

Excel:

Zahlen · rechnen · Formeln

René Martin

12 Eigene Funktionen erstellen

Seit der Version 5.0 liegt unter Excel eine Programmiersprache mit Namen Visual Basic for Applications (VBA). Sinn dieser Makrosprache ist es, per Programmierung auf Tabellen zuzugreifen, Informationen zu verarbeiten und in andere Tabellen zurück schreiben. Dies kann dialoggesteuert funktionieren: Der Benutzer hat eine Eingabemaske vor sich, auf der er in Eingabefelder etwas einträgt, Optionsbuttons und Kontrollkästchen anklickt, etwas aus Listen auswählt und schließlich auf Schaltflächen klickt und so die Bestätigung seiner Aktion einleitet. Dies alles zu erläutern würde den Rahmen des vorliegenden Buchs sprengen. Man benötigt keine profunden Kenntnisse sondern lediglich einige Grundbegriffe der Programmiersprache VBA, um nun eigene, benutzerdefinierte Funktionen in Excel zu erstellen. Denn: Trotz der zirka 350 Funktionen, die Excel zur Verfügung stellt, kann es durchaus sein, dass eine Funktion für die tägliche Arbeit fehlt. Wie man solche Funktionen selbst erstellen kann, wird im Folgenden gezeigt.

12.1 Die Entwicklungsumgebung

Um den VBA-Editor zu öffnen, drücken Sie in Excel die Tastenkombination [Alt] + [F11]. Oder über die Excel-Optionen: Dort können Sie sich in der Kategorie „Menüband anpassen“ die Entwicklerregisterkarte anzeigen lassen. Sie finden diese Einstellung auch im Kontextmenü der anderen Registerkarten. Im nun erscheinenden Register „Entwicklertools“ können Sie den „Visual Basic-Editor“ öffnen.

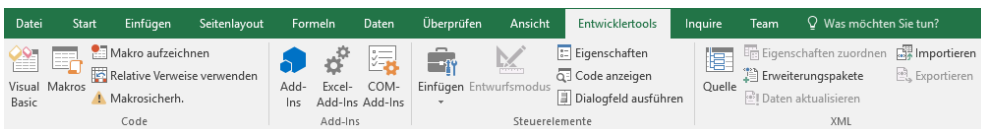


Abbildung 12.1 Die Registerkarte „Entwicklertools“ in Excel

Der Projekt-Explorer zeigt Ihnen an, dass Sie sich im VBA-Projekt „Mappe1“ befinden oder in dem Projekt, das zu Ihrer Datei gehört. Sollte der Projekt-Explorer nicht sichtbar sein, so können Sie ihn über das Menü Ansicht | Projekt-Explorer herholen. Mit der rechten Maustaste oder dem Menü Einfügen | Modul öffnen Sie das Codefenster (das Modulfenster), wo Sie nun beginnen können zu programmieren. Auf den folgenden Seiten finden Sie vorab weitere Erläuterungen zur Codeeingabe in VBA.

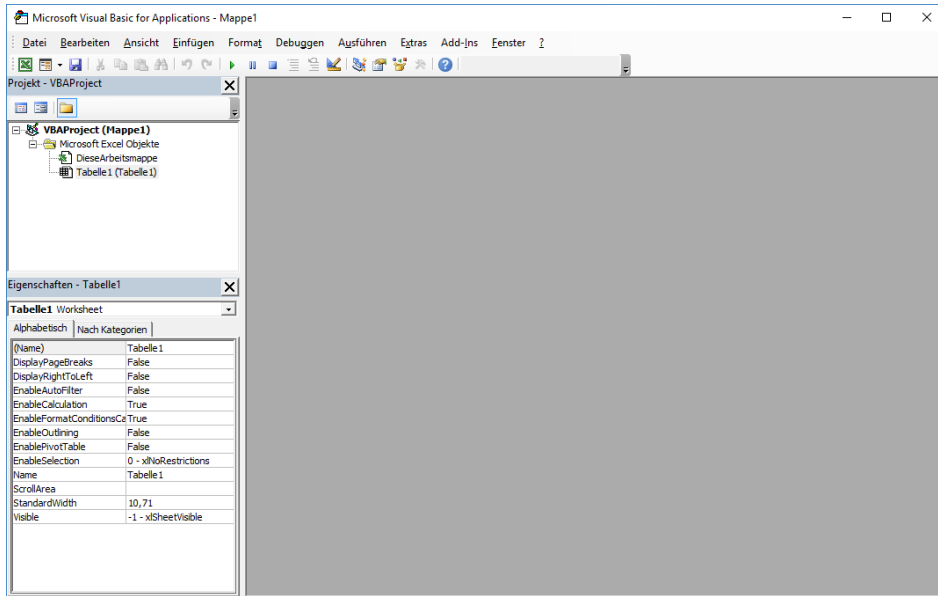


Abbildung 12.2 Der Visual Basic Editor

Hinweis

Die Sprache VBA und auch der Editor haben sich seit Excel 97 nur geringfügig erweitert; die folgenden Beispiele gelten also für alle Versionen ab Excel 8.0 (Excel 97).

12.1.1 Die Codeeingabe

Wenn Sie ein Programm, auch Prozedur oder Makro genannt, erstellen wollen, so beginnen Sie mit dem Schlüsselwort `Sub`. Danach folgt der Name der Prozedur, beispielsweise:

```
Sub Jekt
```

oder:

```
Sub Altern
```

Wenn danach die Eingabetaste gedrückt wird, erscheint zwei Zeilen tiefer ein

```
End Sub
```

während hinter den Namen zwei Klammern gesetzt werden:

```
Sub Altern()
```

```
End Sub
```

Während jede Prozedur mit dem Schlüsselwort `Sub` gefolgt von einem Namen beginnt, so fangen Funktionen mit dem Schlüsselwort `Function` an, denen der Name folgt.

```
Function RechneNeu()
```

```
End Function
```

Der Name darf bis zu 80 Zeichen lang sein. Alle Zeichen außer Leerzeichen, Satz- und Sonderzeichen: `.,:~!"$%&/(){<>}^` sind erlaubt.

Achtung

Auch der Bindestrich "-" darf nicht verwendet werden, der Unterstrich "_" dagegen schon. Die Groß- und Kleinschreibung können Sie ebenso verwenden wie die zehn Ziffern (VBA macht, ebenso wie Excel, keine Unterschiede).

Ein gültiger Funktionsname wäre:

```
Function Mein_erstes_kleines_Programm_von_heute()
```

Der Name der Funktion oder der Prozedur muss selbstverständlich eindeutig sein und darf nicht mit einem Schlüsselwort übereinstimmen. Was sind Schlüsselwörter? Schlüsselwörter sind

- Objekte der verschiedenen Programme: `ActiveWorksheet`, `Cells`, `Font`, `Table` ...
- Alle Befehle und Anweisungen: `GoTo`, `Dim`, `As`, `If`, `While` ...
- Alle Funktionen: `Sin`, `Cos`, `Date`, `Year` ...
- Alle logischen Operatoren: `And`, `Or`, `Not` ...

