

# Standardfunktionen nutzen, eigene Funktionen schreiben

In Excel gibt es eine ganze Reihe von Tabellenfunktionen, die Sie gewinnbringend in Ihren Makros einsetzen können. Warum beispielsweise die Tabellenfunktion *SUMME* in Form eines Makros nachbauen, wenn es diese bereits in Excel gibt. Es ist kein Problem, diese bereits vorgefertigten Funktionen in Excel auch für Makros zu nutzen. Dieses Kapitel zeigt Ihnen aber auch, wie Sie eigene Funktionen erstellen, verwalten und aufrufen können.

Die folgenden Beispiele sind in der Arbeitsmappe *Start.xlsm* im Modul *mdl\_Funktionen* zu finden.



## Die integrierten Tabellenfunktionen von Excel anzapfen

6.1

Alle Tabellenfunktionen liegen im Objekt *WorksheetFunction* in englischer Sprache vor. Einen ersten Überblick über die Funktionsvielfalt können Sie sich verschaffen, indem Sie über die Taste **F2** den Objektkatalog von Excel aufrufen und im Listenfeld *Klassen* das Objekt *WorksheetFunction* anklicken.

Da die Programmiersprache VBA in englischer Sprache vorliegt, ist es mitunter schwer, mit Kenntnis des deutschen Tabellenfunktionsnamens das englische Gegenstück zu finden. Dazu finden Sie bei den Beispieldaten zu diesem Buch eine Arbeitsmappe mit dem Namen **VbaListe.xls**, in der Sie eine Gegenüberstellung in der jeweiligen Landessprache und der dazugehörigen englischen Tabellenfunktion finden. Die Syntax ist bei den Tabellenfunktionen unabhängig von der Sprache natürlich immer dieselbe. Für die Programmierung müssen Sie die Muttersprache von VBA, also Englisch verwenden.



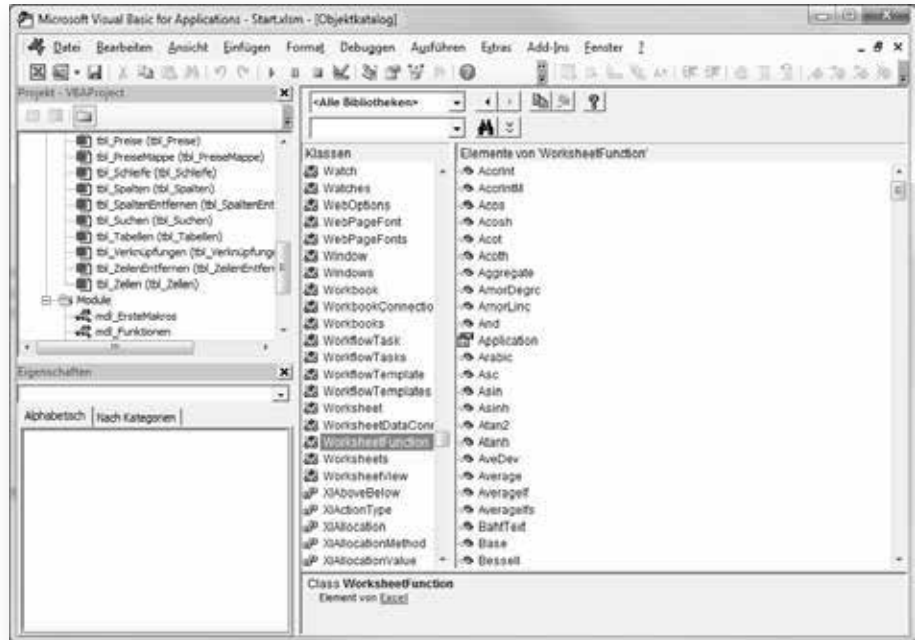




Abbildung 6.1: Alle zur Verfügung stehenden Tabellenfunktionen von Excel



Wenn Sie wissen, wie Sie eine deutsche Tabellenfunktion einsetzen, dann können Sie diese zuerst einmal in eine Zelle schreiben und mit  bestätigen. Danach setzen Sie den Mauszeiger in genau diese Zelle und wechseln in die Entwicklungsumgebung von Excel. Geben Sie dann folgende Zeile in das Direktfenster ein:

```
?ActiveCell.Formula
```

Bestätigen Sie danach mit . Die deutsche Formel wird dann automatisch in die englische Sprache übersetzt und Sie können sie in dieser Form in Ihren Makros verwenden.

## Einen Bereich summieren

In der folgenden Aufgabenstellung soll in der Tabelle *tbl\_Summe* der Bereich A1:A10 summiert und das Ergebnis der Rechenaufgabe in Zelle C1 ausgegeben werden.

```
Sub BereichSummieren()
```

```
tbl_Summe.Range("C1").Value = WorksheetFunction.Sum(tbl_Summe.Range("A1:A10"))
```

```
End Sub
```

Listing 6.1: Einen Bereich blitzschnell summieren

Mit dem schmalen Einzeiler aus Listing 6.1 können Sie die gestellte Aufgabe mit nur einer Codezeile lösen. Dabei übergeben Sie der Funktion `Sum` des Objekts `WorksheetFunction` den Bereich, den Sie summieren möchten. Vergessen Sie dabei nicht, auch die Referenz zur Tabelle mit anzugeben.

Was denken Sie, was am Makro aus Listing 6.1 noch zu verbessern wäre? Nun, der Bereich ist hier relativ statisch, d.h., wenn weitere Daten in Spalte A erfasst werden, dann müsste das Makro jedes Mal angepasst werden. Hier könnte man die Bereichsangabe etwas variabler gestalten.

Im Makro aus Listing 6.2 wird zunächst die letzte verwendete Zelle in Spalte A ermittelt und in der Variablen `ZeileMax` zwischen gespeichert. Danach wird die Variable `ZeileMax` in die Formel integriert.

```
Sub BereichSummierenDynamisch()
    Dim ZeileMax As Long

    With tbl_Summe

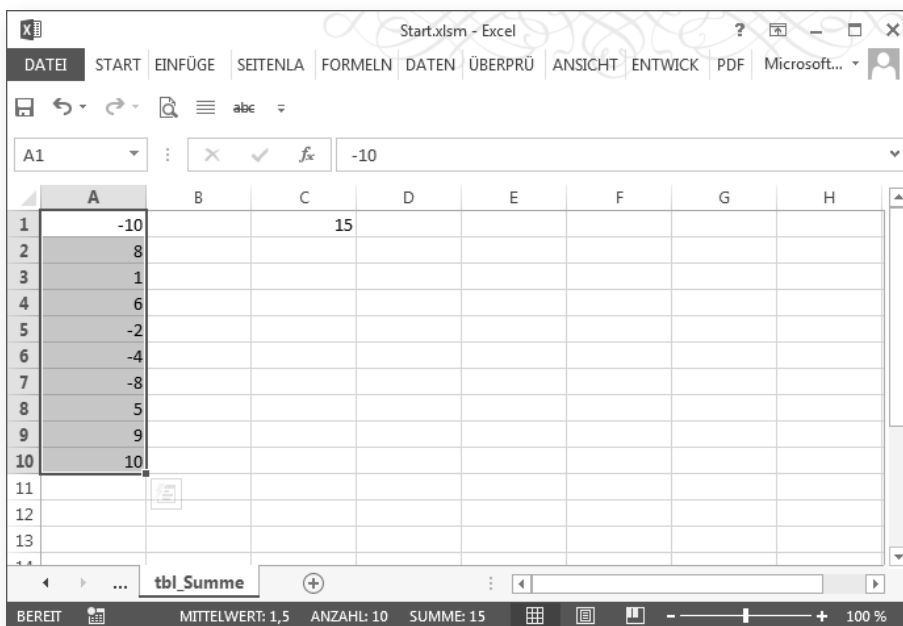
        ZeileMax = .Cells(.Rows.Count, 1).End(xlUp).Row
        .Range("C1").Value = WorksheetFunction.Sum(.Range("A1:A" & ZeileMax))

    End With

End Sub
```

