: " & Eingabe
End Sub
```

Zerlegen Sie die Variable `Eingabe` zunächst in Ihre Bestandteile. Dazu setzen Sie die Funktionen `Day`, `Month` und `Year` ein. Dabei werden diese Informationen in einem numerischen Datentyp zurückgegeben. Damit Sie jetzt eine führende Null beim Tag bzw. beim Monat einfügen können, müssen Sie diese numerischen Werte über die Funktion `CStr` in einen String überführen.

Um die Jahresangabe zweistellig zu bekommen, setzen Sie die Funktion `Right` ein, die beginnend vom rechten Ende einer Variablen in der gewünschten Länge Zeichen in einen neuen String überträgt.

Setzen Sie am Ende die Einzelteile wieder in der gewünschten Form zusammen und beispielsweise als Trennzeichen den Punkt bzw. das Zeichen »/« ein.

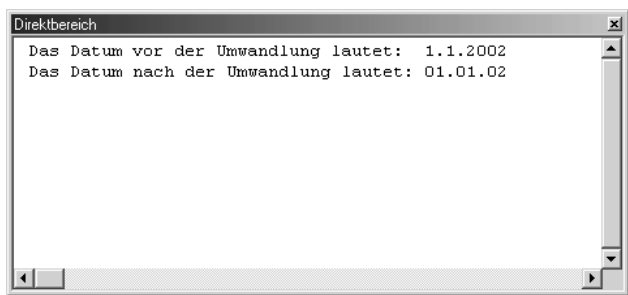


Abbildung 3.43: Datumsangaben in das gewünschte Format bringen

Die Funktion Val

Mithilfe der Funktion `Val` können Sie aus einer Zeichenfolge in ihr enthaltene numerische Werte herausholen.

Im folgenden Praxisbeispiel sollen aus einer Zeichenfolge nur die numerischen Werte extrahiert werden. Für diese Aufgabe schreiben Sie eine Funktion und übergeben dieser die komplette Zeichenfolge, die sowohl numerische als auch alphanumerische Zeichen enthalten kann. Als Rückgabe soll die Funktion nur die numerischen Zeichen zurückliefern.

Listing 3.60: Numerische Zeichen extrahieren

```
Function NumZeichExtra(Z As String) As String
Dim i As Integer
Dim s As String

If IsNull(Z) = False Then
 For i = 1 To Len(Z)
  If Mid(Z, i, 1) >= "0" And Mid(Z, i, 1) <= "9" Then
    s = s & Mid(Z, i, 1)
   End If
  Next i
End If
NumZeichExtra = s
End Function
```

Prüfen Sie zuerst einmal, ob überhaupt eine Zeichenfolge an die Funktion übergeben wurde. Dazu setzen Sie die Funktion `IsNull` ein. Danach ermitteln Sie mithilfe der Funktion `Len` die Anzahl der übergebenen Zeichen. Sie durchlaufen eine Schleife, in der Sie Zeichen für Zeichen überprüfen und in die Variable `s` einfügen, sofern das jeweilige Zeichen einen Wert zwischen 0 und 9 aufweist. Geben Sie am Ende der Funktion den so manipulierten String zurück an die aufrufende Prozedur.

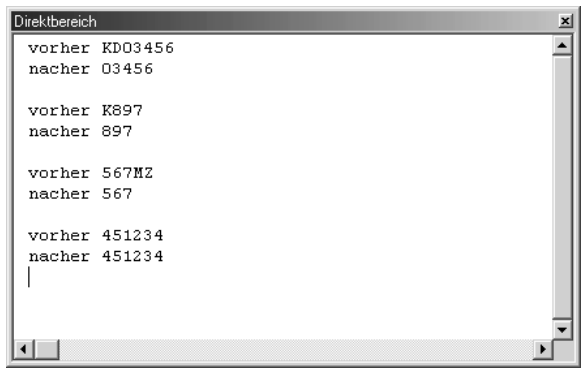
Um diese Funktion richtig zu testen, basteln Sie sich ein Makro, welches testweise mehrere Zeichenfolgen nacheinander an die Funktion übergibt.

Listing 3.61: Textzeichen eliminieren

```
Sub TextzeichenRaus()
Dim s1 As String
Dim s2 As String
Dim s3 As String
Dim s4 As String

s1 = "KD03456"
s2 = "K897"
s3 = "567MZ"
s4 = "451234"

Debug.Print "vorher " & s1
s1 = NumZeichExtra(s1)
Debug.Print "nacher " & s1 & Chr(13)
```

```
Debug.Print "vorher " & s2
s2 = NumZeichExtra(s2)
Debug.Print "nacher " & s2 & Chr(13)

Debug.Print "vorher " & s3
s3 = NumZeichExtra(s3)
Debug.Print "nacher " & s3 & Chr(13)

Debug.Print "vorher " & s4
s4 = NumZeichExtra(s4)
Debug.Print "nacher " & s4
End Sub
```

Abbildung 3.44: Das Ergbnis enthält nur noch numerische Zeichen



3.6 Die IS-Funktionen in VBA

In VBA stehen Ihnen einige wichtige Funktionen zur Verfügung, mit denen Sie prüfen können, welcher Datentyp vorliegt. Auf diese Weise können Sie auf Nummer sicher gehen, wenn Sie Werte weiterverarbeiten möchten.

Die Funktion IsArray

Mithilfe der Funktion `IsArray` prüfen Sie, ob es sich bei der angesprochenen Variablen um ein Datenfeld (Array) handelt.

Um diese Funktion zu üben, schreiben Sie ein Makro, in dem Sie festlegen, wie groß ein Datenfeld angelegt werden soll. Diese Information übergeben Sie einer Funktion, die das Datenfeld in der gewünschten Größe anlegt und an die aufrufende Prozedur zurückliefert.

Listing 3.62: Typprüfung auf Array vornehmen

```
Function ArrayBilden(Größe As Long) As Variant
ReDim K_Array(Größe)
Dim l As Long
```

```
  For l = LBound(K_Array, 1) To UBound(K_Array, 1)
   K_Array(l) = l
  Next l
  ArrayBilden = K_Array
End Function


Sub DynamischenArrayBilden()
Dim K_Array As Variant
Dim l As Long

  K_Array = ArrayBilden(10)
  If IsArray(K_Array) Then
    For l = LBound(K_Array, 1) To UBound(K_Array, 1)
      Debug.Print K_Array(l)
    Next l
  End If
End Sub
```

In der Zeile `K_Arry=ArrayBilden(10)` rufen Sie die Funktion `ArrayBilden` auf und übergeben ihr den Wert `10`. Damit legt die Funktion `ArrayBilden` ein Datenfeld mit genau zehn Datenfeldern an, eigentlich ja elf, da die 0 mitgezählt wird. Nachdem die Funktion an die aufrufende Prozedur das angelegte Datenfeld zurückmeldet, prüfen Sie über die Funktion `IsArray`, ob die Rückgabe der Funktion auch den richtigen Datentyp, nämlich ein Datenfeld, liefert. Wenn ja, dann setzen Sie eine Schleife auf, in der das Datenfeld ausgelesen wird. Setzen Sie dazu den Index `l` zu Beginn der Schleife mithilfe der Funktion `LBound` auf den ersten Eintrag des Datenfeldes und arbeiten Sie sich dann bis zum letzten Feld des Datenfeldes vor, welches Sie über die Funktion `UBound` ermitteln. Geben Sie nach jeden Schleifendurchlauf den Inhalt des jeweiligen Datenfeldes im Direktbereich über die Anweisung `Debug.Print` aus.

Abbildung 3.45: Einen Array in den Direktbereich auslesen

Lernen Sie später in diesem Kapitel mehr zum Thema Arrays.

Die Funktion IsDate

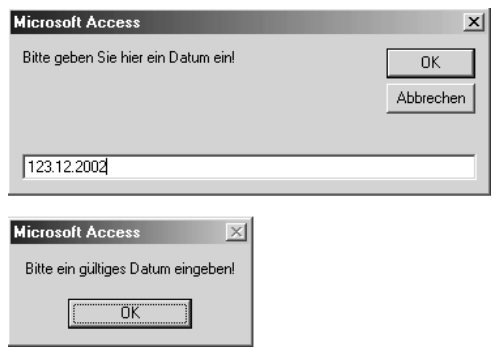
Mithilfe der Funktion `Isdate` überprüfen Sie, ob ein gültiger Datumswert vorliegt. Da eine detaillierte Beschreibung dieser Funktion bereits zu Beginn des Kapitels besprochen wurde, folgt hier nur noch ein typisches Beispiel, wie Sie einen Anwender dazu zwingen können, ein korrektes Datum in eine Inputbox einzutragen. Sehen Sie dazu die Lösung im folgenden Listing.

Listing 3.63: Datumscheck in Inputbox vornehmen