Die Klammern werden bei Funktionen mit Parametern gefüllt. Dies wird in Kapitel 12.2 erläutert.

Nun wird zwischen

```
Function Mein_erstes_kleines_Programm_von_heute()
```

```
End Function
```

der Programmcode eingegeben. Die Texteingabe und -korrektur im Codefenster erfolgt nach den gleichen Regeln wie in der Textverarbeitung:

Es kann nun passieren, dass Sie eine sehr lange Befehlszeile eingeben, zum Beispiel:

```
Anzeigetext = MsgBox("Möchten Sie den Text sehen?", vbYesNo
& vbQuestion, "Text sehen")
```

Ein automatischer Umbruch, wie von der Textverarbeitung bekannt, findet erst nach 1.024 Zeichen statt: Der Text fließt weiter und weiter nach rechts. Der Text kann jedoch in mehrere Zeilen geteilt werden: Am Ende der Zeile steht ein Unterstrich, vor dem sich eine Leerstelle (sie ist unbedingt nötig!) befindet.

```
Anzeigetext = MsgBox ("Möchten Sie den Text sehen?", _
vbYesNo + vbQuestion, "Text sehen")
```

Häufig wünscht der Benutzer seine Programme zu kommentieren; sei es, um sie besser zu strukturieren, sei es, um anderen Programmierern, die damit weiterarbeiten, schnell einen Überblick zu verschaffen. Kommentare können überall eingefügt werden: Sie werden mit einem Apostroph „'“ eingeleitet, das innerhalb einer Zeile stehen kann:

```
MsgBox "x1*x2 = q ( -(x1 + x2) = p" 'Satz von Vieta
```

oder am Anfang einer Zeile:

```
' Nun folgt ein Meldungsfenster:
MsgBox "x1*x2 = q ( -(x1 + x2) = p"
' Dies ist der Satz von Vieta
```

Die Groß- und Kleinschreibung spielt in VBA keine Rolle. Eine praktische Hilfe übrigens: Sie können Tippfehler vermeiden, wenn Sie sämtliche VBA-Befehle in Kleinbuchstaben schreiben und nach Drücken der [Enter]-Taste überprüfen, ob sie verändert wurden. Wenn ja, so war die Eingabe korrekt (zumindest was ihre Schreibweise anbelangt), wenn nein, so haben Sie sich vertippt.

Damit die Funktion wieder etwas ausgibt, wird an sie ein Wert übergeben. Die folgende Funktion berechnet das morgige Datum mit Hilfe der Funktion Date, also das heutige Datum. Zum aktuellen Datum wird ein (Tag) addiert:

```
Function Morgen()
Morgen = Date + 1
End Function
```

Man kann das Datum auch an eine Variable übergeben:

```
Function Morgen()  
    Dim MorgigesDatum As Date  
    MorgigesDatum = Date + 1  
    Morgen = MorgigesDatum  
End Function
```

Die Namen der Variablen dürfen bis zu 255 Zeichen enthalten, müssen mit einem Buchstaben beginnen und dürfen nicht mit einem VBA-Schlüsselwort identisch sein.

Ihnen stehen verschiedene Variablentypen zur Verfügung:

Tabelle 12.1 Liste der Variablentypen in VBA

Datentyp	Variablentyp	Typen- kennzei- chen	Bereich	Speicherplatz (in Bytes)	Präfix
Ja/Nein	Boolean		0 (False) und -1 (True)	2	bln
Ganzzahlen	Byte		0 – 255	1	byt
	Integer	%	-32.768 bis 32.767	2	int
	Long	&	-2.147.483.648 bis 2.147.483.647	4	lng
Dezimalzahlen	Single	!	$-3,402823 \cdot 10^{38}$ bis $3,402823 \cdot 10^{38}$	4	sng
	Double	#	$-1,79769 \cdot 10^{324}$ bis $1,797693 \cdot 10^{324}$	8	dbl
	Currency	@	$-9,22 \cdot 10^{14}$ bis $9,22 \cdot 10^{14}$	8	cur
Datumzahlen	Date		1.1.100 bis 31.12.9999	8	dat
Text	String	\$	zirka 2 Milliarden	10 + Länge der Zeichenkette	str
Wechselnde Typen	Variant		jeder numerische Wert im Bereich Double, jeder String	22 + Länge der Zeichenkette	var

Viele VBA-Programmierer kennzeichnen Variablen, indem sie drei Zeichen vor den Namen setzen. So wird deutlich, dass es sich bei `datMorgen` um eine Datumsvariable handelt, bei `strZuname` um eine Zeichenkette. Die Konvention, die von der Firma Gregory Reddick & Associates, einer Unternehmensberatungsfirma von Microsoft, herausgegeben wurde, ist nicht verbindlich. Allerdings arbeiten sehr viele Programmierer damit. Diese Konvention stellt eine Möglichkeit der Standardisierung für VBA-Programmierung dar. Sie müssen sich natürlich nicht daran halten, sollten sich aber, wenn Sie zu mehreren an einem Projekt programmieren, über die Benennung der Variablen einig sein.

Das obige Beispiel könnte also wie folgt aussehen:

```
Option Explicit
```

```
Function Morgen() As Date
    Dim datMorgen As Date
    datMorgen = Date + 1
    Morgen = datMorgen
End Function
```

Die Zeile

```
Option Explicit
```

weist darauf hin, dass jede Variable deklariert werden muss. Dies hat eine Reihe von Vorteilen: Sie haben bessere Kontrolle über die Variablen, sparen Speicherplatz und vermeiden unnötige Tippfehler.

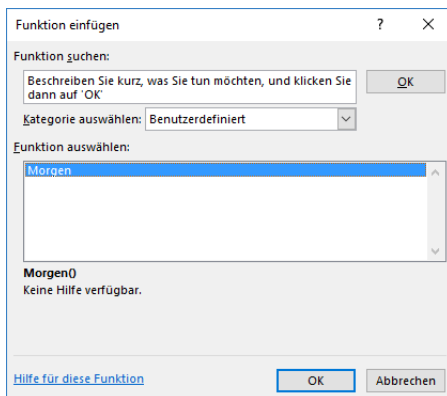


Abbildung 12.3 Die erste benutzerdefinierte Funktion

Angenommen, Sie haben nun in VBA eine Funktion erstellt. Nun steht diese Funktion in Excel zur Verfügung, wo sie per Funktionsassistent aus der benutzerdefinierten Kategorie ausgewählt oder direkt in ein Tabellenblatt eingegeben werden kann.

12.1.2 Rechnen und verknüpfen

Beim Rechnen in VBA gelten die gleichen Regeln wie in Excel. Im Beispiel

```
x = 2^5
```

erhält x den Wert 32,

im Beispiel

```
x = 36 ^ 0.5
```

den Wert 6.