*Listing 6.2: Eine dynamische Formel stricken*

Um die letzte gefüllte Zelle einer Spalte zu ermitteln, arbeiten wir mit der Eigenschaft `End`. Als Ausgangspunkt wählen wir dafür die letzte verfügbare Zelle der Spalte A (`.Cells(.Rows.Count, 1)`). Von dieser Zelle aus bewegen wir uns nach oben (`End(xlUp)`). Damit haben wir erst einmal die Adresse dieser Zelle. Um die Nummer dieser Zelle anzufragen, verwenden wir die Eigenschaft (`Row`).



*Abbildung 6.2: Das Ergebnis der Rechenoperation kann leicht über die Statusleiste kontrolliert werden*



Diesen auf den ersten Blick etwas sonderbaren Befehl können Sie übrigens auch über die Tastenkombination `Strg` + `Pfeil ↑` absetzen, wenn Sie sich in der letzten verfügbaren Zelle der Tabelle befinden. Excel springt dann ausgehend von unten zur letzten belegten Zelle der Spalte. Der Inhalt der Variablen wird mittels des Verkettungsoperators & in die Formel eingebaut.

## Eine bedingte Summierung durchführen

Bei der folgenden Aufgabe wird auf die Tabelle `tbl_Summe` zugegriffen, und es werden nur die Werte summiert, die größer oder gleich Null sind. Für diese Aufgabe wird die Worksheet-Funktion `SumIf` verwendet.

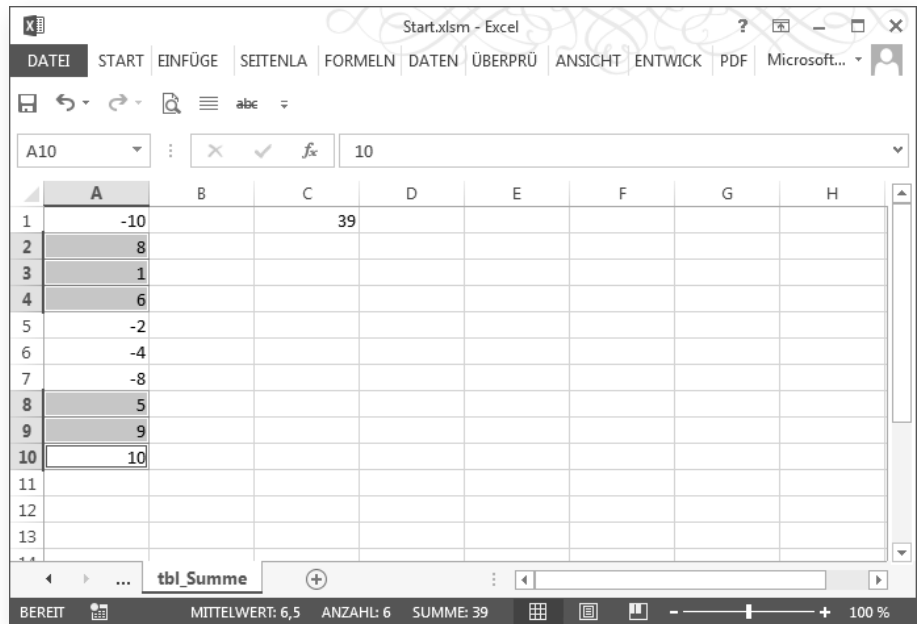
```
Sub BereichBedingtSummierenDynamisch()
    Dim ZeileMax As Long

    With tbl_Summe

        ZeileMax = .Cells(.Rows.Count, 1).End(xlUp).Row
        .Range("C1").Value = WorksheetFunction.SumIf(.Range("A1:A" & ZeileMax), ">=0")

    End With
End Sub
```

*Listing 6.3: Einen variablen Bereich bedingt summieren*



*Abbildung 6.3: Der Bereich wurde bedingt summiert*

Der verwendete Bereich in Spalte A wird mittels der Funktion `SumIf` summiert. Dabei wird im zweiten Argument das Kriterium für die Summierung in doppelten Anführungszeichen angegeben.

Es empfiehlt sich, wirklich jedes Makro zu plausibilisieren. Dazu können Sie alle positiven Werte nacheinander markieren und dabei die Taste `[Strg]` gedrückt halten. In der Statusleiste können Sie dann bequem das Ergebnis mit Zelle C1 vergleichen.

## Extremwerte ermitteln

Beim nächsten Beispiel aus Listing 6.4 werden die drei größten Werte aus einem Bereich abgefragt. Dazu kommt die Tabellenfunktion `Large` zum Einsatz.

```
Sub ExtremwerteInBereichErmittleIn()
    Dim Wert1 As Long
    Dim Wert2 As Long
    Dim Wert3 As Long
    Dim Bereich As Range

    Set Bereich = tbl_Bereich.Range("A1:D10")

    Wert1 = Application.WorksheetFunction.Large(Bereich, 1)
    Wert2 = Application.WorksheetFunction.Large(Bereich, 2)
    Wert3 = Application.WorksheetFunction.Large(Bereich, 3)

    MsgBox "Wert 1: " & Wert1 & vbCrLf & _
        "Wert 2: " & Wert2 & vbCrLf & _
        "Wert 3: " & Wert3, vbInformation, "Die größten Zahlen"

End Sub
```

Wir deklarieren im ersten Schritt drei Variablen vom Typ `Long`. Dort sollen später die drei Höchstwerte gesammelt werden. Dann deklarieren wir die Objektvariable `Bereich`, bei der wir mithilfe der Anweisung `Set` den zugrundeliegenden Bereich und die Tabelle bekanntgeben. Über den Einsatz der Funktion `Large`, bei der Sie im ersten Argument den Bereich und im zweiten Argument die Position übergeben müssen, erhalten Sie je nach Position den größten Wert (1), den zweitgrößten Wert (2) sowie den drittgrößten Wert (3). Zuletzt geben wir alle drei Werte auf dem Bildschirm mithilfe der Funktion `MsgBox` aus.