```
Sub InputboxPrüfen()
Dim datum As String

Do While IsDate(datum) = False
  datum = InputBox _
  ("Bitte geben Sie hier ein Datum ein!")
  If IsDate(datum) = False Then MsgBox _
  "Bitte ein gültiges Datum eingeben!"
Loop
End Sub
```

Der Dialog wird solange aufgerufen, bis der Anwender ein gültiges Datum eingibt.

Abbildung 3.46: Eingegebenes Datum überprüfen



Die Funktionen IsEmpty und IsNull

Über die Funktion `IsEmpty` können Sie prüfen, ob eine Variable initialisiert wurde.

Mithilfe der Funktion `IsNull` können Sie ermitteln, ob ein Ausdruck gültige Daten enthält. Um den Unterschied beider Funktionen deutlich zu machen, starten Sie das nachfolgende Makro.

Listing 3.64: Der Unterschied zwischen `IsNull` und `IsEmpty`

```
Sub NullOderLeer()
Dim var As Variant

  Debug.Print "Leere Variable"
  Debug.Print "IsEmpty " & IsEmpty(var)
```

```
  Debug.Print "IsNull " & IsNull(var)
  Debug.Print Chr(13)

  Debug.Print "var = Null"
  var = Null
  Debug.Print "IsEmpty " & IsEmpty(var)
  Debug.Print "IsNull " & IsNull(var)
  Debug.Print Chr(13)

  Debug.Print "var = 0"
  var = 0
  Debug.Print "IsEmpty " & IsEmpty(var)
  Debug.Print "IsNull " & IsNull(var)
  Debug.Print Chr(13)

  Debug.Print "Leerstring "
  var = ""
  Debug.Print "IsEmpty " & IsEmpty(var)
  Debug.Print "IsNull " & IsNull(var)
  Debug.Print Chr(13)

  Debug.Print "var = Held"
  var = "Held"
  Debug.Print "IsEmpty " & IsEmpty(var)
  Debug.Print "IsNull " & IsNull(var)
  Debug.Print Chr(13)

  Debug.Print "var = 256"
  var = 256
  Debug.Print "IsEmpty " & IsEmpty(var)
  Debug.Print "IsNull " & IsNull(var)
End Sub
```

Die Funktion IsMissing

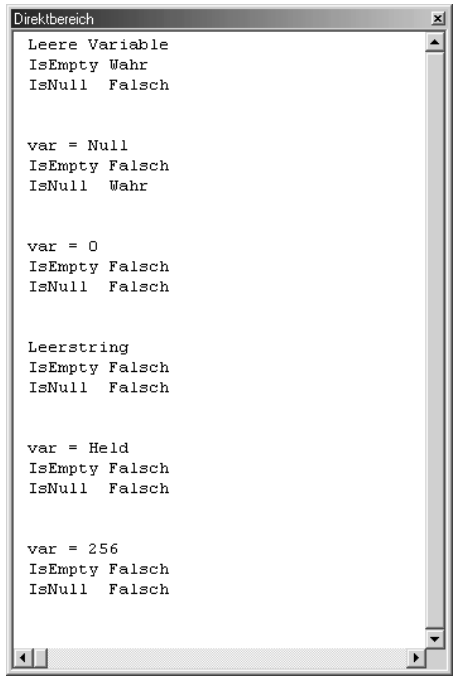
Wenn Sie eine Funktion aufrufen, die einen bzw. mehrere Argumente erwartet, dann können Sie mithilfe der Funktion `IsMissing` prüfen, ob alle optionalen Parameter auch übergeben wurden.

Für diese Funktion habe ich mir ein Beispiel überlegt, welches gerade auf fehlende Parameter bei Funktionen reagiert.

Stellen Sie sich vor, Sie stellen eine Rechnung. Übergeben Sie die Informationen wie Rechnungsnummer, Rechnungsdatum sowie das Zahlzieldatum an eine Funktion. Dabei möchten Sie die Funktion aber so programmieren, dass Sie darauf reagieren kann, wenn Sie das Zahldatum, also das dritte Argument nicht angeben. In diesem Fall soll automatisch auf das Rechnungsdatum genau 14 Tage dazu addiert werden.

Diese Aufgabe sieht umgesetzt wie folgt aus:

Abbildung 3.47: Verschiedene Rückmeldungen bei `IsEmpty` und `IsNull`



Listing 3.65: Auf nicht angegebene Argumente flexibel reagieren

```
Sub AufrufD()
Dim Datwert As Date

Debug.Print "Rechnungsnummer: K100, Rechnungsdatum: 30.3.2002
Zahldatum: 30.4.2002"
Datwert = Zahlung(K100, "30.3.2002", "30.4.2002")
Debug.Print "Das Zahldatum lautet: " & Datwert & Chr(13)


Debug.Print "Rechnungsnummer: K100, Rechnungsdatum: 30.3.2002
Zahldatum: nicht angegeben"
Datwert = Zahlung(K100, "30.3.2002")
Debug.Print "Das Zahldatum lautet: " & Datwert
End Sub


Function Zahlung _
(ByVal RechName As String, RechDat As Date, _
 Optional ByVal ZahlDatum As Variant) As Date

 If IsMissing(ZahlDatum) Then ZahlDatum = RechDat + 14
  Zahlung = ZahlDatum
End Function
```

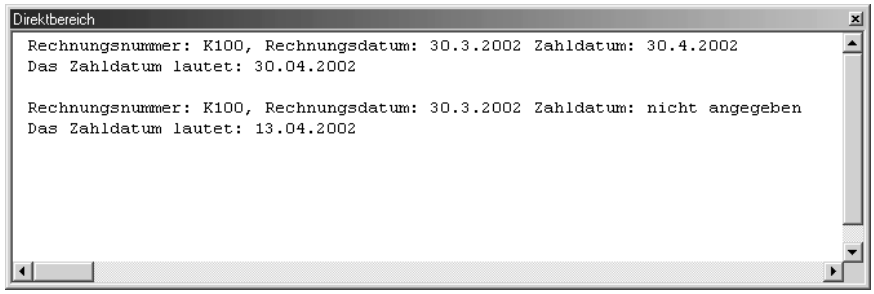


Abbildung 3.48: Die Funktion `IsMissing` einsetzen, um Zahlungsziele automatisch zu erhalten

Die Funktion IsObject

Mithilfe der Funktion `IsObject` können Sie beispielsweise überprüfen, ob sich ein bestimmtes Objekt in Ihrer Datenbank befindet. So wird diese Funktion im nächsten Beispiel eingesetzt, um zu prüfen, ob sich eine bestimmte Tabelle in der aktuellen Datenbank befindet. Nur dann soll diese Tabelle auch geöffnet werden. Diese Prüfroutine sieht wie folgt aus:

Listing 3.66: Befindet sich das Objekt TABELLE PERSONAL in der Datenbank?

```
Function Tabelleda(s As String) As Boolean
  On Error Resume Next
  Tabelleda = IsObject(CurrentDb.TableDefs(s))
End Function


Sub PrüfenAufTabelle()
Dim b As Boolean

If Tabelleda("Personal") = True Then
DoCmd.OpenTable "Personal"
 Else
MsgBox "Tabelle ist nicht vorhanden!"
End If
End Sub
```

Über die Methode `CurrentDb` haben Sie Zugriff auf die aktuelle geöffnete Datenbank. Mithilfe der Auflistung `TabelDefs` können Sie kontrollieren, ob sich die übergebene Tabelle in der Variablen `s`, in der Datenbank befindet. Wenn Ja, dann wenden Sie die Methode `OpenTable` an, um die gewünschte Tabelle zu öffnen.

3.7 Arbeiten mit Arrays

Wenn Sie viele Daten schnell verarbeiten müssen, dann bewährt sich der Einsatz von Arrays. Ein Array besteht aus einem Variablennamen und einem oder mehreren Indizes, die das einzelne Datenfeld eindeutig identifizieren.

Einfache Arrays

Wenn Sie vorher genau wissen, wie viele Daten in ein Array eingefügt werden sollen, können Sie dieses fest anlegen.

Im folgenden Beispiel werden die sieben Wochentage in ein Array eingelesen und über einen Index angesprochen und ausgegeben.

Listing 3.67: Array füllen und auslesen

```
Sub EinfacherArray()
Dim Woche As Variant
Dim Tag1 As String
Woche = Array("Montag", "Dienstag", "Mittwoch", "Donnerstag",
"Freitag", "Samstag", "Sonntag")

Select Case Weekday(Date, vbMonday)
Case 1
 Tag1 = Woche(0)
Case 2
 Tag1 = Woche(1)
Case 3
 Tag1 = Woche(2)
Case 4
 Tag1 = Woche(3)
Case 5
 Tag1 = Woche(4)
Case 6
Tag1 = Woche(5)
Case 7
 Tag1 = Woche(6)
End Select

Debug.Print "Heute ist " & Tag1
End Sub
```

Über die Funktion `Array` füllen Sie das Array namens `Woche`. Dabei schreiben Sie die einzelnen Tage getrennt durch Kommata und doppelte Anführungszeichen direkt in das Array.

Danach wenden Sie eine `Select case`-Anweisung an, um den aktuellen Tag auszuwerten. Die Funktion `Weekday` liefert Ihnen eine Zahl zwischen 1 und 7, welche für den entsprechenden Tag der Woche steht. Diese Zahl werten Sie aus und geben den dazugehörigen Array-Wert über die Vergabe des Indexes aus.

Im folgenden Beispiel durchlaufen Sie eine Schleife so oft, bis die Grenze des Arrays erreicht ist.

Listing 3.68: Schleifendurchläufe über Array steuern