Achtung

Bitte beachten Sie: Dezimalstellen werden im Quellcode mit einem Punkt (".") geschrieben, nicht mit Komma (","). Die Eingabe (zum Beispiel in Excel) erfolgt allerdings mit einem Komma, da hier die Systemsteuerung greift.

Achtung

Mit Prozenten kann in VBA nicht gerechnet werden!

```
BuchMWSt = BuchPreis * 7%
```

führt zu einer Fehlermeldung. Richtig wäre:

```
BuchMWSt = BuchPreis * 0.07
```

Tabelle 12.2 Zusammenfassung der mathematischen Funktionen.

(Funktions-)Name in VBA	Name in Excel	Bedeutung
+, -, *, /	+, -, *, /	Dürfte bekannt sein ...
\	GANZZAHL	Ganzzahliges Ergebnis der Division
MOD	REST	Ganzzahliger Rest bei der Division
SQR	WURZEL	Quadratwurzel
SIN	SIN	Sinus
COS	COS	Cosinus
TAN	TAN	Tangens

(Funktions-)Name in VBA	Name in Excel	Bedeutung
ATN	ARCTAN	Der Arkutangens, die Umkehrfunktion des Tangens
EXP	EXP	Exponentialfunktion auf Basis e
LOG	LN	Der natürliche Logarithmus zur Basis e
ABS	ABS	Gibt den Absolutwert einer Zahl zurück:
INT	GANZZAHL	Gibt einen Wert zurück, der den gleichen Typ wie der übergebene Wert hat und den ganzzahligen Anteil einer Zahl enthält:
FIX	KÜRZEN	Gibt einen Wert zurück, der den gleichen Typ wie der übergebene Wert hat und den ganzzahligen Anteil einer Zahl enthält:
SGN	VORZEICHEN	Das Vorzeichen einer Zahl: 0, 1 oder -1
ROUND	RUNDEN	Rundet eine Zahl (erst ab Excel 9.0 vorhanden)
RND	ZUFALLSZAHL	Eine Zufallszahl

Neben den Rechenfunktionen stellt VBA eine Reihe finanzmathematischer Funktionen zur Verfügung, die bereits im entsprechenden Kapitel erläutert wurden.

Tabelle 12.3 Die folgende Tabelle listet die Funktionsnamen, ihren Excel-Namen und ihre Bedeutung auf.

Funktionsname	Funktionsname in Excel	Bedeutung
DDB	GDA	Die Abschreibung eines Anlageobjekts nach der geometrisch degressiven Methode für einen spezifischen Zeitraum
SYD	DIA	Die Abschreibung eines Anlageobjekts nach der arithmetisch degressiven Methode für einen spezifischen Zeitraum
SLN	LIA	Die Abschreibung eines Anlageobjekts nach der linearen Methode für einen spezifischen Zeitraum
FV	ZW	Der Endwert einer Annuität ausgehend von regelmäßigen Zahlungen und einem konstanten Zinssatz
RATE	ZINS	Der Zinssatz einer Annuität
IRR	IKV	Der interne Zinsfluss für regelmäßige Cash-flows

Die Entwicklungsumgebung

Funktionsname	Funktionsname in Excel	Bedeutung
MIRR	QIKV	Der modifizierte interne Zinsfluß für regelmäßige Cash-Flows
IPMT	ZINSZ	Der Zinsanteil einer Annuität für einen spezifischen Zeitraum
PMT	RMZ	Die Zahlung einer Annuität
PPMT	KAPZ	Der Kapitalanteil einer Annuität
NPV	NBW	Der Netto-Barwert für regelmäßige Cash-Flows
PV	BW	Der Barwert einer Annuität

Die vollständige Liste aller Funktionen finden Sie im Objektkatalog. Sie öffnen ihn über die Funktionstaste [F2], über das Symbol oder über den Befehl Ansicht | Objektkatalog.

Ebenso wie Zahlen durch Rechenoperationen miteinander verknüpft werden, können auch Textblöcke verknüpft werden. Zwei (oder mehrere) Zeichenketten können mit dem Zeichen + oder dem Zeichen & verbunden werden (hier sind beide möglich). Dies ist bei der Ausgabe längerer Texte nötig, da innerhalb der Textzeile nicht mit dem Unterstrich "_" umbrochen werden darf. Beispielsweise:

```
Function Busch()
```

```
    Dim strLämpel1 As String, strLämpel2 As String  
    Dim strLämpel3 As String, strLämpel4 As String  
    Dim strLämpel5 As String, strLämpel6 As String  
    Dim strLämpel7 As String, strLämpel8 As String  
    Dim strLämpel9 As String, strLämpel10 As String
```

```
    strLämpel1 = "Also lautet ein Beschluss"  
    strLämpel2 = "Dass der Mensch was lernen muss"  
    strLämpel3 = "Nicht allein das Abc"  
    strLämpel4 = "Bringt den Menschen in die Höh"  
    strLämpel5 = "Nicht allein in Schreiben, Lesen"  
    strLämpel6 = "Übt sich ein vernünftig Wesen"  
    strLämpel7 = "Nicht allein in Rechnungssachen"
```

```

strLämpel8 = "Soll der Mensch sich Mühe machen"
strLämpel9 = "Sondern auch der Weisheit Lehren"
strLämpel10 = "Muss man mit Vergnügen hören"
Busch = strLämpel1 & " / " & strLämpel2 & " / " & _
strLämpel3 & " / " & strLämpel4 & " / " & _
strLämpel5 & " / " & strLämpel6 & " / " & _
strLämpel7 & " / " & strLämpel8 & " / " & _
strLämpel9 & " / " & strLämpel10

```

End Function

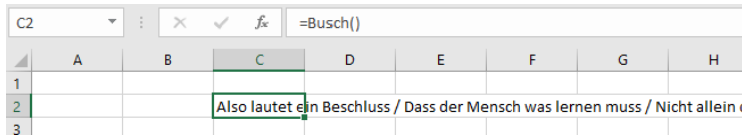


Abbildung 12.4 Eine Zeichenkette als Ergebnis einer Funktion

In der folgenden Tabelle finden Sie eine Zusammenfassung der String-Funktionen und ihre Excel-Entsprechung.