Listing 6.4: Die Top-3-Werte aus einem Bereich abfragen



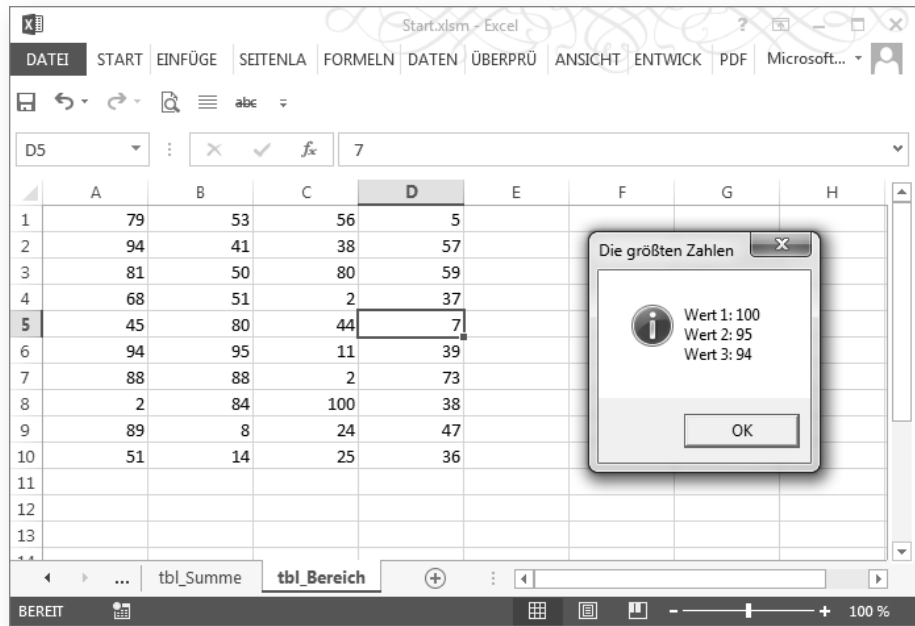


Abbildung 6.4: Die Top-Werte aus einem Bereich abfragen

## Leere Tabellen aus einer Arbeitsmappe entfernen

Bei der folgenden Aufgabe aus Listing 6.5 werden alle leeren Tabellen aus einer Arbeitsmappe entfernt.

```

Sub LeereTabellenAusMappeEntfernen()
    Dim Blatt As Worksheet

    Application.DisplayAlerts = False

    For Each Blatt In ThisWorkbook.Worksheets

        If Application.WorksheetFunction.CountA(Blatt.Cells) = 0 Then

            Blatt.Delete

        End If

    Next Blatt

    Application.DisplayAlerts = True

End Sub

```

Listing 6.5: Alle nicht gefüllten Tabellen einer Arbeitsmappe entfernen

Zu Beginn des Makros deklarieren wir eine Objektvariable vom Typ `Worksheet`. Um die Löschrückfrage von Excel auszuschalten, weisen wir der Eigenschaft `DisplayAlerts` den Wert `False` zu. Jetzt reagiert Excel beim Löschen einer Tabelle nicht mehr mit einer Rückfrage. Stattdessen werden die leeren Tabellen ohne weitere Interaktion aus der Arbeitsmappe entfernt. Dabei dürfen wir nicht vergessen, dass wir am Ende des Makros die Rückfragen wieder einschalten müssen.

Anschließend durchlaufen wir alle Tabellen der Arbeitsmappe, indem wir Blatt für Blatt über eine `For Each...Next`-Schleife abarbeiten. Innerhalb der Schleife wenden wir die Funktion `CountA` an, die den Wert `Null` zurückliefert, wenn wirklich keine einzige Zelle der jeweiligen Tabelle gefüllt ist. In der `Cells`-Auflistung sind alle Zellen einer Tabelle gleich einem Inhaltsverzeichnis verzeichnet. Die als leer identifizierten Tabellen werden danach mittels der Methode `Delete` aus der Arbeitsmappe gelöscht.

## Min- und Max-Wert in Bereich finden und einfärben

Bei der folgenden Aufgabe liegt in der Tabelle `tbl_MaxMin` ein Datenbereich vor, indem der größte sowie der kleinste Wert ermittelt und entsprechend gekennzeichnet werden sollen. Tritt der jeweilige Extremwert mehrfach auf, sollen alle entsprechende Zellen eingefärbt werden.

	A	B	C	D	E	F	G	H
1	59	71	88	44	29	95	40	
2	76	89	2	11	41	2	20	
3	72	8	54	97	16	34	56	
4	100	72	19	15	90	28	39	
5	76	42	53	3	92	41	43	
6	29	24	27	10	80	49	14	
7	78	78	61	64	38	52	2	
8	18	90	60	38	80	17	78	
9	81	24	85	57	30	100	83	
10	14	96	37	83	97	93	14	
11								
12								
13								
14								

Abbildung 6.5: Die Ausgangssituation – ein Bereich mit Zahlenwerten zwischen 1 und 100

Färben wir nun den größten Wert mit der Farbe Grün und den kleinsten Wert im Bereich mit der Farbe Rot. Dazu setzen wir das Makro aus Listing 6.6 ein.



```

Sub ExtremWerteFindenUndKennzeichnen()
    Dim Zelle As Range
    Dim Bereich As Range
    Dim WertMax As Long
    Dim WertMin As Long

    Set Bereich = tbl_MaxMin.Range("A1:G10")
    Bereich.Interior.ColorIndex = xlColorIndexNone
    WertMax = Application.WorksheetFunction.Max(Bereich)
    WertMin = Application.WorksheetFunction.Min(Bereich)

    For Each Zelle In Bereich

        Select Case Zelle.Value

            Case WertMax
                Zelle.Interior.ColorIndex = 4 'grün

            Case WertMin
                Zelle.Interior.ColorIndex = 3 'rot

        End Select

    Next Zelle

End Sub

```

*Listing 6.6: Die Extremwerte in einem Makro ermitteln und lokalisieren*

Wir deklarieren im ersten Schritt zwei Objektvariablen vom Typ `Range`. Die Variable `Zelle` benötigen wir zur Steuerung der späteren `For Each...Next`-Schleife. Mithilfe der Objektvariablen `Bereich` und der Anweisung `Set` geben wir den Bereich der Tabelle an, den wir verarbeiten möchten. In den beiden Variablen `WertMax` und `WertMin` ermitteln wir gleich im Anschluss mithilfe der Funktionen `Max` und `Min` den größten bzw. den kleinsten Wert im angegebenen Bereich.

Wir kennen jetzt zwar den größten und kleinsten Wert im Bereich, wissen aber nicht, wie oft diese Werte vorkommen und wo diese im Bereich genau liegen. Zu diesem Zweck setzen wir eine `For Each...Next`-Schleife auf, die alle Zellen des Bereichs nacheinander durchläuft. Innerhalb der Schleife vergleichen wir mittels der `Select Case`-Anweisung jede einzelne Zelle mit den vorher zwischengespeicherten Variablen `WertMax` und `WertMin`. Tritt eine Übereinstimmung auf, färben wir die jeweilige Zelle über den Einsatz der Eigenschaft `ColorIndex` ein, der wir für die Farbe Grün den Farbindex 4 und für die Farbe Rot den Farbindex 3 zuweisen.