```
Sub FesterArray()
Dim i As Long
Dim I_Array(1 To 100) As Long
```

```
  For i = 1 To UBound(I_Array)
    Debug.Print "Schleifendurchlauf " & i
  Next i
End Sub
```

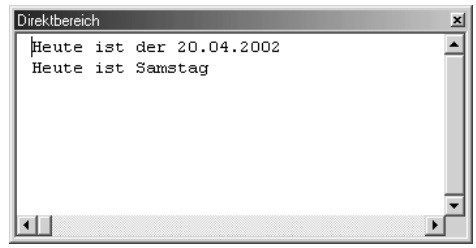


Abbildung 3.49: Arrays füllen und auslesen

Definieren Sie im ersten Schritt ein Array vom Typ `Long`. Dort geben Sie an, in welchen Grenzen sich das Array bewegen soll. Setzen Sie danach eine Schleife ein, die solange durchlaufen wird, bis der letzte mögliche Wert im Array `I_Array` erreicht wird. Mithilfe der Funktion `UBound` bekommen Sie gerade diesen größten verfügbaren Index für das Array.

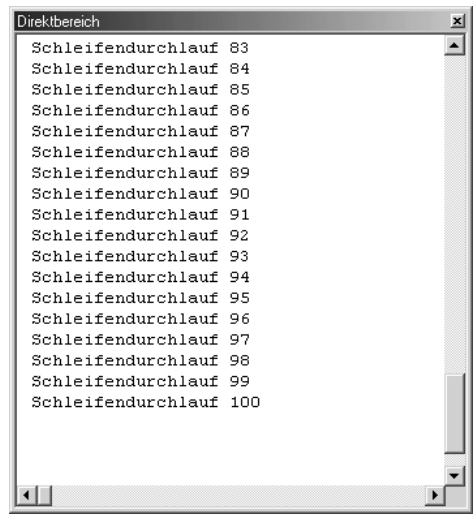


Abbildung 3.50: Fest definierte Schleifendurchläufe über Arrays definieren

Mehrdimensionale Arrays

Etwas komplexer wird es, wenn Sie mehrdimensionale Arrays füllen und wieder auslesen müssen. Im nachfolgenden Beispiel gehen Sie von der Tabelle PERSONAL in Abbildung 3.51 aus.

Erfassen Sie nun das folgende Makro:

Abbildung 3.51: Die Ausgangstabelle mit Personaldaten

Personal : Tabelle

Name	Vorname	Standort	Umsatz
Müller	Hans	München	23.456,00 €
Meister	Bernd	Stuttgart	45.000,00 €
Schmidt	Gustav	Hamburg	10.000,00 €
Reichert	Sonja	Köln	17.500,00 €
Teichmann	Werner	Saarbrücken	22.567,00 €
Fuchs	Hugo	München	50.000,00 €
Schiller	Franz	Hamburg	34.689,00 €
Igel	Sabine	Bonn	25.231,00 €
Baumstark	Hera	Stuttgart	59.561,00 €
Huber	Karl	Hannover	19.999,00 €
Walter	Barbara	Stuttgart	28.990,00 €
Kilian	Vera	München	51.983,00 €
Kuhn	Fritz	Bonn	18.750,00 €
Gans	Gustav	Trier	52.651,00 €
Paul	Peter	Köln	51.411,00 €
Müller	Ute	Stuttgart	38.790,00 €
			0,00 €

Datensatz: 17 von 17

Listing 3.69: Mehrdimensionales Array füllen und wieder auslesen

```
Sub MehrdimArraysFüllen()
Dim P_Array() As String
Dim i As Integer
Dim AnzSätze As Integer
Dim DBS As New ADODB.Recordset

On Error GoTo fehler
  Set Conn = CurrentProject.Connection
  Set DBS = New ADODB.Recordset


  With DBS
  .Open "Personal", Conn, adOpenKeyset, _
   adLockOptimistic
   Debug.Print "In der Tabelle befinden sich " & _
        DBS.RecordCount & " Datensätze"
   AnzSätze = DBS.RecordCount
  End With

ReDim P_Array(1 To AnzSätze, 1 To 2)

i = 1
For i = 1 To AnzSätze Step 1
 P_Array(i, 1) = DBS!Name
 P_Array(i, 2) = DBS!Vorname
 Debug.Print P_Array(i, 1) & " " & P_Array(i, 2)
 DBS.MoveNext
Next i

DBS.Close
Set DBS = Nothing
Exit Sub
```

```
fehler:
MsgBox "Die Tabelle konnte nicht gefunden werden!"
End Sub
```

Definieren Sie zu Beginn des Makros ein Array. Da Sie zu diesem Zeitpunkt noch nicht wissen, wie viele Sätze in der Tabelle PERSONAL vorhanden sind, müssen Sie diese Angabe zunächst flexibel halten. Stellen Sie danach über das Objekt `CurrentProject` die Verbindung zu Ihrer momentan geöffneten Datenbank her. Legen Sie daraufhin ein neues ADO-Objekt an und öffnen über die Methode `Open` die Tabelle PERSONAL.

Mithilfe der Eigenschaft `RecordCount` können Sie die Anzahl der Datensätze in der Tabelle ermitteln. Sie haben somit die Obergrenze, die Sie zur Definition Ihres Arrays einsetzen können. Diese dynamische Zuweisung von Speicherplatz wird über die Anweisung `ReDim` realisiert.

Füllen Sie nun in einer Schleife das Datenfeld `P_Array`, indem Sie über die Methode `MoveNext` einen Datensatz nach dem anderen in der Tabelle abarbeiten und die Felder NAME und VORNAME direkt in das Datenfeld `P_Array` schieben. Geben Sie testweise die Inhalte des Datenfeldes im Direktbereich Ihrer Entwicklungsumgebung über die Anweisung `Debug.Print` aus.

Schließen Sie am Ende der Verarbeitung die Tabelle PERSONAL über die Methode `Close` und geben den Speicherplatz, der für die Objektvariable `DBS` reserviert wurde, über die Anweisung `Set DBS = Nothing` wieder frei.

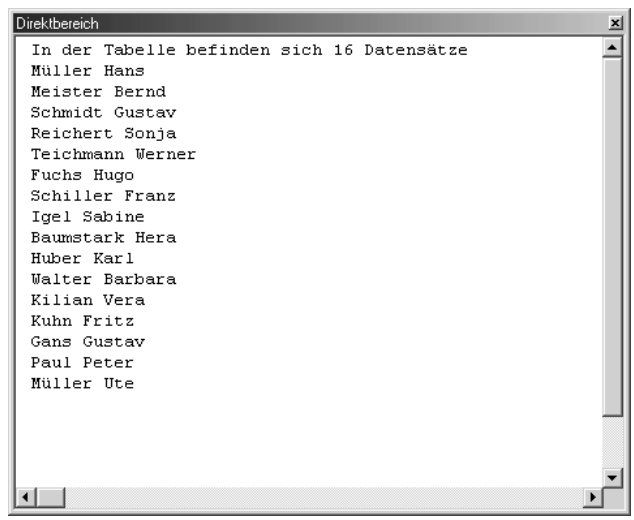


Abbildung 3.52: Die Daten aus dem Array in den Direktbereich ausgeben

Das Praxisbeispiel Straßentausch

Als abschließendes Beispiel zu den Arrays erweitern Sie die Tabelle PERSONAL aus der Datenbank SPRACHELEMENTE.MDB um eine Spalte mit dem Namen `Straße`. Füllen Sie diese Spalte zunächst mit unterschiedlich geschriebenen Straßenbezeichnungen wie Str., Straße oder Strasse. Diese unheitlichen Bezeichnungen sollen später einheitlich umgesetzt werden.

Abbildung 3.53: Eine Tabelle mit unterschiedlichen Straßenbezeichnungen

Personal : Tabelle

Name	Vorname	Straße	Standort	Umsatz
Müller	Hans	Burgstraße 10	München	23.456,00 €
Meister	Bernd	Hohezollerstraße 34	Stuttgart	45.000,00 €
Schmidt	Gustav	Teerweg 5	Hamburg	10.000,00 €
Reichert	Sonja	Amselweg 13	Köln	17.500,00 €
Teichmann	Werner	Kirchstr. 45	Saarbrücken	22.567,00 €
Fuchs	Hugo	Leopoldstr. 101	München	50.000,00 €
Schiller	Franz	Unterer Damm 1	Hamburg	34.689,00 €
Igel	Sabine	Bahnhofsstraße 4	Bonn	25.231,00 €
Baumstark	Hera	Württembergerstraße 90	Stuttgart	59.561,00 €
Huber	Karl	Ginsterstr. 5	Hannover	19.999,00 €
Walter	Barbara	Fuchs-Allee 12	Stuttgart	28.990,00 €
Kilian	Vera	Max-Blank-Strasse 5	München	51.983,00 €
Kuhn	Fritz	Kleinstr. 89	Bonn	18.750,00 €
Gans	Gustav	Burgweg 67	Trier	52.651,00 €
Paul	Peter	Thüringerstraße 21	Köln	51.411,00 €
Müller	Ute	Walstr. 99	Stuttgart	38.790,00 €
				0,00 €

Datensatz: 17 von 17

Um diese Straßenbezeichnungen zu vereinheitlichen, erfassen Sie folgenden Code.

Listing 3.70: Straßenbezeichnungen vereinheitlichen