Tabelle 12.4 Textfunktionen

Funktionsname	Excel-Funktion	Bedeutung
Left	LINKS	Schneidet eine bestimmte Anzahl von Zeichen von links ab
Right	RECHTS	Schneidet eine bestimmte Anzahl von Zeichen von rechts ab
Mid	TEIL	Schneidet eine bestimmte Anzahl von Zeichen aus der "Mitte" heraus, das heißt ab einer bestimmten Position
InStr	SUCHEN, FINDEN	Überprüft, ob eine Zeichenfolge innerhalb einer Zeichenkette vorhanden ist, und gibt die Position an
LTrim		Löscht Leerzeichen am Anfang eines Strings
RTrim		Löscht Leerzeichen am Ende eines Strings
Trim	GLÄTTEN	Löscht Leerzeichen am Anfang und Ende eines Strings
Len	LÄNGE	Ermittelt die Länge einer Zeichenkette

Die Entwicklungsumgebung

Funktionsname	Excel-Funktion	Bedeutung
Chr	ZEICHEN	Wandelt einen ASCII-Code in einen String um
Asc	CODE	Wandelt einen String in die entsprechende Zahl des ASCII-Codes um
Val	WERT	Wandelt einen Zahlenstring in eine Zahl um
Str	TEXT	Wandelt eine Zahl in einen String um
LCase	KLEIN	Wandelt eine Zeichenkette in Kleinbuchstaben um
UCase	GROSS	Wandelt eine Zeichenkette in Großbuchstaben um
StrComp	IDENTISCH	Vergleicht zwei Zeichenketten. Ist die erste Zeichenkette größer als die zweite, wird -1 zurückgegeben, im umgekehrten Fall 1. Sind beide gleich: 0. Ist einer der beiden Strings leer, so wird Null übergeben.
Space		Gibt eine Folge von Leerzeichen aus
Split		Trennt eine Zeichenfolge und liefert einen Array (ab Excel 9.0)
Join		Setzt eine Zeichenfolge zusammen (ab Excel 9.0)
Filter		Durchsucht eine Zeichenfolge (ab Excel 9.0)
InStrRev("abcd", "ä")		Überprüft von hinten, ob eine Zeichenfolge in einer anderen vorhanden ist (ab Excel 9.0)

Auch die Vergleichsoperatoren und Verknüpfungskonjunkturen sind ähnlich wie in Excel.

Tabelle 12.5 Die Liste der Vergleichsoperatoren

Operator	Beschreibung	Operator	Beschreibung	Operator	Beschreibung
=	Gleich	<=	kleiner gleich	like	Stringvergleich
<	kleiner als	>=	größer gleich		
>	größer als	<>	ungleich		

Das Gleichheitszeichen hat in VBA zwei Funktionen. Zum einen kann mit = einer Variablen ein Wert zugewiesen werden, zum anderen können mit = zwei Werte (normalerweise

eine Variable und ein fester Wert oder zwei Variablen) verglichen werden. Im zweiten Fall bedeutet = „gleich“, so wie wir es aus unserem Sprachgebrauch kennen.

Ebenso können zwei logische Operatoren miteinander verknüpft werden.

Tabelle 12.6 Liste der logischen Operatoren

Operator	Excel-Name	Beschreibung
Not	NICHT	Negation
And	UND	Logisches Und
Or	ODER	Logisches Oder (das eine oder das andere)
Xor		Exklusives Oder (entweder genau das eine oder genau das andere)
Eqv		Logische Äquivalenz
Imp		Implikation
Is		Vergleich von Objektvariablen (für selbst erstellte Excel-Funktionen ohne Bedeutung)

Tabelle 12.7 Zusammenfassung der Datumsfunktionen

Funktionsname	Excel-Name	Bedeutung
DATE	HEUTE	Setzt das aktuelle Systemdatum ein oder stellt das Systemdatum um
NOW	JETZT	Gibt das Systemdatum und die aktuelle Systemzeit zurück
TIME		Setzt die aktuelle Systemzeit ein oder stellt die Systemzeit um
DATESERIAL	DATUM	Gibt die fortlaufende Datumszahl eines Datums zurück: DateSerial (Year, Month, Day)
DATEVALUE	DATWERT	Gibt das Datum eines String-Arguments zurück
TIMESERIAL	ZEIT	Gibt einen fortlaufenden Zeitwert für eine Uhrzeit zurück
TIMEVALUE	ZEITWERT	Wandelt einen String in eine Uhrzeit um
DATEADD		Addiert oder subtrahiert ein angegebenes Intervall zu einem oder von einem Datum Syntax: DateAdd(Intervall, Anzahl, Datum) Dabei wird das Intervall als String ausgegeben (vergleiche DateDiff)

Die Entwicklungsumgebung

Funktionsname	Excel-Name	Bedeutung
DATEDIFF		Gibt die Anzahl der Zeitintervalle zurück, die zwischen zwei Datumsangaben liegen Syntax: DateDiff(Intervall, Date1, Date2 [, FirstDayofWeek] [, FirstDayofYear])
DATEPART		Berechnet, zu welchem Teil eines angegebenen Intervalls ein Datum gehört: DatePart(Intervall, Date [, FirstDayofWeek] [, FirstDayofYear]) Die Zahlen und Variablen entsprechen denen von DateDiff
DAY	TAG	Filtert den Tag aus einem Datum
MONTH	MONAT	Filtert den Monat aus einem Datum
YEAR	JAHR	Filtert das Jahr aus einem Datum
WEEKDAY	WOCHENTAG	Gibt eine Zahl zwischen 1 und 7 zurück, die dem Wochentag entspricht: Weekday(Date, [FirstDayofWeek]) Dabei entsprechen Date einem Datum und FirstDayofWeek der gleichen Variable wie bei DateDiff. Der zurückgegebene Wert ist ebenso eine Zahl von 1 bis 7 oder von vbSunday bis vbSaturday
HOURL	STUNDE	Filtert die Stunde aus einer Uhrzeit
MINUTE	MINUTE	Filtert die Minutenanzahl aus einer Uhrzeit
SECOND	SEKUNDE	Filtert die Sekundenanzahl aus einer Uhrzeit

Die Funktion FORMAT ähnelt sehr der bedingten Formatierung in Excel:

Tabelle 12.8 Folgende Argumente sind bei „Format“ möglich:

Zeichen	Beschreibung	Beispiel
Kein Zeichen	Zeigt die Zahl ohne Formatierung an	
0	Beliebige Ziffer	Format(1234, "0") liefert 1234 Format(1234, "0.00") liefert 1234,00 Format(1234.5678, "0") liefert 1235 Format(1234.5678, "0.00") liefert 1234,57

Zeichen	Beschreibung	Beispiel
#	Platzhalter für eine Ziffer, die nur angezeigt wird, wenn sich an dieser Stelle eine Ziffer befindet. Gedacht für Tausendertrennzeichen	Format(1234, "0") liefert 1234 Format(1234, "#,##0") liefert 1.234 Format(123, "#,##0") liefert 123
. und ,	Der Punkt dient als Trennzeichen für Dezimalzeichen, das Komma für Tausendertrennzeichen. Also umgekehrt als im Deutschen!	Siehe oben
%	Multipliziert die Zahl mit 100 und fügt ein %-Zeichen an	Format(0.15, "#,##0.00%") liefert 15,00% Format(0.125, "0.00%") liefert 12,50%
E- E+ e- e+	Wissenschaftliche Zahlenschreibweise	Format(1250000, "0.00 E+00") liefert 1,25 E+06 Format(1250000, "0.00 E-00") liefert 1,25 E06 Format(1250000, "0.00 e+0") liefert 1,25 e+6 Format(0.125, "0.00 e-0") liefert 1,25 e-1
+, - und Leerzeichen	Können zur Darstellung direkt in die Formatierung eingefügt werden	Format(1234, "#,##0.00 ") liefert 1.234,00
currency	Liefert das voreingestellte Währungsformat	Format(1234, "currency") liefert 1.234,00 € (wenn so voreingestellt)
\	Das nächste Zeichen wird als Zeichen und nicht als Formatierung ausgegeben. Das "\" verschwindet in der Anzeige	Format(1234, "#,##0.00 \\E\\u\\r\\o") liefert 1.234,00 Euro