Was denken Sie, warum ich bei der Select Case-Anweisung aus Listing 6.6 keinen Case Else-Zweig eingesetzt habe? Weil mich nur die beiden Extremwerte interessieren. Aber es gibt noch einen weiteren Grund. Der Case Else-Zweig könnte dafür herhalten, alle anderen Zellen, die weder etwas mit dem größten noch kleinsten Wert zu tun haben, zu entfarben. Es könnte schließlich sein, dass ich einen noch größeren Wert im Bereich erfasse. In diesem Fall müssten die ehemals grünen Flecken wieder zurückgesetzt, also entfärbt werden.

Da ich jedoch ganz zu Beginn des Makros den kompletten Bereich entfärbte, wäre die Case Else-Anweisung an dieser Stelle unnötig. Außerdem ist es zweckmäßiger, einen Bereich zu Beginn komplett einmal zu entfarben, als jede nicht zutreffende Zelle in der Schleife mehrfach zu entfarben.

	A	B	C	D	E	F	G	H
1	59	71	88	44	29	95	40	
2	76	89	2	11	41	2	20	
3	72	8	54	97	16	34	56	
4	100	72	19	15	90	28	39	
5	76	42	53	3	92	41	43	
6	29	24	27	10	80	49	14	
7	78	78	61	64	38	52	2	
8	18	90	60	38	80	17	78	
9	81	24	85	57	30	100	83	
10	14	96	37	83	97	93	14	
11								
12								
13								

Abbildung 6.6: Die größten und kleinsten Werte sind gekennzeichnet worden

## Leere Zeilen aus einer Tabelle entfernen

Beim folgenden Beispiel aus Listing 6.7 werden alle leeren Zeilen in einer Tabelle entfernt.

```
Sub LeereZeilenLöschen()
```

```
Dim Zeile As Long
```

```
Dim ZeileMax As Long
```

```
With tbl_LeereZeilen
```

```

ZeileMax = .UsedRange.Rows.Count

For Zeile = ZeileMax To 1 Step -1

    If Application.WorksheetFunction.CountA(.Rows(Zeile)) = 0 Then
        .Rows(Zeile).Delete
    End If

Next Zeile

End With
End Sub

```

*Listing 6.7: Alle leeren  
Zeilen aus einer Tabelle  
entfernen*

Wie Sie im zweiten Kapitel 2 »Die wichtigsten Sprachelemente von Excel-VBA« bereits kennengelernt haben, müssen Sie eine `For...Next`-Schleife, die Zeilen löschen soll, rückwärts laufen lassen. Dazu werden zunächst zwei Variablen vom Typ `Long` benötigt. Die Variable `Zeile` wird benötigt, um die Schleife zu steuern und um die jeweiligen Zeilen ansprechen zu können. In der Variablen `ZeileMax` wird die letzte verwendete Zeile in der Tabelle ermittelt und zwischengespeichert. Die letzte Zeile wird mittels der Eigenschaft `UsedRange`, die den benutzten Bereich einer Tabelle zurückgibt, sowie der Eigenschaft `Rows` und der Zählfunktion `Count` ermittelt. Es werden zu gut Deutsch die Anzahl (`Count`) der Zeilen (`Rows`) im benutzten Bereich (`UsedRange`) ermittelt.

In der rückwärts laufenden Schleife werden die Zeilen von unten nach oben abgearbeitet. Dabei legt `Step -1` die Laufrichtung der Schleife fest. Im Innern der Schleife kommt die Funktion `CountA` zum Einsatz. Dieser Funktion übergeben wir die komplette Zeile, die wir gerade verarbeiten. Meldet die Funktion uns den Rückgabewert `0`, dann dürfen wir die komplette Zeile mithilfe der Methode `Delete` löschen.



Gerade beim Löschen von Zeilen aus einer Tabelle kann es zu längeren Wartezeiten kommen, vor allem dann, wenn sich viele Formeln in der Tabelle befinden. Bei jeder Löschung berechnet Excel alle Zellen neu. Daher können Sie zu Beginn des Makros aus Listing 6.7 die Berechnungsfunktion ausschalten und am Ende wieder einschalten. Die beiden Anweisungen dazu lauten wie folgt:

```

'Berechnung temporär ausschalten
Application.Calculation = xlCalculationManual
'Berechnung wieder einschalten
Application.Calculation = xlCalculationAutomatic

```

## Eigene Funktionen schreiben

## 6.2

Im zweiten Teil dieses Kapitels widmen wir uns dem Thema, wie Sie eigene Funktionen schreiben und in Excel einbinden können. Mithilfe dieser Funktionen sparen Sie sich Schreibarbeit und vermeiden Redundanz im Quellcode.

Ein Beispiel für diese Art von Funktionen wäre eine Funktion, die überprüft, ob eine bestimmte Tabelle in einer Arbeitsmappe vorhanden ist. Da Sie eine derartige Funktionalität in der Programmierung bestimmt sehr oft benötigen, empfiehlt es sich, den entsprechenden Code in eine allgemeine Funktion zu packen und diese dann aus den verschiedenen Makros heraus aufzurufen. Der Vorteil liegt auf der Hand: Wenn Sie die Funktion ändern möchten, dann müssen Sie dies nicht an unzähligen Stellen im Quellcode tun, sondern eben nur in dieser einen Funktion.

### Der Aufbau einer Funktion

Jede Funktionsdefinition beginnt mit dem Schlüsselwort `Function`, gefolgt vom Namen der Funktion, der nahezu beliebig gewählt werden kann. Nach dem Namen folgen in Klammern die Argumente, die der Funktion übergeben werden sollen.

Die allgemeine Syntax für den Bau einer Funktion lautet:

```
[Public | Private [Static] Function Name [(ArgListe)] [As Typ]
[Anweisungen]
[Name = Ausdruck]
[Exit Function]
[Anweisungen]
[Name = Ausdruck]
End Function
```

Nachfolgend finden Sie die Bestandteile dieser Syntax näher erläutert:

- **Public:** Dieses optionale Argument legt fest, dass die Funktion für alle anderen Prozeduren in allen Modulen zugänglich ist. Wird dieses Argument in einem Modul verwendet, das ein optionales Argument `Private` enthält, so ist die Prozedur außerhalb des Projekts nicht verfügbar.
- **Private:** Dieses optionale Argument legt fest, dass die Funktion nur für andere Prozeduren in dem Modul verfügbar ist, in dem sie deklariert wurde
- **Static:** Dieses optionale Argument legt fest, dass die lokalen Variablen der Funktion zwischen den Aufrufen erhalten bleiben. Das Attribut `Static` hat keinen Einfluss auf Variablen, die außerhalb der Funktion deklariert wurden, auch wenn sie in der Prozedur verwendet werden.



- *Name*: Dieses erforderliche Argument legt den Namen der Funktion fest
- *ArgListe*: Optional. Liste der Variablen, die Argumente darstellen, die beim Aufruf an die Funktionsprozedur übergeben werden. Mehrere Variablen werden durch Kommata voneinander getrennt angegeben.
- *Typ*: Dieses optionale Argument legt den Datentyp des von der Funktion zurückgegebenen Werts fest. Zulässige Typen sind `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Decimal` (derzeit nicht unterstützt), `Date`, `String` (ausgenommen feste Länge), `Object`, `Variant` und benutzerdefinierte Typen.
- *Anweisungen*: Hier handelt es sich um die eigentlichen Anweisungen, die in der Funktion ausgeführt werden sollen
- *Ausdruck*: Bei diesem Argument ist der Rückgabewert der Funktion gemeint. Oft wird hier ein boolescher Wert (`True` oder `False`), ein anderer Wert oder ein verarbeiteter Text an die aufrufende Prozedur zurückgegeben.