```
Sub StraßenUmsetzen()
Dim P_Array() As String
Dim i As Integer
Dim AnzSätze As Integer
Dim DBS As New ADODB.Recordset

On Error GoTo fehler
  Set Conn = CurrentProject.Connection
  Set DBS = New ADODB.Recordset

  With DBS
  .Open "Personal", Conn, adOpenKeyset, _
   adLockOptimistic
   Debug.Print "In der Tabelle befinden sich " & _
        DBS.RecordCount & " Datensätze"
   AnzSätze = DBS.RecordCount
  End With


ReDim P_Array(1 To AnzSätze, 1 To 2)
```

```
i = 1
For i = 1 To AnzSätze Step 1
 P_Array(i, 1) = DBS!Straße
 DBS!Straße = Tausch(P_Array(i, 1))
 DBS.MoveNext
Next i
DBS.Close
Set DBS = Nothing
Exit Sub

fehler:
MsgBox "Die Tabelle konnte nicht gefunden werden!"
End Sub
```

Definieren Sie zu Beginn des Makros ein Array. Da Sie zu diesem Zeitpunkt noch nicht wissen, wie viele Sätze in der Tabelle PERSONAL vorhanden sind, müssen Sie diese Angabe zunächst flexibel halten. Stellen Sie danach über das Objekt `CurrentProject` die Verbindung zu Ihrer momentan geöffneten Datenbank her. Legen Sie daraufhin ein neues ADO-Objekt an und öffnen über die Methode `Open` die Tabelle PERSONAL.

Mithilfe der Eigenschaft `RecordCount` können Sie die Anzahl der Datensätze in der Tabelle ermitteln. Sie haben somit die Obergrenze, die Sie zur Definition Ihres Arrays einsetzen können. Diese dynamische Zuweisung von Speicherplatz wird über die Anweisung `ReDim` realisiert.

Füllen Sie nun in einer Schleife das Datenfeld `P_Array`, indem Sie über die Methode `MoveNext` einen Datensatz nach dem anderen in der Tabelle abarbeiten und die Felder NAME und VORNAME direkt in das Datenfeld `P_Array` schieben.

Rufen Sie die Funktion `Tausch` auf, die Ihnen bis jetzt noch fehlt. Die Funktion sieht wie folgt aus:

Listing 3.71: Funktion zum Tauschen von Zeichen

```
Function Tausch(s As String) As String
Dim i As Integer

i = InStr(s, "Straße")
If i > 0 Then
 Tausch = Left$(s, i - 1) & "str."
 If Len(s) > i + 6 Then
   Tausch = Tausch & Right$(s, Len(s) - (i + 5))
 End If
 Else
Tausch = s
End If
End Function
```

In der Funktion `Tausch` ermitteln Sie mit der Funktion `Instr` das Vorkommen der Zeichenfolge `Straße`. Wird im Datenbankfeld dieser String gefunden, dann wird ein Wert > 0 zurückgegeben. Wenn dem so ist, dann tauschen Sie die Bezeichnungen miteinander aus. Dabei kommen die Funktionen `Left`, `Right` und `Len` zum Einsatz.

Abbildung 3.54: Als Ergebnis liegt eine bereinigte Tabelle vor

Personal : Tabelle

Name	Vorname	Straße	Standort	Umsatz
Müller	Hans	Burgstr. 10	München	23.456,00 €
Meister	Bernd	Hohezollerstr. 34	Stuttgart	45.000,00 €
Schmidt	Gustav	Teerweg 5	Hamburg	10.000,00 €
Reichert	Sonja	Amselweg 13	Köln	17.500,00 €
Teichmann	Werner	Kirchstr. 45	Saarbrücken	22.567,00 €
Fuchs	Hugo	Leopoldstr. 101	München	50.000,00 €
Schiller	Franz	Unterer Damm 1	Hamburg	34.689,00 €
Igel	Sabine	Bahnhofsstr. 4	Bonn	25.231,00 €
Baumstark	Hera	Württembergerstr. 90	Stuttgart	59.561,00 €
Huber	Karl	Ginsterstr. 5	Hannover	19.999,00 €
Walter	Barbara	Fuchs-Allee 12	Stuttgart	28.990,00 €
Kilian	Vera	Max-Blank-str.e 5	München	51.983,00 €
Kuhn	Fritz	Kleinstr. 89	Bonn	18.750,00 €
Gans	Gustav	Burgweg 67	Trier	52.651,00 €
Paul	Peter	Thüringerstr. 21	Köln	51.411,00 €
Müller	Ute	Walstr. 99	Stuttgart	38.790,00 €
				0,00 €

Datensatz: 1 von 16

Das Praxisbeispiel Top3 Max und Min

Beim folgenden Beispiel sollen aus der Tabelle PERSONAL die Mitarbeiter extrahiert werden, die den größten bzw. die niedrigsten Umsätze erwirtschaftet haben. Dazu können Sie Arrays einsetzen, um die drei höchsten bzw. die drei niedrigsten Umsätze mit den dazugehörigen Mitarbeitern zu ermitteln.

Listing 3.72: Die drei erfolgreichsten Mitarbeiter ermitteln

```
Sub Top3WerteMax()
Dim Conn As New ADODB.Connection
Dim DBS As ADODB.Recordset
Dim wert(2) As Variant
Dim Mitarb(2) As Variant
Dim i As Integer

Set Conn = CurrentProject.Connection
Set DBS = New ADODB.Recordset
With DBS
 .CursorLocation = adUseClient
 .Open "Personal", Conn, adOpenKeyset, _
   adLockOptimistic
 .Sort = "Umsatz DESC"
  For i = 0 To 2
  wert(i) = DBS("Umsatz")
  Mitarb(i) = DBS("Name") & " " & DBS("Vorname")
```

```
  .MoveNext
  Next i
 DBS.Close
End With

Conn.Close
Set DBS = Nothing
Set Conn = Nothing

MsgBox "Die drei höchsten Umsätze lauten: " & vbLf & _
Format(wert(0), "0,000") & "--> " & Mitarb(0) & vbLf & _
Format(wert(1), "0,000") & "--> " & Mitarb(1) & vbLf & _
Format(wert(2), "0,000") & "--> " & Mitarb(2), _
vbInformation, "Die Top-3 Mitarbeiter"
End Sub
```

Bei diesem Beispiel ist die Datenbank NORDWIND.MDB bereits geöffnet. Daher können Sie sich den `Open`-Befehl sparen und stattdessen beim Öffnen der Tabelle PERSONAL auf die geöffnete Datenbank verweisen. Zu diesem Zweck haben Sie der Eigenschaft `Connection` die aktuelle Datenbank über das Objekt `CurrentProject` zugewiesen.

Über die Anweisung `Set` mit dem Zusatz `New` erstellen Sie ein neues `RecordSet`-Objekt. In dieses Objekt wird später der gefundene Satz übertragen, geändert und dann zurückgeschrieben.

Wenden Sie danach die Eigenschaft `Sort` an und geben Sie vor, nach welchen Kriterien sortiert werden soll. Haben Sie mehrere Sortierkriterien zur Auswahl, dann geben Sie diese entsprechend der Sortierreihenfolge getrennt durch Kommata ein. Bei der Sortierreihenfolge selbst können Sie entweder `ASC` für aufsteigende Sortierung oder `DESC` für absteigende Sortierung angeben. Dabei erfassen Sie nach dem Feldnamen ein Leerzeichen und hängen die gewünschte Sortierkonstante an.

In einer nachfolgenden Schleife, die genau fünfmal durchlaufen wird, werden die Umsätze sowie die dazugehörenden Mitarbeiternamen ermittelt und in die Variablen `Wert` und `Mitarb` geschrieben.

Haben Sie die Mitarbeiter mit den größten Umsätzen ermittelt, geben Sie diese am Bildschirm über die Funktion `Msgbox` aus. Bringen Sie die Umsätze dabei über die Funktion `Format` in das richtige Format.

Vergessen Sie nicht, die Tabelle sowie die Verbindung über die Methode `Close` zu schließen und die Objektverweise wieder aufzuheben.

Analog zum vorherigen Beispiel können Sie die drei schlechtesten Umsätze ausweisen, indem Sie das folgende Makro starten:

Abbildung 3.55: Die drei erfolgreichsten Mitarbeiter wurden ermittelt

Listing 3.73: Die drei schlechtesten Umsätze ermitteln

```
Sub Top3WerteMin()
Dim Conn As New ADODB.Connection
Dim DBS As ADODB.Recordset
Dim wert(2) As Variant
Dim Mitarb(2) As Variant
Dim i As Integer