Es können bis zu vier verschiedene Zahlenformate in Abschnitten ausgegeben werden. Dabei bedeuten:

Zeichen	Beschreibung	Beispiel
Nur ein Abschnitt	Alle Werte	
Zwei Abschnitte	Der erste Abschnitt bezieht sich auf positive	Format(123, "0;(0)") liefert 123

Zeichen	Beschreibung	Beispiel
	Werte und die Null, der zweite auf negative	
Drei Abschnitte	Wie zwei Abschnitte; der dritte Abschnitt bezieht sich auf die Null	Format(-123, "0;(0);\N\i\x") liefert (123)
Vier Abschnitte	Wie drei Abschnitte; der vierte Abschnitt bezieht sich auf Null-Werte	Format(0, "0;(0);\N\i\x") liefert Nix

Tabelle 12.9 Datums- und Zeitformatierungen:

Zeichen	Beschreibung	Beispiel
/	Datumstrennzeichen	
d	Zeigt den Tag als Zahl	Format("2.9.2018", "d/m/yy") liefert 2.9.18
dd	Zeigt den Tag als Zahl, wobei Tage zwischen 1 und 9 mit führender Null dargestellt werden	Format("2.9.2018", "dd/mm/yyyy") liefert 02.09.2018
ddd	Zeigt den Wochentag als Abkürzung	Format("2.9.2018", "ddd") liefert So
dddd	Zeigt den Wochentag ausgeschrieben	Format("2 / 9 / 2018", "dddd, \d\e\n dd. mmmm yyyy") liefert Sonntag, den 02. September 2018
dddddd		
short date	Zeigt das vollständige Datum gemäß der Systemsteuerung im Kurzformat an (dd.mm.yyyy)	Format("2 / 9 / 2018", "dddddd") und Format("2 / 9 / 2018", "short date") liefern 02.09.2018
dddddd		
long date	Zeigt das vollständige Datum gemäß der Systemsteuerung im Langformat an (dd.mmmm.yyyy)	Format("2 / 9 / 2018", "dddddd") und Format("2 / 9 / 2018", "long date") liefern Sonntag, 02 September 2018
w	Der Wochentag als Zahl (1 = Sonntag, 7 = Samstag)	Format("2.9.2018", "w") liefert 1

Zeichen	Beschreibung	Beispiel
ww	Die Kalenderwoche (1 – 54)	Format("2.9.2018", "ww") liefert 35
m	Monat als Zahl (1 – 12)	
mm	Monat als Zahl. Monate zwischen 1 und 9 mit führender Null	
mmm	Monat als Text mit drei Buchstaben (Jan – Dez)	
mmmm	Vollständiger Monatsname	
q	Quartal	Format("2.9.2018", "q") liefert 3
yy	Jahr als zweistellige Zahl (00 – 99)	
yyyy	Jahr als vierstellige Zahl (100 – 9999)	

Tabelle 12.10 Die Zeitangaben

Zeichen	Beschreibung	Beispiel
:	Zeittrennzeichen	
h	Stunde als Zahl ohne führende Null	Format("2:4", "h:m") liefert 2:4
hh	Stunde als Zahl mit führender Null	Format("2:4", "hh:mm") liefert 02:04
n oder m	Die Minute ohne führende Null	Format("15:4", "hh:mm") liefert 15:04
short time	Liefert die Kurzform gemäß Systemsteuerung	Format("15:4", "short time") liefert 15:04
nn oder mm	Die Minute mit führender Null	
s	Die Sekunden ohne führende Null	
ss	Die Sekunden mit führender Null	
tttt	Die vollständige Zeitangabe	zahl = Format("2:4", "tttt") liefert 02:04:00

Zeichen	Beschreibung	Beispiel
long time	Liefert die Langform gemäß Systemsteuerung	Format("15:4", "long time") liefert 15:04:00
AM/PM, am/pm, A/P, A/p, AMPM	Verschiedene 12-Stunden-Formate	

Tabelle 12.11 Zeichenformatierungen

Zeichen	Beschreibung	Beispiel
@	Platzhalter für ein Zeichen. Ist die Variable leer, wird ein leerer String ausgegeben.	Format("Sonnenschein", "@") liefert "Sonnenschein"
&	Platzhalter für ein Zeichen. Ist die Variable leer, wird nichts ausgegeben.	Format("Sonnenschein", "&") liefert "Sonnenschein"
!	Auffüllen aller Platzhalter von links nach rechts	Format("Sonnenschein", "!") liefert "Sonnenschein"
<	Zeigt den Text in Kleinbuchstaben	Format("Sonnenschein", "<") liefert "sonnenschein"
>	Zeigt den Text in Großbuchstaben	Format("Sonnenschein", ">") liefert "SONNENSCHNEN"

12.1.3Verzweigungen

Wenn-Verzweigungen nehmen auch in VBA eine zentrale Rolle ein. Zur Logik ist wenig hinzuzufügen, nur noch zur Syntax. Sie kann lauten:

```
If Bedingung Then Anweisung1
```

Der Übersichtlichkeit halber wird dieser Befehl normalerweise untereinander geschrieben und eingerückt. Beachten Sie aber, dass dann die If-Verzweigung mit einem End If geschlossen werden muss. Und vergessen Sie das Then nicht! Die Syntax sieht so aus:

```
If Bedingung Then
    Anweisung1
    [Anweisung2]
    [Anweisung3]
```

```
[Else Anweisung2]
[Anweisung1]
[Anweisung2]
[Anweisung3]
End If
```

Beispiel: Der Benutzer gibt die beiden Variablen für a und b ein, mit denen die Gleichung

$$0 = x^2 + a \cdot x + b$$

berechnet werden soll. Die Lösung der Gleichung lautet:

$$x_{1,2} = -\left(\frac{a}{2}\right) \pm \sqrt{\left(\frac{a}{2}\right)^2 - b}$$

Also wird überprüft, ob

$$\left(\frac{a}{2}\right)^2 - b \geq 0$$

Sie funktioniert nur, wenn die Diskriminante

$$\left(\frac{a}{2}\right)^2 - b < 0$$

ist. Wenn dies der Fall ist, so existiert keine Lösung. Sollte jedoch

$$\left(\frac{a}{2}\right)^2 - b = 0$$

sein, so existiert genau eine Lösung, nämlich: $-\frac{a}{2}$. Andernfalls gibt es die beiden oben angegebenen Lösungen. Der Benutzer wird aufgefordert, die beiden Variablen für a und b auszurechnen. Dann wird überprüft, ob es keine Lösung gibt. Danach wird getestet, ob es nur eine Lösung gibt oder zwei. Diese werden angezeigt.

```
Function Quadratische_Gleichung_L1 _
(a As Double, b As Double) As Double
```