Nach dieser etwas trockenen Theorie folgen eine Reihe interessanter Beispiele.

## Aktuelle Arbeitsmappe ermitteln

Um zu prüfen, welchen Namen die momentan aktive Arbeitsmappe hat, können Sie eine benutzerdefinierte Funktion schreiben, die den Namen der Arbeitsmappe in eine Zelle schreibt. Die Funktion für diese Aufgabe können Sie in Listing 6.8 sehen:

```
Function AktMap()
```

```
    AktMap = ActiveWorkbook.Name
```

```
End Function
```

Über die Anweisung `ActiveWorkbook.Name` können Sie den Namen der Arbeitsmappe ermitteln. Wenn Sie zusätzlich zum Namen auch noch den Speicherpfad ermitteln möchten, dann erweitern Sie die Funktion aus Listing 6.8 wie folgt:

```
Function AktMapV()
```

```
    AktMapV = ActiveWorkbook.FullName
```

```
End Function
```

Über die Eigenschaft `FullName` können Sie den Namen sowie den Pfad einer gespeicherten Arbeitsmappe ermitteln. Allerdings muss diese Arbeitsmappe vorher einmal gespeichert worden sein.

*Listing 6.8: Den Namen der aktuellen Arbeitsmappe abfragen*

*Listing 6.9: Den Namen sowie den Pfad der Arbeitsmappe abfragen*

## Funktionen testen

Testen Sie die beiden benutzerdefinierten Funktionen im Direktfenster der Entwicklungsumgebung.

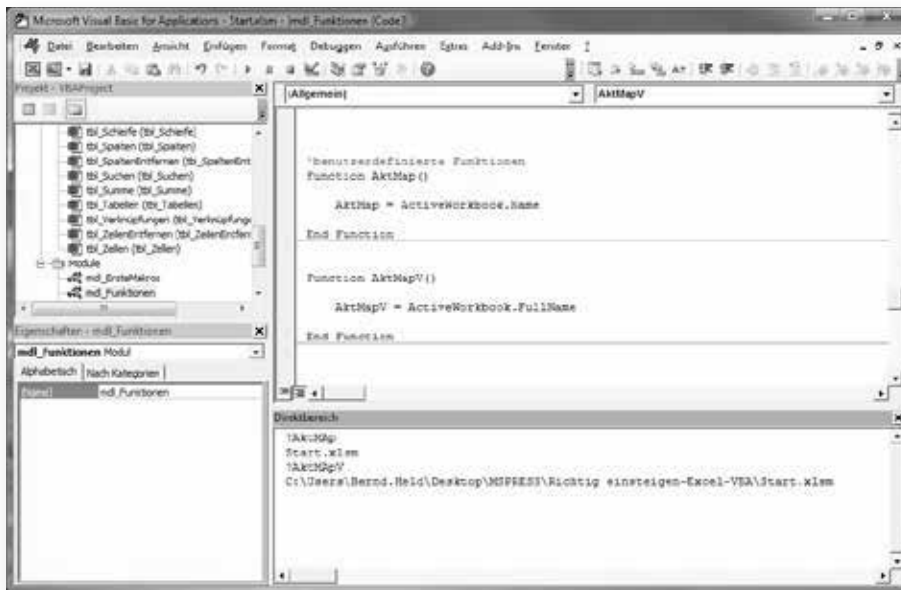



Abbildung 6.7: Die beiden Funktionen machen das, was sie sollen

Anstatt im Direktfenster der Entwicklungsumgebung können Sie die beiden Funktionen direkt in eine Zelle eingeben und mit  bestätigen.

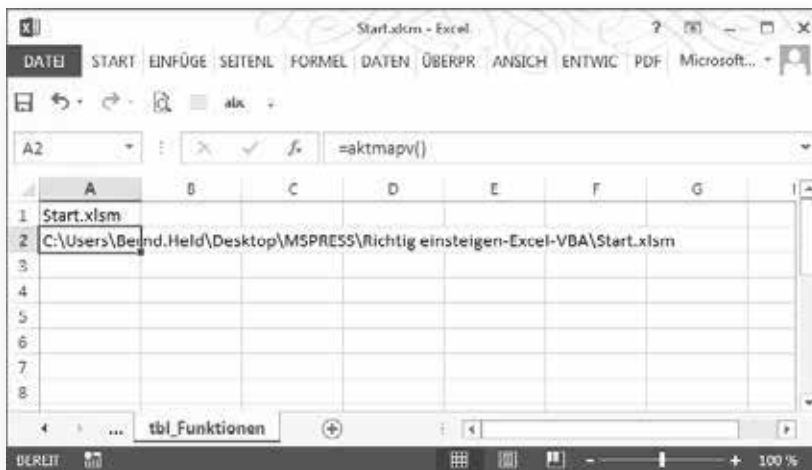


Abbildung 6.8: Kontrolle der Rückgabe von Funktionen direkt in einer Zelle

Eine dritte Variante, um eine Funktion zu testen, ist es, die Funktion über ein Makro aufzurufen. Dies wird exemplarisch im Makro aus Listing 6.10 gezeigt.



```
Sub AufrufFunktion()
```

```
    MsgBox "Aktuelle Mappe: " & AktMap & vbCrLf & _  
        "Pfad- und Name: " & AktMapV
```

```
End Sub
```

*Listing 6.10: Der Aufruf einer Funktion über ein Makro*

## Bestimmte Zeichen aus einer Zelle entfernen

Im folgenden Beispiel werden unerwünschte Zeichen aus Zellen entfernt. Dazu liegt eine Tabelle wie in Abbildung 6.9 gezeigt vor.

	A	B	C	D	E	F	G
1	Nr	Nr bereinigt					
2	123/45\678-01						
3	123/45\678-02						
4	123/45\678-03						
5	123/45\678-04						
6	123/45\678-05						
7	123/45\678-06						
8	123/45\678-07						
9	123/45\678-08						
10	123/45\678-09						
11							
12							
13							

*Abbildung 6.9: Einige Zeichen müssen entfernt werden*

Erfassen Sie jetzt die Funktion aus Listing 6.11, die wir im Anschluss Zeile für Zeile besprechen werden.

```
Function ZeichenRaus(strText As String) As String
```

```
    Dim i As Integer
```

```
    For i = 1 To Len(strText)
```

```
        Select Case Mid(strText, i, 1)
```

```
            Case "-", "/", ":", "\"
```

```
                'Diese Zeichen werden ignoriert
```

```

Case Else
    ZeichenRaus = ZeichenRaus & Mid(strText, i, 1)
End Select

Next i

End Function

```

Listing 6.11:  
*Unerwünschte Zeichen  
werden entfernt*

Nach dem Namen der Funktion aus Listing 6.11 folgt eine Klammer. In dieser Klammer werden die Übergabeargumente definiert. Also das, was die Funktion an Information braucht, um die Aufgabe auszuführen. Für unser Beispiel übergeben wir der Funktion `ZeichenRaus` einen Text. Nach der schließenden Klammer wird der Datentyp dessen bestimmt, was die Funktion zurückliefern soll. Wir übergeben der Funktion also einen Text über die Variable `strText` und bekommen einen geänderten Text (Text ohne die unerwünschten Zeichen) wieder zurück. Der Name der Funktion ist gleichzeitig auch die Rückgabeargumente an das aufrufende Makro, welches den zu verarbeitenden Text übergibt.