Set Conn = CurrentProject.Connection
Set DBS = New ADODB.Recordset
With DBS
 .CursorLocation = adUseClient
 .Open "Personal", Conn, adOpenKeyset, _
   adLockOptimistic
 .Sort = "Umsatz ASC"
  For i = 0 To 2
  wert(i) = DBS("Umsatz")
  Mitarb(i) = DBS("Name") & " " & DBS("Vorname")
  .MoveNext
  Next i
 DBS.Close
End With

Conn.Close
Set DBS = Nothing
Set Conn = Nothing

MsgBox "Die drei niedrigsten Umsätze lauten: " & _
vbLf & Format(wert(0), "0,000") & "--> " & _
Mitarb(0) & vbLf & Format(wert(1), "0,000") _
& "--> " & Mitarb(1) & vbLf & _
Format(wert(2), "0,000") & "--> " & Mitarb(2), _
vbInformation, "Die schlechtesten Umsätze"
End Sub
```

Wenden Sie bei der Methode `Sort` den Zusatz `ASC` an, um zu erreichen, dass in der Tabelle PERSONAL alle Daten nach dem Umsatz in aufsteigender Art und Weise sortiert werden. Dadurch befinden sich die drei niedrigsten Umsätze ganz oben in der Tabelle und können direkt im Anschluss in die Arrays `Wert` und `Mitarb` eingelesen werden.

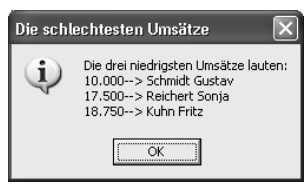


Abbildung 3.56: Die drei schlechtesten Umsätze wurden ermittelt

Eine Erweiterung dieser Aufgabenstellung stellt das folgende Makro dar. Dabei werden alle Umsätze und Mitarbeiternamen in den Direktbereich der Entwicklungsumgebung geschrieben. Da zu Beginn des Makros die genaue Anzahl der Mitarbeiter, die sich in der Tabelle befinden, noch nicht bekannt ist, müsen später während des Makros die Variablen `Wert` und `Mitarb` neu dimensioniert werden.

Listing 3.74: Variabler Einsatz von dynamischen Arrays

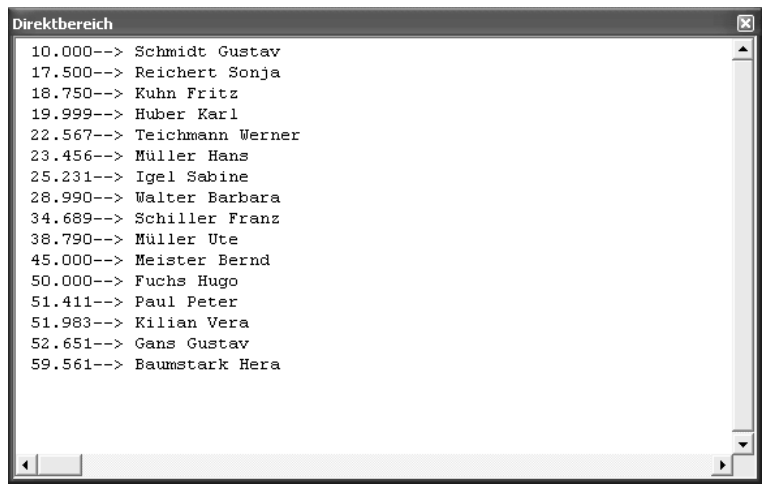
```
Sub AlleUmsätzeImDirektbereich()
Dim Conn As New ADODB.Connection
Dim DBS As ADODB.Recordset
Dim wert() As Variant
Dim Mitarb() As Variant
Dim i As Integer
Dim AnzSätze As Long

Set Conn = CurrentProject.Connection
Set DBS = New ADODB.Recordset
With DBS
 .CursorLocation = adUseClient
 .Open "Personal", Conn, adOpenKeyset, _
   adLockOptimistic
 .Sort = "Umsatz ASC"

AnzSätze = .RecordCount
ReDim wert(0 To AnzSätze)
ReDim Mitarb(0 To AnzSätze)

 i = 0
 Do Until .EOF
  wert(i) = DBS("Umsatz")
  Mitarb(i) = DBS("Name") & " " & DBS("Vorname")
  .MoveNext
  i = i + 1
 Loop
 .Close
End With

For i = 0 To UBound(wert) - 1
 Debug.Print Format(wert(i), "0,000") & _
"--> " & Mitarb(i)
Next i

Conn.Close
Set DBS = Nothing
Set Conn = Nothing
End Sub
```

Nachdem Sie über die Eigenschaft `RecordCount` ermittelt haben, wie viele Datensätze sich in der Tabelle PERSONAL befinden, können Sie über die Anweisung `Redim` die genaue Größe der beiden Arrays `Wert` und `Mitarb` festlegen. Danach füllen Sie diese beiden Arrays mithilfe einer Schleife. Über die Methode `MoveNext` wird dabei jeweils der nächste Datensatz in der Tabelle

angesprungen. Achten Sie darauf, dass die Zählvariable `i` vor der `Do Until`-Schleife auf den Wert `0` gesetzt und innerhalb der Schleife dann hochgezählt wird. Sind die Arrays gefüllt, können Sie die Tabelle mittels der Methode `Close` schließen.

In einer anschließenden Schleife durchlaufen Sie alle Einträge des Arrays `Wert`. Dabei können Sie mithilfe der Funktion `UBound` den letzten verfügbaren Wert des Arrays ermitteln, der letztendlich dann auch das Endekritrium der Schleife ist. Der erste Eintrag im Array ist immer der Eintrag mit dem Index `0`. Innerhalb der `For Next`-Schleife geben Sie die Inhalte der Arrays im Direktbereich der Entwicklungsumgebung aus, indem Sie die Anweisung `Debug.Print` einsetzen.

Abbildung 3.57: Die Umsätze und Mitarbeiternamen wurden in den Direktbereich von Access geschrieben



3.8 Operatoren

Ein wichtiges Mittel in der Programmierung sind Operatoren. Diese setzen Sie beispielsweise in Verzweigungen ein, um Werte zu prüfen oder zu vergleichen. Man unterscheidet in der Programmierung zwischen vier verschiedenen Arten von Operatoren:

- Arithmetische Operatoren
- Vergleichsoperatoren
- Verkettungsoperatoren
- Logische Operatoren

Lernen Sie dazu ein paar Beispiele kennen.

Arithmetische Operatoren

Beim Rechnen in Access-VBA verwenden Sie dieselben Operatoren, die Sie vom Taschenrechner her schon kennen. Auch bei den Rechenregeln rechnet Access nach der allgemein gültigen Punkt-vor-Strich-Regel.

Auf die vier Grundrechenarten werden wir nicht weiter eingehen und kein Beispiel dazu geben, wohl aber auf die restlichen Operatoren, die nicht so geläufig sein dürften.

Im folgenden Beispiel wird der Operator `Mod` eingesetzt, um den Rest einer ganzzahligen Division zweier Zahlen zurückzugeben.

Listing 3.75: Rest einer ganzzahligen Division ermitteln

```
Sub Rechnen()
Dim Zahl1 As Integer
Dim Zahl2 As Integer
Dim Erg As Integer

 Zahl1 = 10
 Zahl2 = 3.5

 Erg = Zahl1 Mod Zahl2
 MsgBox Erg
End Sub
```

Der Vollständigkeit halber können Sie die restlichen arithmetischen Operatoren in der Tabelle 3.9 einsehen.

Tabelle 3.9: Die arithmetischen Operatoren

Operator	Beschreibung
+	Addiert Zahlen miteinander
-	Subtrahiert Zahlen voneinander
*	Multipliziert Zahlenwerte
/	Dividiert Zahlenwerte
\	Dient zur Division zweier Zahlen und gibt ein ganzzahliges Ergebnis zurück
^	Potenziert eine Zahl mit einem Exponenten

Vergleichsoperatoren

Mithilfe der Vergleichsoperatoren können Sie Zahlenwerte oder auch Texte miteinander vergleichen.

Im folgenden Beispiel in Listing 3.76 werden zwei Zahlenwerte miteinander verglichen.

Listing 3.76: Zahlenwerte vergleichen

```
Sub Vergleich()
Dim Zahl1 As Integer
Dim Zahl2 As Integer

Zahl1 = 100
Zahl2 = 95

If Zahl1 >= Zahl1 Then MsgBox _
 "Zahl1 ist größer als Zahl2" _
Else MsgBox "Zahl1 ist kleiner als Zahl2"
End Sub
```

In der Verzweigung fragen Sie über das Zeichen `>=` ab, ob `Zahl1` größer oder gleich groß ist wie `Zahl2`. Entsprechend geben Sie dann eine Meldung auf dem Bildschirm aus.

In Tabelle 3.10 finden Sie eine Auflistung der möglichen Vergleichsoperatoren.

Tabelle 3.10: Die Vergleichsoperatoren

Operator	Beschreibung
<	kleiner als
<=	kleiner oder gleich
>	größer als
>=	größer oder gleich
=	gleich
<>	ungleich

Verkettungsoperatoren

Bei den Verkettungsoperatoren verketten Sie Zeichenfolgen miteinander. Der Verkettungsoperator lautet `&`.

Im nächsten Beispiel in Listing 3.77 werden mehrere Variablen miteinander verkettet und auf dem Bildschirm ausgegeben.

Listing 3.77: Zeichenfolgen miteinander verketten