Die Entwicklungsumgebung

```
Dim dblD As Double
dblD = (a / 2) ^ 2 - b
If dblD < 0 Then
    Quadratische_Gleichung_L1 = "#Keine Lösung"
Else
    Quadratische_Gleichung_L1 = -(a / 2) + Sqr(dblD)
End If
End Function
```

```
Function Quadratische_Gleichung_L2 _
(a As Double, b As Double) As Double
Dim dblD As Double
dblD = (a / 2) ^ 2 - b
If dblD < 0 Then
    Quadratische_Gleichung_L2 = "#Keine Lösung"
Else
    Quadratische_Gleichung_L2 = -(a / 2) - Sqr(dblD)
End If
End Function
```

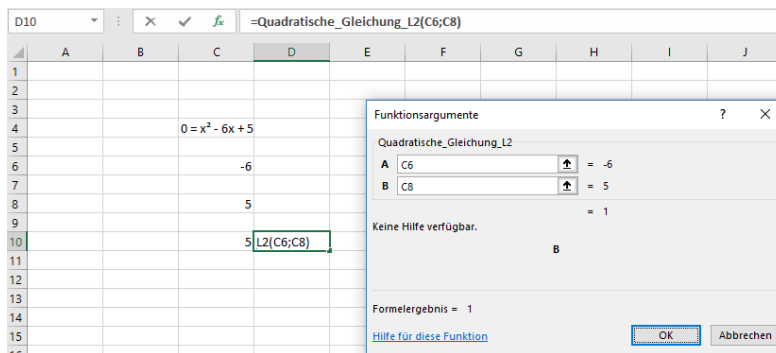


Abbildung 12.5 Zwei Eingaben und zwei Lösungen

12.1.4 Schleifen

Es existieren mehrere Schleifentypen, die einige fundamentale Unterschiede aufweisen.

In der For...Next-Schleife wird eine Variable definiert. Im zweiten Schritt wird diese Variable weitergezählt (bezeichnenderweise wird die Variable oft Zähler oder i genannt). Sie wird so lange vergrößert, bis ein vorgegebener Endwert erreicht ist. Ist der Endwert erreicht, verlässt das Programm die Schleife und arbeitet die Befehle ab, die nach, das heißt hinter oder außerhalb, der For...Next-Schleife stehen. Sie hat folgende Syntax:

```
For Zähler = Anfangswert To Endwert [Step Schrittweite]
    Anweisungen
Next [Zähler]
```

Der Befehl Zähler = Anfangswert bedeutet, dass der Variablen Zähler ein Wert (zum Beispiel 1) zugewiesen wird. Dieser wird nun vergrößert – entweder um 1 oder um die Zahl, die hinter Step steht.

Angenommen die Funktion soll nicht in Einerschritten hochzählen, sondern in einer anderen Schrittweise, so kann hinter der Zeile

```
For AdamRiese = 49 To 63
```

ein Step mit einer Zahl folgen. Zum Beispiel

```
For AdamRiese = 49 To 63 Step 3
```

Sollte der Endwert überschritten werden, so wird die Schleife automatisch beendet. Auch ein Abwärtszählen ist möglich:

```
For AdamRiese = 63 To 49 Step -3
```

Und sogar ein schrittweises Weiterzählen mit Dezimalzahl. (Achtung: VBA verlangt die US-amerikanische Dezimalschreibweise!)

```
For AdamRiese = 49 To 63 Step 2.5
```

Die For...Next-Schleife kann folglich gar nicht, einmal oder beliebig oft ausgeführt werden (theoretisch und leider auch praktisch sogar unendlich oft).

Beispiel: Der Benutzer gibt eine Zahl ein, von der überprüft wird, ob es sich um eine Primzahl handelt oder nicht:

```
Function prim(Zahl As Double) As Boolean
    Dim dblZähler As Double
```

```
If Zahl < 1 Then
    prim = "Fehler"
    Exit Function
ElseIf Int(Zahl) <> Zahl Then
    prim = "Fehler"
    Exit Function
End If

For dblZähler = 2 To Sqr(Zahl) + 1
    If Zahl Mod dblZähler = 0 Then
        prim = False
        Exit Function
    End If
Next dblZähler

prim = True
End Function
```

Alle Zahlen zwischen 2 und der Wurzel der eingegebenen Zahl werden überprüft, ob sie Teiler der eingegebenen Zahl sind. Wenn sich ein Teiler findet, dann ist die Zahl keine Primzahl – die Berechnung kann beendet werden. Wurden alle Zahlen überprüft und kein Teiler gefunden, so ist die Zahl eine Primzahl.

Hinweis

Die Funktion hat einen kleinen Haken. Zwar könnte jede Zahl bis $1,797693 \cdot 10^{324}$ getestet werden, aber die VBA-Funktion `Mod` kann nur Long-Werte verarbeiten. Um den Wertebereich zu vergrößern, müsste die Zeile, in der geprüft wird, korrigiert werden:

```
...
    If Zahl /dblZähler = Int(Zahl /dblZähler) Then
...

```

Nun setzt nur noch Excel die Obergrenze mit 999.999.999.999.999 (siehe Kapitel 1).

	C2																	
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O			
1	117	119	121	123	125	127	129	131	133	135	137	139	141	143	145			
2	FALSCH	FALSCH	FALSCH	FALSCH	FALSCH	WAHR	FALSCH	WAHR	FALSCH	FALSCH	WAHR	WAHR	FALSCH	FALSCH	FALSCH			

Abbildung 12.6 Der Primzahltest

Oder Sie verwenden eine andere Schleifenform, die für diese Zwecke besser geeignet ist – die Bedingungschleife:

```
Do
    Anweisungen
[Exit Do]
Loop Until
```

Mit `Do...Loop Until` wird eine Bedingung überprüft. Ist sie korrekt, wird weitergearbeitet, falls nicht, wird die Schleife verlassen. Im Primzahlenbeispiel sieht das folgendermaßen aus:

```
Function prim2(Zahl As Double) As Boolean
    Dim dblZähler As Double
```

```
    If Zahl < 1 Then
        prim2 = "Fehler"
        Exit Function
    ElseIf Int(Zahl) <> Zahl Then
        prim2 = "Fehler"
        Exit Function
    End If
```

```
    dblZähler = 2
    Do
        If Zahl Mod dblZähler = 0 Then
            prim2 = False
            Exit Function
        End If
```

Benutzerdefinierte Funktionen

```
dblZähler = dblZähler + 1  
Loop Until dblZähler >= Sqr(Zahl) + 1
```

```
prim2 = True  
End Function
```

Die Do-Schleife wird betreten, solange der Zähler, der zuvor auf 2 gesetzt wurde, kleiner ist als die Wurzel aus dem eingegebenen Wert. Innerhalb der Schleife wird überprüft, ob der eingegebene Wert einen Teiler hat. Wenn ja, dann ist die Zahl keine Primzahl. Wenn nein, so wird der Zähler um 1 erhöht, und die Schleife beginnt von vorne mit dem Zählen. Ähnlich funktionieren die Schleifen

```
Do  
[...]  
[Exit Loop]  
Loop While
```

Der Unterschied liegt in der Überprüfung: Das Loop While weist bleibt solange in der Schleife, wie die Bedingung noch gilt. Loop Until verlässt die Schleife dagegen, wenn die Bedingung erfüllt ist.