Da wir nicht wissen, wie viele Zeichen der Funktion als Text übergeben werden, setzen wir in der Funktion eine Schleife auf, die sich beginnend beim ersten Zeichen, von links nach rechts durch den übergebenen Text zeichenweise durcharbeitet.

Innerhalb der Schleife arbeiten wir mit der `Select Case`-Anweisung sowie mit der Funktion `Mid`, die es uns erlaubt, einzelne Zeichen einer Zeichenfolge auszuwerten. Die Funktion `Mid` hat drei Argumente. Im ersten Argument befindet sich der komplette Text, der unverändert an die Funktion übergeben wurde. Im zweiten Argument befindet sich die Positionsangabe des Zeichens, das gerade verarbeitet wird. Im dritten Argument legen wir fest, wie viele Zeichen ausgewertet werden sollen. Da wir Zeichen für Zeichen nacheinander verarbeiten, setzen Sie dieses Argument auf den Wert `1`.

Die unerwünschten Zeichen werden im ersten `Case`-Zweig, durch Kommata voneinander getrennt und in Anführungszeichen eingefasst, nacheinander angegeben. Dieser Zweig bleibt jedoch ohne Aktion, was letztendlich bedeutet, dass diese Zeichen nicht mehr in das Ergebnis der Funktion übertragen werden.

Außerdem muss die sogenannte Rückgabezeichenfolge wieder Zeichen für Zeichen zusammengesetzt werden. Dies geschieht hier ebenfalls mithilfe der Funktion `Mid`. Mit dem Verkettungsoperator `&` hängen Sie die gültigen Zeichen einfach wieder zusammen. Was jetzt noch fehlt, ist das die Funktion aufrufende Makro, das Sie in Listing 6.12 einsehen können.

```

Sub ZeichenEntfernen()
    'Unerwünschte(s) Zeichen in der Zelle entfernen
    Dim Zeile As Long
    Dim ZeileMax As Long

```



```

With tbl_ZeichenEntfernen
    ZeileMax = .Range("A65536").End(xlUp).Row

    For Zeile = 2 To ZeileMax

        .Range("B" & Zeile).Value = ZeichenRaus(.Range("A" & Zeile).Value)

    Next Zeile

End With

End Sub

```

*Listing 6.12: Das Makro ZeichenEntfernen ruft die Funktion ZeichenRaus auf*

In einer For...Next-Schleife wird die Tabelle *tbl\_ZeichenErsetzen* Zeile für Zeile von oben nach unten abgearbeitet. Innerhalb der Schleife wird die Funktion *ZeichenRaus* aufgerufen. Dabei wird der jeweilige Inhalt der jeweiligen Zelle aus Spalte A übergeben. Jetzt wird die übergebene Zeichenfolge von der Funktion zerlegt und die gültigen Zeichen wieder zusammengesetzt, die dann an das aufrufende Makro zurückgegeben werden. Der geänderte Inhalt wird direkt in Spalte B geschrieben.



Führen Sie das Makro *ZeichenEntfernen* einmal schrittweise aus, indem Sie mit der Taste **F8** arbeiten. Kontrollieren Sie, was dabei jeweils in der Funktion ankommt und was wieder zurückgeliefert wird. Streichen Sie dazu mit der Maus jeweils über die Variablen *strText* und *ZeichenRaus*.

	A	B	C	D	E	F	G
1	<b>Nr</b>	<b>Nr bereinigt</b>					
2	123/45\678-01	1234567801					
3	123/45\678-02	1234567802					
4	123/45\678-03	1234567803					
5	123/45\678-04	1234567804					
6	123/45\678-05	1234567805					
7	123/45\678-06	1234567806					
8	123/45\678-07	1234567807					
9	123/45\678-08	1234567808					
10	123/45\678-09	1234567809					
11							
12							
13							

*Abbildung 6.10: Die unerwünschten Zeichen wurden eliminiert*



## Kalenderwoche nach DIN ermitteln

Das folgende Beispiel ermittelt anhand eines Datums die Kalenderwoche nach DIN 28601. Eigentlich gibt es eine Standardfunktion in Excel, über die Sie die Kalenderwoche ermitteln können. Die Funktion `KALENDERWOCHE()` rechnet jedoch nach amerikanischem Standard. Die erste Woche des Jahres ist nach deutschem Standard definiert als die Woche, die aus mindestens vier Tagen besteht. Daher muss eine eigene benutzerdefinierte Funktion her, die Sie in Listing 6.13 einsehen können.

```
Function KalenderwocheNachDIN(Datum As Date) As Integer
    KalenderwocheNachDIN = Format(Datum, "WW", vbMonday, vbFirstFourDays)
End Function

Sub KwNachDIN()

    MsgBox KalenderwocheNachDIN(Date)

End Sub
```

*Listing 6.13: Die Kalenderwochen-ermittlung nach DIN 28601*

Das aufrufende Makro übergibt an die Funktion das aktuelle Tagesdatum. In der Funktion selbst wird die Funktion `Format` verwendet, um die Kalenderwoche über das Kürzel `WW` abzufragen. Dabei geht man von dem Standard aus, dass die Woche mit Montag beginnt und dass die erste Kalenderwoche des Jahres mindestens vier Tage beinhalten muss. So werden demnach auch die Konstanten für die Funktion `Format` eingesetzt.

## Die Existenz einer Tabelle prüfen

Bevor Sie einer Tabelle einen Namen geben, sollten Sie prüfen, ob der Name nicht bereits in der Arbeitsmappe vergeben ist. Zu diesem Zweck können Sie die Prüffunktion aus Listing 6.14 einsetzen.

```
Function BlattDa(StrTab As String) As Boolean
    Dim Blatt As Worksheet

    BlattDa = False

    For Each Blatt In ThisWorkbook.Worksheets

        If Blatt.Name = StrTab Then
            BlattDa = True
            Exit Function
        End If
    End For
End Function
```



*Listing 6.14: Sicherheitshalber einmal prüfen, ob eine bestimmte Tabelle in der Arbeitsmappe existiert*

```
Next Blatt
```

```
End Function
```

Die Funktion `BlattDa` bekommt als Übergabeargument den Namen der Tabelle, deren Existenz sie in der Arbeitsmappe prüfen soll. Zu Beginn der Funktion aus Listing 6.14 setzen Sie die »Erwartungshaltung« der Rückgabe auf `False`. Sie gehen im Prinzip erst einmal davon aus, dass sich die gesuchte Tabelle nicht in der Arbeitsmappe befindet. Diese Zeile wäre eigentlich gar nicht notwendig, doch sie veranschaulicht das zugrunde liegende System besser.