```
Sub Verketten()
Dim Stadt As String
Dim PLZ As String
Dim s_erg As String

 Stadt = InputBox("Geben Sie Ihren Wohnort ein!")
 If Stadt = "" Then Exit Sub
 PLZ = InputBox("Geben Sie die PLZ Ihres Wohnorts ein!")
 If PLZ = "" Then Exit Sub
 s_erg = PLZ & " " & Stadt
 MsgBox s_erg
End Sub
```

Nützen Sie den Verkettungsoperator, um die Variablen `PLZ` und `Stadt` miteinander zu verketten.

Logische Operatoren

Mithilfe der logischen Operatoren können Sie beispielsweise Bedingungen für Schleifen bzw. Verzweigungen oder für deren Abbruch formulieren.

Im nächsten Makro in Listing 3.78 prüfen Sie, ob zwei Bedingungen für eine Abfrage zutreffen. Dabei lauten die Bedingungen wie folgt:

Listing 3.78: Logische Operatoren einsetzen

```
Sub LogischeOper()
Dim Zahl1 As Integer
Dim Zahl2 As Integer

Zahl1 = 100
Zahl2 = 95

If Zahl1 = 100 And Zahl2 = 95 Then _
MsgBox "Beide Zahlen korrekt!"
End Sub
```

Mithilfe des logischen Operators `And` können Sie überprüfen, ob die beiden Bedingungen zutreffen.

Tabelle 3.11 zeigt weitere mögliche und geläufige logische Operatoren.

Tabelle 3.11: Die logischen Operatoren

Operator	Beschreibung
`And`	Hier müssen beide Bedingungen zutreffen.
`Or`	Es muss eine der beiden Bedingungen zutreffen.
`Xor`	Dient zum Durchführen einer logischen Exklusion zwischen zwei Ausdrücken
`Eqv`	Dient zum Bestimmen einer logischen Äquivalenz zwischen zwei Ausdrücken. Hat einer der beiden Ausdrücke den Wert `0`, so ist Ergebnis ebenfalls Null.
`Not`	Führt eine logische Negation eines Ausdrucks durch

3.9 Eigene Funktionen schreiben

Programmieren Sie Funktionen, die Sie innerhalb der Entwicklungsumgebung im Zusammenspiel mit Makros einsetzen. Diese Funktionen sind dann ratsam, wenn sie in mehreren Makros gebraucht werden. Anstatt denselben Programm-Code mehrfach zu erfassen, schreiben Sie einmal eine Funktion dazu und rufen diese aus den Makros einfach auf. Diese Programmierweise ist übersichtlich, pflegeleicht und macht Spaß. Lernen Sie den Einsatz von Funktionen anhand einiger ausgesuchter Beispiele aus der Praxis kennen.

Dateien in einem Verzeichnis zählen

Stellen Sie sich vor, Sie müssen in einem Makro feststellen, wie viele Dateien sich in einem Verzeichnis befinden. Dazu erfassen Sie zunächst folgende Funktion:

Listing 3.79: Funktion zum Zählen von Dateien

```
Function DZ(str) As Long
Dim DatNam As String
Dim n As Long

  DatNam = Dir$(str & "\*.*")
  Do While Len(DatNam) > 0
      n = n + 1
      DatNam = Dir$()
  Loop
  DZ = n
End Function
```

Die Funktion `DZ` erwartet als Eingabe den Namen des Verzeichnisses, auf das Sie zugreifen möchten. Als Ergebnis liefert die Funktion Ihnen im Datentyp `Long` die Anzahl der ermittelten Dateien. Wenn Sie nur bestimmte Dateien gezählt haben möchten, können Sie die obige Funktion abändern, indem Sie die Zeichenfolge `DatNam = Dir$(str & "\*.*")` beispielsweise in `DatNam = Dir$(str & "\*.mdb")` ändern. Diese kleine Änderung bewirkt, dass nur Access-Datenbanken gezählt werden. Jetzt fehlt nur noch das Makro, welches der Funktion das Verzeichnis übergibt und die Rückmeldung der Funktion auswertet.

```
Sub ZählenDateien()
Const Verz = "C:\Eigene Dateien\"
Dim i As Long

 i = DZ(Verz)
 MsgBox "Das Verzeichnis " & Verz & _
    " enthält " & i & " Dateien!"
End Sub
```

Legen Sie am besten gleich zu Beginn fest, welches Verzeichnis Sie durchsuchen möchten. Übergeben Sie anschließend der Funktion `DZ` genau dieses Verzeichnis.

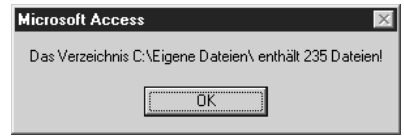


Abbildung 3.58: Datenbanken zählen und ausgeben

Prüfen, ob eine bestimmte Datei existiert

In diesem Beispiel möchten Sie über eine Funktion prüfen lassen, ob es eine bestimmte Datenbank überhaupt gibt. Insbesondere wenn Sie vorhaben, eine Datenbank mit VBA zu öffnen, sollten Sie vorher sicherstellen, dass es diese Datenbank auch gibt. Dazu erfassen Sie eine Funktion und übergeben dieser den Datenbanknamen mitsamt der Laufwerks- und Pfadangabe.

```
Function DBVorhanden(str As String) _
 As Boolean
  DBVorhanden = False
  If Len(str) > 0 Then DBVorhanden = _
  (Dir(str) <> "")
  Exit Function
End Function
```

Listing 3.80: Funktion zum Prüfen, ob eine Datenbank existiert

Wie schon gesagt, erwartet die Funktion den Namen der Datenbank, deren Vorhandensein Sie prüfen möchten. Die Prüfung, ob überhaupt eine Zeichenfolge an die Funktion übergeben wurde, erfolgt über die Funktion `Len`. Wird eine Länge von `0` gemeldet, wurde überhaupt keine Zeichenfolge an die Funktion übergeben. Wenn ja, entspricht diese in jedem Fall einer Größe `> 0`. Die Funktion `Dir` versucht nun auf die Datenbank zuzugreifen. Ist die Datenbank nicht vorhanden, meldet die Funktion eine Leerfolge zurück. Damit wird der Datentyp `Boolean` mit dem Wert `False` an das aufrufende Makro zurückgemeldet. Anderenfalls liefert die Funktion den Wert `True` zurück.

```
Sub DateiDa()
Dim bln As Boolean
Const DB = "DB1.mdb"

bln = DBVorhanden("C:\eigene Dateien\" & DB1)
If bln = True Then MsgBox "Die Datenbank " & _
 DB & " ist vorhanden!" Else _
 MsgBox "Die Datenbank existiert nicht!"
End Sub
```

Definieren Sie auch hier gleich zu Beginn des Makros die gewünschte Datenbank mit einer Konstanten. Änderungen können somit schneller ausgeführt werden.

Abbildung 3.59: Die Existenz von Datenbanken prüfen

Prüfen, ob eine Datei gerade bearbeitet wird

Wenn Sie in einem Netzwerk arbeiten und versuchen, eine Datenbank zu öffnen, die ein Kollege bereits geöffnet hat, dann sollten Sie vor dem Öffnen der Datenbank prüfen, ob Sie diese im Exklusivzugriff haben. Die Funktion für diesen Zweck lautet:

Listing 3.81: Funktion zum Prüfen, ob eine Datenbank bereits geöffnet ist

```
Function DBInBearbeitung(s As String)_
 As Boolean
On Error Resume Next
Open s For Binary Access Read Lock Read As #1
Close #1
If Err.Number <> 0 Then
  DBInBearbeitung = True
  Err.Clear
End If
End Function
```

Mit der Methode `Open` öffnen Sie die Datenbank mit Lesezugriffsrechten. Ist diese Datenbank bereits geöffnet, liefert Ihnen die Eigenschaft `Number` des `Err`-Objekts einen Laufzeitfehler `> 0`. In diesem Fall wird die Datenbank zurzeit von einem anderen Anwender bearbeitet. Das aufrufende Makro für diese Aufgabe lautet:

```
Sub DateiFrei()
Const DB = "DB1.mdb"