Man kann die Bedingungen auch in den Kopf der Schleife schreiben – dann könnte die Schleife möglicherweise gar nicht ausgeführt werden. Steht die Bedingung im Fuß, so wird sie immer mindestens einmal ausgeführt.

Achtung vor Endlosschleifen! Dann bleibt nur noch der „Ausstieg“ aus dem Makro mit der Tastenkombination [Strg] + [Unterbr] ...

12.2 Benutzerdefinierte Funktionen

Folgende Funktionsklassen stehen in Excel zur Verfügung:

12.2.1 Funktionen ohne Parameter

Funktionen, die eine Konstante zurückgeben, da sie keinen Wert in den runden Klammern haben: in Excel PI(), HEUTE(), JETZT(). Sie sind einfach nachzubilden, wie beispielsweise folgende Funktionen:


```
Function E() As Double
    E = Exp(1)
End Function
```

```
Function WURZEL2() As Double
    WURZEL2 = Sqr(2)
End Function
```

```
Function WURZEL05() As Double
    WURZEL05 = Sqr(0.5)
End Function
```

Oder auch folgende Konstante, mit welcher der goldene Schnitt berechnet werden kann:

```
Function GOLDENERSCHNITT() As Double
    GOLDENERSCHNITT = 0.5 * Sqr(5) + 0.5
End Function
```

Gerade im Bereich Naturwissenschaften lässt sich eine Vielzahl an Konstanten finden, die als benutzerdefinierte Funktionen gespeichert werden können.

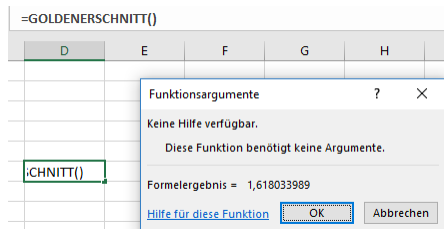


Abbildung 12.7 Die Funktion GOLDENERSCHNITT () $0,5 + 0,5 \times \sqrt{5}$.

Aber auch Excelinterna können berechnet werden, wie beispielsweise den Namen des Autor der Datei, der in den Eigenschaften gespeichert wurde:

```
Function Autor() As String
    Autor = ActiveWorkbook. _
        BuiltinDocumentProperties("Author")
End Function
```

Oder man liest den Namen des Anwenders vom System aus – also den Namen, unter dem sich der Benutzer angemeldet hat:

```
Function Benutzer() As String
    Benutzer = Environ("username")
End Function
```

Die folgende Funktion liefert den Namen.

```
Function Blattname() As String
    Blattname = ActiveSheet.Name
End Function
```

So erhält man den Namen des Blattes, auf dem sich diese Funktion befindet. Die Excel-funktion BLATT liefert leider nur die Nummer, aber nicht den Namen.

12.2.2 Funktionen, die ein Argument verarbeiten

In Excel sind dies Funktionen wie: WURZEL(Zahl), COS(Zahl), MONAT(Datum), LÄNGE(Text); hier finden Sie einige Funktionen:

```
Function Ostern(Jahreszahl As Integer) As Date
    a = Jahreszahl - 1900
    b = a Mod 19
    c = (7 * b + 1) \ 19
    d = (11 * b + 4 - c) Mod 29
    e = a \ 4
    f = (a + e + 31 - d) Mod 7
    Ostern = DateSerial(Jahreszahl, 4, 25 - d - f)

End Function
```

Oder zwei andere Funktionen:

```
Function Quersumme(Zahl As Long) As Long
    Dim intZähler As Integer
    Dim lngQ As Long
```

```
lngQ = 0

For intZähler = 1 To Len(CStr(Zahl))
    lngQ = lngQ + Mid(CStr(Zahl), intZähler, 1)
Next
Quersumme = lngQ
End Function
```

```
Function Endquersumme(Zahl As Long) As Long
    Dim intZähler As Integer
    Dim lngQ As Long
    lngQ = 0

    For intZähler = 1 To Len(CStr(zahl))
        lngQ = lngQ + Mid(CStr(zahl), intZähler, 1)
    Next
    Do While lngQ >= 10
        lngQ = Quersumme(lngQ)
    Loop
    Endquersumme = lngQ
End Function
```

Oder eine Funktion, die eine Note (also eine Zahl) in einen Text umwandelt:

```
Function Notentext(Note As Byte) As String
    Select Case Note
        Case 1
            Notentext = "sehr gut"
        Case 2
            Notentext = "gut"
        Case 3
```

```
        Notentext = "befriedigend"
    Case 4
        Notentext = "ausreichend"
    Case 5
        Notentext = "mangelhaft"
    Case 6
        Notentext = "ungenügend"
    Case Else
        Notentext = "undefiniert"
End Select
End Function
```

Zu dieser Kategorie gehört auch die bereits beschriebene Funktion `prim`. Diese Funktionen greifen auf eine Zahl, einen Text oder ein Datum zurück und verarbeiten ihn mit Hilfe der Grundrechenarten oder VBA-interner Funktionen.

Da Excel keine Funktion `Quartal` besitzt, kann diese nachgebaut werden:

```
Function QUARTAL(DatAngabe)
    QUARTAL = DatePart("q", DatAngabe)
End Function
```

Dabei können Funktionen andere Funktionen verwenden und auch rekursiv rechnen (`ENDQUERSUMME`). Oder auch Informationen von Excel zurückgeben – beispielsweise das Zahlenformat:

```
Function Zellenformat(Zelle As Range) As String
    Zellenformat = Zelle.NumberFormatLocal
End Function
```

Oder auch eine Funktion, die den englischen Funktionsnamen einer ausgewählten Funktion liefert:

```
Function ENGLISCH(Zelle As Range) As String
    ENGLISCH = Zelle.Formula
End Function
```