In einer anschließenden `For Each...Next`-Schleife wird jede einzelne Tabelle der Arbeitsmappe abgearbeitet. Innerhalb der Schleife wird der jeweilige Blattname mit dem übergebenen Tabellennamen verglichen. Ist dieser gleich, dann setzen Sie den Rückgabewert der Funktion auf den Wert `True` und verlassen über die Anweisung `Exit Function` die Funktion auf dem schnellsten Weg. Es ist ja schließlich nicht notwendig, in einer Arbeitsmappe mit zwanzig Tabellen die restlichen achtzehn noch abzuarbeiten, wenn bereits nach der zweiten Tabelle eine Übereinstimmung gefunden wurde.

## Die Existenz einer Datei prüfen

Bevor Sie eine Arbeitsmappe öffnen, sollten Sie prüfen, ob die angegebene Arbeitsmappe überhaupt existiert. Wie das geht, sehen Sie in der Funktion aus Listing 6.15.

```
Function DateiDa(strDatei As String) As Boolean
```

```
    If Dir(strDatei) <> "" Then
        DateiDa = True
    End If
```

```
End Function
```

```
Sub Aufruf()
```

```
    Dim strMappe As String
```

```
    strMappe = ThisWorkbook.Path & "\Ausgabe.xls"
```

```
    If DateiDa(strMappe) = True Then
```

```
        Workbooks.Open strMappe
```

```
    Else
```

```
        MsgBox "Die Mappe " & strMappe & " ist nicht verfügbar!"
```

```
    End If
```

```
End Sub
```

*Listing 6.15: Besser vor dem Öffnen prüfen, ob eine Mappe überhaupt existiert*

Die Funktion `DateiDa` erwartet als notwendige Information den Namen sowie den Pfad der Arbeitsmappe. In der Funktion selbst wird die Funktion `Dir` verwendet, um zu prüfen, ob am gemeldeten Ort unter dem übergebenen Namen überhaupt die Datei zur Verfügung steht. Wenn ja, dann benutzen Sie als Rückgabewert `True`.

## Die Existenz eines Verzeichnisses prüfen

Eine Ebene weiter nach oben kommt nun das Verzeichnis an die Reihe. Bevor Sie ein Verzeichnis anlegen, müssen Sie sicher sein, dass das Verzeichnis nicht bereits existiert. In diesem Fall würde Excel nämlich einen Laufzeitfehler melden.

In der Funktion aus Listing 6.16 wird ebenfalls die Funktion `Dir` verwendet. Dabei wird geprüft, ob es ein Verzeichnis mit dem Namen *Daten* unterhalb des Verzeichnisses gibt, in dem die Arbeitsmappe *Start.xlsm* gespeichert ist.

```
Function OrdnerDa(strOrdner As String) As Boolean
```

```
    If Dir(strOrdner, vbDirectory) = "" Then
        OrdnerDa = False
    Else
        OrdnerDa = True
    End If
```

```
End Function
```

```
Sub AufrufVerzeichnisAnlegen()
```

```
    Dim strVerzeichnis As String
```

```
    strVerzeichnis = ThisWorkbook.Path & "\Daten\"
```

```
    If OrdnerDa(strVerzeichnis) = False Then
```

```
        'Verzeichnis anlegen
```

```
        MkDir strVerzeichnis
```

```
    Else
```

```
        MsgBox "Das Verzeichnis " & strOrdner & " ist bereits angelegt!"
```

```
    End If
```

```
End Sub
```

Die Funktion `OrdnerDa` erwartet als notwendige Information den Namen des Pfades, der angelegt werden soll. Mithilfe der Funktion `Dir`, bei der Sie im zweiten Argument die Konstante `vbDirectory` angeben, wird geprüft, ob das Verzeichnis bereits angelegt ist. Wenn ja, dann liefert uns diese Funktion den Namen des Verzeichnisses. Liefert die Funktion als Rückgabe eine leere Zeichenfolge, dann gibt es dieses Verzeichnis noch nicht. In diesem Fall wenden Sie die Anweisung `MkDir` an, um ein neues Verzeichnis zu erstellen.

*Listing 6.16: Vor dem Anlegen eines Verzeichnisses erst eine Prüfung durchführen*



## Funktionen im Funktionsassistenten einsehen

Alle bis jetzt geschriebenen Funktionen können im Funktionsassistenten von Excel betrachtet werden. Setzen Sie dazu den Mauszeiger in eine beliebige Zelle und klicken in der Bearbeitungsleiste von Excel auf das Symbol *Funktion einfügen*. Stellen Sie danach im Dialogfeld *Funktion einfügen* im Dropdown-Listefeld *Kategorie auswählen* die Kategorie *Benutzerdefiniert* ein.

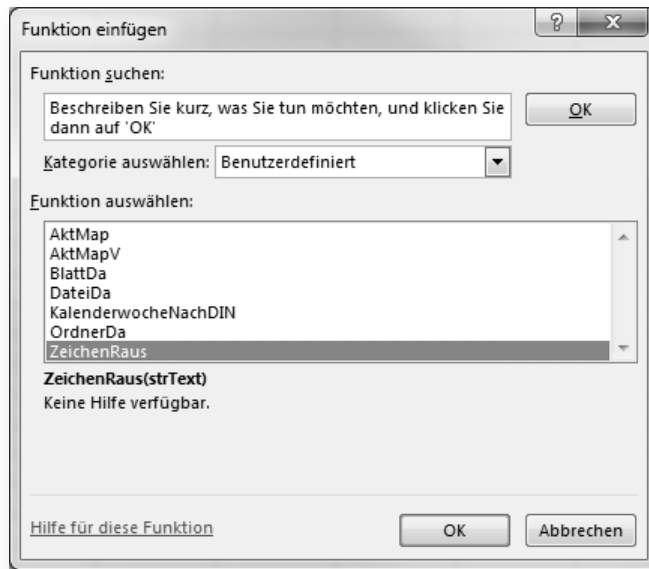


Abbildung 6.11: Alle benutzerdefinierten Funktionen können hier eingesehen werden

## Funktionen in eine andere Funktionskategorie hängen

Wie Sie in Abbildung 6.11 sehen, werden automatisch alle Funktionen, die Sie geschrieben haben, in der Rubrik *Benutzerdefiniert* angeboten. Dort tummeln sich unter Umständen aber auch weitere Funktionen, die von Drittanbietern über Add-Ins zur Verfügung gestellt werden.