If DBInBearbeitung("C:\Eigene Dateien\" & DB) = False _
Then MsgBox "Die Datenbank " & DB & _
 " ist für Bearbeitung frei!" Else _
 MsgBox "Die Datenbank " & DB & " ist in Bearbeitung!"
End Sub
```

Übergeben Sie der Funktion `DBInBearbeitung` den Namen der Datenbank und werten Sie die Rückgabe aus.

Abbildung 3.60: Ist eine Datenbank doppelt geöffnet?

Dokumenteigenschaften einer Arbeitsmappe ermitteln

Anhand einer Funktion können Sie diverse Dokumenteigenschaften einer Datenbank ermitteln. Dabei wird der Funktion der Datenbankname sowie eine Eigenschaftsnummer übergeben, durch die die Funktion dann die entsprechenden Informationen zur Verfügung stellt.

Die einzelnen Informationen und die dazugehörigen Eigenschaftsnummern entnehmen Sie Tabelle 3.12.

Tabelle 3.12: Die Aufschlüsselung der Eigenschaftsnummer

Eigenschaftsnummer	Beschreibung
0	Dateiname mit Pfad
1	Nur Pfad
2	Nur Dateiname
3	Dateityp
4	Dateigröße in Byte
5	Erstellt am
6	Letzte Änderung am
7	Letzter Zugriff am

Wenden Sie diese Eigenschaftsnummern nun in einer eigenen Funktion an.

Listing 3.82: Dokumenteigenschaften einer Datenbank ermitteln

```
Function ZeigeDBEigenschaften _
(Dateiname, EigenschaftsNr As Byte)
Dim fso As Object
Dim tmp As String

On Error Resume Next
Set fso = CreateObject("Scripting.FileSystemObject")
With fso.GetFile(Dateiname)
Select Case EigenschaftsNr
 Case Is = 0: tmp = .Path
 Case Is = 1: tmp = Mid(.Path, 1, _
    Len(.Path) - Len(.Name))
 Case Is = 2: tmp = .Name
 Case Is = 3: tmp = .Type
 Case Is = 4: tmp = .Size
```

```
 Case Is = 5: tmp = CDate(.DateCreated)
 Case Is = 6: tmp = CDate(.DateLastModified)
 Case Is = 7: tmp = CDate(.DateLastAccessed)
 Case Else
  tmp = "Ungültige Eigenschaftsnummer!"
End Select
End With
ZeigeDBEigenschaften = tmp
End Function
```

Übergeben Sie der Funktion jetzt die Eigenschaftsnummer `5`, um das Erstellungsdatum einer Datenbank zu ermitteln.

```
Sub DokumentEigenschaften()
Const verz = "C:\Eigene Dateien\"
Const DB = "DB1.mdb"

 MsgBox "Das Erstellungsdatum der Datenbank: " & DB & _
 " ist der " & ZeigeDBEigenschaften(verz & DB, 5)
End Sub
```

In der Funktion erstellen Sie im ersten Schritt einen Verweis auf das `FileSystemObject`, um damit die Informationen bezüglich der Arbeitsmappe zu erlangen. Danach werten Sie die übergebene Eigenschaftsnummer in einer `Select Case`-Anweisung aus.

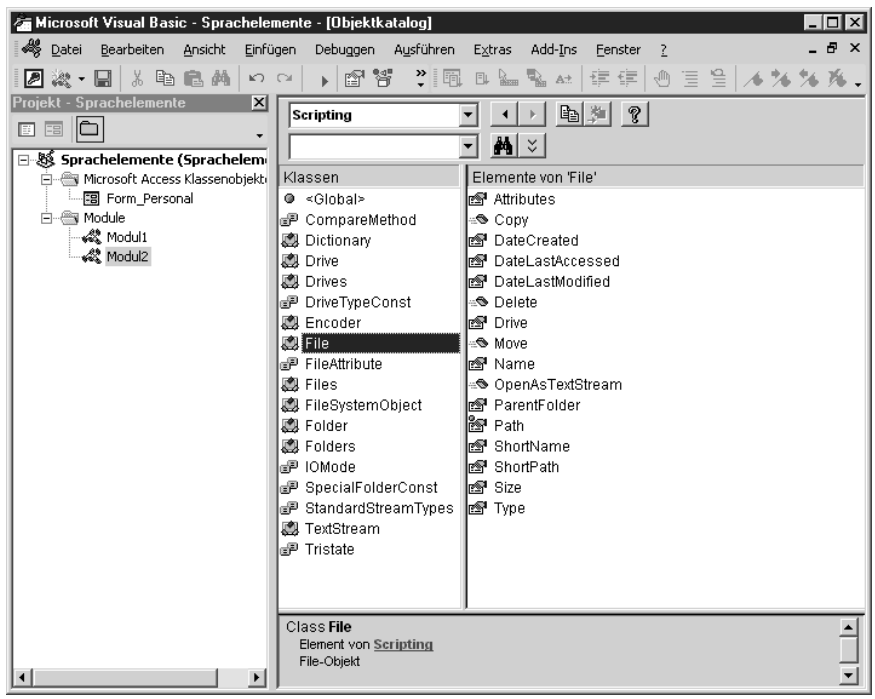


Abbildung 3.61: Die Bibliothek MICROSOFT SCRIPTING RUNTIME

Die verwendeten Eigenschaften des `FileSystemObjekts` *können Sie im Objektkatalog nachsehen. Dazu müssen Sie vorher die Bibliothek* MICROSOFT SCRIPTING RUNTIME *in der Entwicklungsumgebung über den Menübefehl* EXTRAS/VERWEISE *aktivieren.*

Letzten Tag im Monat ermitteln

Vielleicht haben Sie manchmal auch Probleme, den letzten Tag eines Monats schnell zu erkennen. Hat der Monat jetzt 30 oder 31 Tage? Es gibt hierfür zwar recht einfache Bauernregeln wie etwa das Zählen der Mulden zwischen Fingerknochen. Knöchel bedeutet 31 Tage, Mulde 30 (außer Februar). Ob man mit dem linken oder rechten Knöchel anfängt, ist allerdings ziemlich egal. Eine VBA-Lösung bietet die Funktion aus Listing 3.83.

```
Function LTImMo(inputdate As Date) As Date
 LTImMo = DateSerial _
 (Year(inputdate), Month(inputdate) + 1, 0)
End Function
```

Listing 3.83: Den letzten Tag im Monat ermitteln

Das aufrufende Makro könnte wie folgt aussehen:

```
Sub LetzterTagImMonatErmitteln()
Dim s As String
s = LTImMo("12.02.02")
MsgBox s
End Sub
```

Mithilfe der Funktion `DateSerial` wird ein Datum in seine Bestandteile zerlegt. Über die Funktionen `Year` und `Month` extrahieren Sie dann das jeweilige Jahr sowie den Monat.

Möchten Sie nicht das komplette Datum wissen, sondern nur den Tag, dann schreiben Sie die Anweisung `MsgBox Day(s)`.

Sonderzeichen aus Strings entfernen

Müssen Sie Daten weiterverarbeiten, in denen Sonderzeichen wie Punkte und Kommata vorkommen, die Sie nicht weiterverarbeiten möchten, dann schreiben Sie eine Funktion, die diese Zeichen aus einem String entfernt.

Sehen Sie sich dazu die folgende Funktion an.

```
Function SonderzRaus(s As String) As Double
Dim Puffer As String
Dim i As Integer
```

Listing 3.84: Kommata aus String entfernen

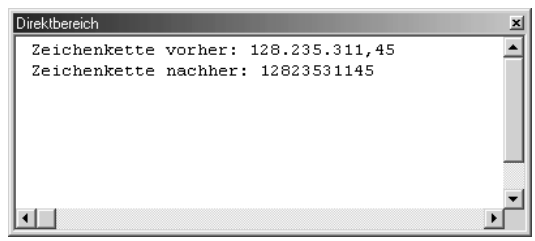
```
Puffer = ""
i = 1
While InStr(i, s, ",") > 0
  Puffer = Puffer & Mid(s, i, InStr(i, s, ",") - i)
  i = InStr(i, s, ",") + 1
Wend
Puffer = Puffer & Mid(s, i)
SonderzRaus = CDbl(Puffer)
End Function
```

In der Funktion durchlaufen Sie eine Schleife, in der die Zeichen jeweils bis zum nächsten Komma in den String `Puffer` übertragen werden. Dabei wird das Komma aber nicht übertragen, da Sie es jeweils wieder über die Variable `i` subtrahieren. Ermitteln Sie dananch die Position des nächsten Kommas über die Funktion `Instr`.

Erfassen Sie nun die aufrufende Prozedur und übergeben der Funktion einen String, der Sonderzeichen wie Punkte und Kommata enthält.