Wie wäre es denn, wenn Sie Ihre eigenen Funktionen in einer separaten Kategorie anordnen könnten? Das können Sie ganz leicht bewerkstelligen. Im Makro aus Listing 6.17 werden die Funktionen *BlattDa*, *DateiDa* und *OrdnerDa* in der Kategorie *Held* einsortiert.

```
Sub FunktionInEigeneKategoriezuweisen()
```

```
Application.MacroOptions Macro:="BlattDa", _
Description:="Prüft, ob eine Tabelle existiert", Category:="Held"
```

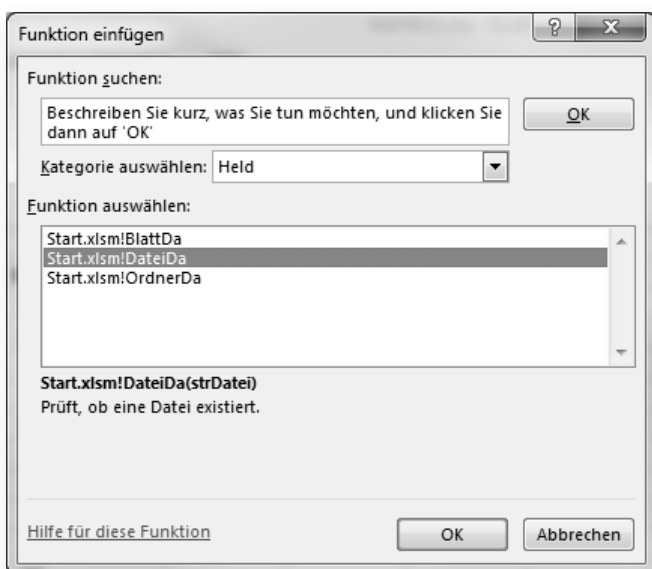
```
Application.MacroOptions Macro:="DateiDa", _
Description:="Prüft, ob eine Datei existiert", Category:="Held"
```

```
Application.MacroOptions Macro:="OrdnerDa", _
Description:="Prüft, ob ein Ordner existiert", Category:="Held"
```

End Sub

*Listing 6.17: Funktionen in eine andere/neue Kategorie umziehen*

Mithilfe der Methode `MacroOptions` können Sie eine Funktion in eine bestimmte Kategorie oder sogar in eine neue Kategorie einordnen lassen. Über den etwas widersprüchlichen Parameter `Macro` geben Sie den Namen der Funktion an, welche umziehen soll. Im Parameter `Description` können Sie die Aufgabe der Funktion etwas näher beschreiben. Im Parameter `Category` geben Sie den gewünschten Namen für Ihre eigene Kategorie an, in der Ihre Funktion angeboten werden soll.



*Abbildung 6.12: Die eigenen Funktionen in einer separaten Kategorie anbieten*

Das Makro aus Listing 6.17 muss jedoch immer dann automatisch gestartet werden, wenn Sie die Mappe, die die Funktionen enthält, öffnen. Und da sind wir bereits fast schon beim folgenden Kapitel, bei dem es um Ereignisprogrammierung geht, gelangt.

Gehen Sie wie folgt vor, um dieses Makro automatisch immer beim Öffnen der Arbeitsmappe auszuführen.

- 1 Drücken Sie die Tastenkombination `[Alt] + [F11]`, um in die Entwicklungsumgebung von Excel zu gelangen.
- 2 Führen Sie im Projekt-Explorer einen Doppelklick auf der Rubrik *DieseArbeitsmappe* aus.
- 3 Stellen Sie oberhalb des Codefensters im Dropdown-Listenfeld das Objekt *Workbook* ein.
- 4 Ergänzen Sie das sich nun selbst einstellende Ereignisgerüst wie in Abbildung 6.13 gezeigt.



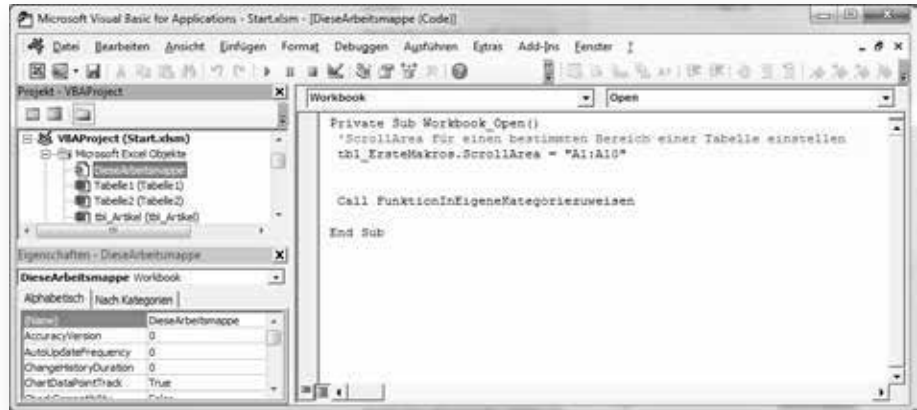


Abbildung 6.13: Das bereits eingestellte Ereignis erweitern

Wir hatten bereits im ersten Kapitel das Ereignis `Workbook_Open` dazu verwendet, einen Bildlaufbereich (`ScrollArea`) dauerhaft anzulegen. Jetzt wenden Sie die Anweisung `Call` an, um das Makro `FunktionInEigeneKategoriezuweisen` aufzurufen.



Rein theoretisch brauchen Sie die Anweisung `Set` vor dem Makronamen nicht. Es würde auch nur der Name des Makros reichen, um ein Makro aufzurufen. Ich denke nur, dass der Quellcode mit dieser Anweisung besser lesbar ist. Sie sehen beim `Call` genau, dass jetzt ein anderes Makro aufgerufen wird.

Übrigens ist `Call` ein Sprung mit Wiederkehr, d.h., nach Abarbeiten des aufgerufenen Makros wird der nachfolgende Quellcode ausgeführt.

```
Call Aufgabe1
Call Aufgabe2
Call Aufgabe3
```

Auf diese Art und Weise können Sie größere Aufgaben in kleinere Aufgaben gliedern und nacheinander Stück für Stück lösen. Am Ende rufen Sie die einzelnen Bauteile sukzessive auf. Generell würde ich Ihnen so oder so diese Art der Umsetzung empfehlen. Auch im Hinblick auf das Aufspüren von Fehlern ist es meiner Ansicht nach besser, einen Fehler in einem kleineren Bauteil zu finden und zu beheben, als sich durch seitenlange Makros durcharbeiten zu müssen.

## Zusammenfassung

6.3

Sie haben in diesem Kapitel gelernt, wie Sie die Standardtabellenfunktionen auch in Ihren Makros verwenden können. Des Weiteren haben Sie erfahren, wie Sie eigene Funktionen stricken, testen und einsetzen können.

Im nächsten Kapitel »Die Ereignisprogrammierung in Excel« lernen Sie, wie Sie Ereignisse in Excel einstellen und erweitern können. Eines der bekanntesten Ereignisse dabei ist das Ereignis `Workbook_Open`, das automatisch beim Öffnen einer Arbeitsmappe ausgeführt wird. An diese Ereignisse werden wir uns dranhängen und weitere Aktionen einstellen.

## Die Lernkontrolle

6.4

Zum Abschluss dieses Kapitels stelle ich Ihnen ein paar Verständnisfragen. Die Antworten auf die hier gestellten Fragen finden Sie im Anhang dieses Buchs.

- Welche Möglichkeiten haben Sie, um eine deutsche Tabellenfunktion zu übersetzen?
- Wie heißt die Funktion, um mehrere Extremwerte zu ermitteln?
- Wie heißt die Methode, um eine Funktion in einer anderen Funktionskategorie anzuordnen?