```
Sub PunkteUndKommataRaus()
Dim zk As String
Dim s As String

zk = "128.235.311,45"
Debug.Print "Zeichenkette vorher: " & zk
s = SonderzRaus(zk)
Debug.Print "Zeichenkette nachher: " & s
End Sub
```

Abbildung 3.62: Zeichenketten bereinigen



Eine eigene Runden-Funktion erstellen

Selbstverständlich gibt es eine Standardfunktion für das Runden in Access mit dem Namen ROUND. Bei dieser Funktion kommt es aber zu einem Problem:

Der Wert 0,5 wird abgerundet auf den Wert 0.

Der Wert 1,5 wird aufgerundet auf den Wert 2.

Zu erwarten wäre aber, dass der Wert 0,5 auf den Wert 1 aufgerundet würde. Wenn Sie das genaue Ergebnis erhalten möchten, dann schreiben Sie eine Funktion, die Sie in der folgenden Funktion `RundenW` sehen können.

Listing 3.85: Richtiges Runden mit einer eigenen Funktion

```
Function RundenW(Wert, Stellen)
Dim Div As Integer

  If IsNumeric(Wert) Then
    Div = 10 ^ Stellen
    RundenW = Int(Wert * Div + 0.5) / Div
  Else
    Exit Function
  End If
End Function
```

Rufen Sie die Funktion `RundenW` auf, indem Sie Ihr den Wert sowie die Information übergeben, nach der wievielten Stelle nach dem Komma gerundet werden soll.

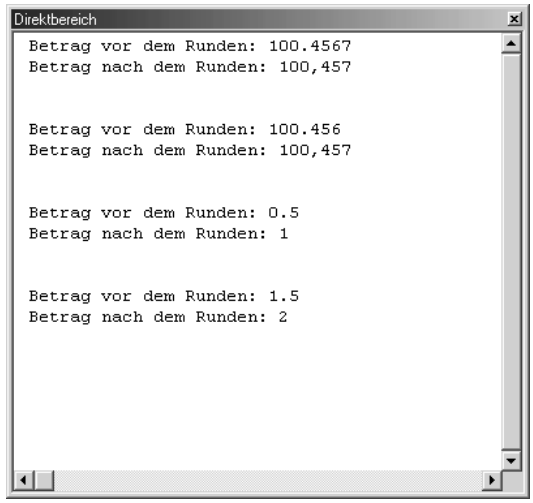
Listing 3.86: Testwerte für das Runden bereitstellen

```
Sub RundenWerte()
Dim Betrag As Currency

Debug.Print "Betrag vor dem Runden: " & "100.4567"
Betrag = RundenW(100.4567, 3)
Debug.Print "Betrag nach dem Runden: " & Betrag
Debug.Print Chr(13)

Debug.Print "Betrag vor dem Runden: " & "100.456"
Betrag = Round(100.4567, 3)
Debug.Print "Betrag nach dem Runden: " & Betrag
Debug.Print Chr(13)

Debug.Print "Betrag vor dem Runden: " & "0.5"
Betrag = RundenW(0.5, 0)
Debug.Print "Betrag nach dem Runden: " & Betrag
Debug.Print Chr(13)

Debug.Print "Betrag vor dem Runden: " & "1.5"
Betrag = RundenW(1.5, 0)
Debug.Print "Betrag nach dem Runden: " & Betrag
End Sub
```

Die Position der ersten Zahl eines Strings ermitteln

Haben Sie einen String, der sowohl numerische als auch alphanumerische Wert enthält, und möchten Sie nun die Position der ersten Zahl des Strings ermitteln, dann wird Sie die folgende Funktion interessieren. Sie meldet Ihnen die Position des ersten numerischen Zeichens eines Strings.

Abbildung 3.63: Die Werte werden richtig gerundet



Listing 3.87: Die Position der ersten Zahl im String ermitteln

```
Function PosErsteZahl(Text) As Integer
Dim i As Integer

 For i = 1 To Len(Text)
  Select Case Asc(Mid(Text, i, 1))
    Case 0 To 64, 123 To 197
      PosErsteZahl = i
      Exit Function
    End Select
 Next i
 PosErsteZahl = 0
End Function

Sub ErsteZahlErmitteln()
MsgBox PosErsteZahl("KT678GB")
End Sub
```

Ermitteln Sie im ersten Schritt die Länge des übergebenen Strings und setzen dafür die Funktion `Len` ein. Danach prüfen Sie mithilfe der Funktion `Asc` das jeweils aktuelle Zeichen des Strings, indem Sie dieses in einen `Integer`-Wert umwandeln. Mit der Funktion `Mid` extrahieren Sie jeweils das nächste Zeichen aus dem String. Dabei entsprechen die Werte 65-90 Kleinbuchstaben und die Werte 97-122 den Großbuchstaben. Diese Wertbereiche grenzen Sie innerhalb der `Select Case`-Anweisung aus. Wird das erste numerische Zeichen im String gefunden, dann springen Sie mit der Anweisung `Exit Function` aus der Funktion. In der Variablen steht dann automatisch die richtige Position des Zeichens. Wurde kein numerisches Zeichen gefunden, dann meldet die Funktion den Wert 0 zurück.

Buchstaben eliminieren

Bei der nächsten Aufgabe sollen aus einem übergebenen String, der sowohl Buchstaben als auch Zahlen enthält, alle Buchstaben rausgeworfen werden. Diese Aufgabe lösen Sie mit der folgenden Funktion:

Listing 3.88: Buchstaben aus Strings entfernen

```
Function BuchstRaus(Text) As Integer
Dim i As Integer

 For i = 1 To Len(Text)
  Select Case Asc(Mid(Text, i, 1))
    Case 0 To 64, 123 To 197
      BuchstRaus = BuchstRaus & Mid(Text, i, 1)
    End Select
 Next i
End Function

Sub TextAusZahlenEntfernen()
MsgBox BuchstRaus("Access 2003")
End Sub
```

Ermitteln Sie im ersten Schritt die Länge des Strings und setzen dafür die Funktion `Len` ein. Danach prüfen Sie mithilfe der Funktion `Asc` das jeweils aktuelle Zeichen des Strings, indem Sie dieses in einen `Integer`-Wert umwandeln. Mit der Funktion `Mid` extrahieren Sie jeweils das nächste Zeichen aus dem String. Dabei entsprechen die Werte 65-90 Kleinbuchstaben und die Werte 97-122 den Großbuchstaben und die restlichen Werte den Sonderzeichen. Diese Wertbereiche grenzen Sie innerhalb der `Select Case`-Anweisung aus. Wird ein Zeichen im String gefunden, welches numerisch ist bzw. einem Sonderzeichen entspricht, dann wird dieses gesammelt und bereits ermittelten Zahlen angehängt.

Römische Ziffern in arabische wandeln

Beim nächsten Beispiel soll eine römische Zahl in eine arabische Zahl umgewandelt werden. Die Funktion für diese Aufgabe können Sie im nächsten Listing sehen.

Listing 3.89: Römische Zahlen in unser Zahlenformat umwandeln

```
Function Arabisch(s As String) As Integer
Dim i As Integer
Dim TeilW  As Integer
Dim TeilW2 As Integer
Dim GesamtW As Integer

GesamtW = 0
TeilW = 0
TeilW2 = 0
```

```
 For i = 1 To Len(s)
  Select Case Mid(s, i, 1)
   Case Is = "M"
    TeilW = 1000
   Case Is = "D"
    TeilW = 500
   Case Is = "C"
    TeilW = 100
   Case Is = "L"
    TeilW = 50
   Case Is = "X"
    TeilW = 10
   Case Is = "V"
    TeilW = 5
   Case Is = "I"
    TeilW = 1
   Case Else
    TeilW = 0
  End Select
  If TeilW2 < TeilW Then
   GesamtW = GesamtW - TeilW2 * 2 + TeilW
  Else
   GesamtW = GesamtW + TeilW
  End If
  TeilW2 = TeilW
 Next i

 Arabisch = GesamtW
End Function

Sub Konvertieren()
MsgBox "Aus DXII wird " & Arabisch("DXII"), _
 vbInformation
End Sub
```

Das römische Zahlensystem verwendete Buchstaben als Zahlen. So war der Buchstabe M gleichbedeutend mit dem Wert 1000. Deshalb müssen Sie Zeichen für Zeichen im String abarbeiten und die entsprechenden Werte sammeln und am Ende als Rückgabewert für die Funktion `Msgbox` bereitstellen.

Abbildung 3.64: Wandeln römischer Zeichen in arabische Zahlen

Arabische Zahlen in römische Syntax wandeln

Beim gerade umgekehrten Vorgang werden Sie an dieser Stelle eine etwas andere Vorgehensweise kennen lernen. Möchten Sie eine arabische Zahl in eine römische Zahl wandeln, dann können Sie die Excel-Tabellenfunktion RÖMISCH einsetzen. Diese Funktion können Sie direkt aus Access aufrufen, indem Sie in der Entwicklungsumgebung die Bibliothek MICROSOFT EXCEL mithilfe des Menübefehls EXTRAS/VERWEISE aktivieren. Danach erfassen Sie folgenden Quellcode:

```
Function Römisch(s As String) As String
Dim xlApp As Excel.Application

  Set xlApp = New Excel.Application
  Römisch = xlApp.worksheetfunction.roman(s)
  xlApp.Quit
  Set xlApp = Nothing
End Function

Sub Konvertieren2()
MsgBox Römisch(123)
End Sub
```

Listing 3.90: Arabische Zahlen in römische Ziffern umwandeln

Erstellen Sie zunächst ein neues Excel-Objekt. Danach greifen Sie über das Objekt `WorksheetFunction` auf die Tabellenfunktion ROMAN zurück und übergeben dieser Funktion die arabische Zahl. Als Rückgabewert liefert die Funktion die dazugehörende römische Ziffer.

Abbildung 3.65: Wandeln arabischer Zahlen in römische Zeichen