12.2.3 Funktionen, die mehrere Argumente verarbeiten, die alle eingegeben werden müssen

In Excel zählen dazu: DATUM(Jahr; Monat; Tag), RMZ(Zins; Zzr; Bw), ZÄHLENWENN(Bereich; Suchkriterien), SUCHEN(Suchtext; Text). Zwei Beispiele, die in Excel fehlen:

```
Function Suchen_Von_Rechts(Suchtext As String, _
    Text As String) As Integer
    Dim intZähler As Integer
    For intZähler = Len(Text) To 1 Step -1
        If InStr(intZähler, LCase(Text), _
            LCase(Suchtext)) > 0 Then
            Suchen_Von_Rechts = _
                Len(Text) - intZähler + 1
            Exit Function
        End If
    Next
    Suchen_Von_Rechts = 0
End Function
```

oder die Funktion Finden_Von_Rechts, bei der zwischen Groß- und Kleinschreibung unterschieden wird:

```
Function Finden_Von_Rechts(Suchtext, Text)
    Dim intZähler As Integer
    For intZähler = Len(Text) To 1 Step -1
        If InStr(intZähler, LCase(Text), _
            LCase(Suchtext)) > 0 Then
            Finden_Von_Rechts = Len(Text) - intZähler + 1
            Exit Function
        End If
    Next
```

Benutzerdefinierte Funktionen

```
Finden_Von_Rechts = 0
```

```
End Function
```

Beispiel: Der Benutzer gibt die Seitenlängen der Katheten eines rechtwinkligen Dreiecks ein und erhält die Länge der Hypothenuse berechnet. Angenommen diese Berechnung wird häufig verwendet, dann kann das Problem folgendermaßen gelöst werden:

```
Function RechtwinklDreieck(Seite1 As Double, Seite2 As Double) As Double
```

```
    RechtwinklDreieck = Sqr(Seite1 ^ 2 + Seite2 ^ 2)
```

```
End Function
```

Es lassen sich noch viele Beispiele finden, in denen mehrere Parameter benötigt werden: finanzmathematische Berechnungen (beispielsweise zur Rentenversicherung), Berechnungen aus naturwissenschaftlichen Diskursen oder auch einfache geometrische Berechnungen. So kann beispielsweise aus den drei Seiten eines Dreiecks der Flächeninhalt bestimmt werden oder das Volumen eines Kegelstumpfes aus der Höhe und den beiden Radien:

```
Function DREIECK_FLÄCHE _  
    (SeiteA As Double, SeiteB As Double, _  
    SeiteC As Double) As Double  
Dim dblALPHA As Double  
dblALPHA = Application. _  
    WorksheetFunction.Acos((SeiteB ^ 2 + _  
    SeiteC ^ 2 - SeiteA ^ 2) / (2 * SeiteB * SeiteC))  
DREIECK_FLÄCHE = 0.5 * SeiteB * SeiteC * Sin(dblALPHA)  
End Function
```

Hierbei wird in VBA die Excel-Funktion ACOS verwendet, die in VBA nicht direkt zur Verfügung steht. Alle Excel-Funktionen können über das Objekt

```
Application.WorksheetFunction
```

benutzt werden. Oder die bereits genannten Funktionen Quadratische_Gleichung_L1, Quadratische_Gleichung_L2. Oder auch der BMI (Body-Mass-Index), der berechnet werden kann über:

```
BMI = Gewicht / (Größe_in_cm * Größe_in_cm / 10000)
```

Am Ende dieses Kapitels wird gezeigt, wie man fehlende Funktionen (aus dem Bereich Geometrie) in Excel nachbilden kann.

12.2.4 Funktionen mit mehreren Parametern, die nicht alle eingegeben werden müssen

Angenommen ein Argument ist optional, dann kann es mit Optional eingefügt werden. Aus Excel sind solche bekannt, wie: RMZ(Zins, Zzr, Bw, [Zw], [F]), SVERWEIS(Suchkriterium, Matrix, Spaltenindex, [Bereich_Verweis]), LINKS(Text, [Anzahl_Zeichen]) oder auch SPALTE([Bezug])

Im folgenden Beispiel kann optional ein „u“ (umgangssprachlich) für den Index eingegeben werden:

```
Function Zahlentext(Zahl As Double, _
    Optional Index As String) As String
    Dim intZähler As Integer
    Dim strZiffer As String
    Dim strZiffertext As String
    Dim strGanzeZahl As String
    For intZähler = 1 To Len(CStr(Zahl))
        strZiffer = Mid(CStr(Zahl), intZähler, 1)
        If strZiffer = "0" Then
            strZiffertext = "null"
        ElseIf strZiffer = "1" Then
            strZiffertext = "eins"
        ElseIf strZiffer = "2" Then
            If LCase(Index) = "u" Then
                strZiffertext = "zwo"
            Else
                strZiffertext = "zwei"
            End If
        ElseIf strZiffer = "3" Then
            strZiffertext = "drei"
```

```
[...]
    ElseIf strZiffer = "9" Then
        strZiffertext = "neun"
    ElseIf strZiffer = "," Then
        strZiffertext = "Komma"
    Else
        strZiffertext = Mid(CStr(Zahl), intZähler, 1)
    End If
    strGanzeZahl = strGanzeZahl & " " & strZiffertext
Next intZähler
Zahlentext = Trim(strGanzeZahl)
End Function
```

Gibt der Benutzer ein „u“ ein, so wird er in der If-Verzweigung verarbeitet, gibt er etwas anderes oder nichts ein, so folgt die Alternative.

Hinweis

Beachten Sie, dass sämtliche optionalen Argumente am Ende der Parameterliste stehen müssen! Zuerst werden die notwendigen Parameter aufgelistet, dann die „freien“.

Mit diesem Wissen könnte man die Funktion `Blattname` aus Abschnitt 12.2.1 erweitern:

```
Function Blattname(Optional Nummer As Integer) As String
    If Nummer = 0 Then
        Blattname = ActiveSheet.Name
    Else
        Blattname = ActiveWorkbook.Sheets(Nummer).Name
    End If
End Function
```

Hinweis

Leider werden die optionalen Parameter im Funktionsassistenten nicht als solche (nicht fette Schrift) gekennzeichnet. Auch bei der Eingabe in einer Zelle kann man nicht erkennen, ob es sich um einen optionalen Parameter handelt.

12.2.5 Funktionen, die mehrere Zellen oder Bereiche verarbeiten können

In Excel SUMME(Zahl1, Zahl2, ...), VERKETTEN(Text1, Text2, ...), ODER(Wahrheitswert1, Wahrheitswert2, ...). Aus einem Bereich sollen beispielsweise alle Zahlen addiert werden, allerdings ohne Vorzeichen, das heißt: nur deren Beträge. In den Anweisungen ist also zu überprüfen, ob der Zellinhalt positiv oder negativ ist. Ist r2 negativ, dann wird r2 abgezogen – ansonsten addiert:

```
Function SummeAbs (ParamArray Bereich())
    Dim xlMarkierung
    Dim xlBereich As Range
    Dim xlZelle As Range
    Dim dblZ As Double

    For Each xlMarkierung In Bereich()
        For Each xlBereich In xlMarkierung.Areas
            For Each xlZelle In xlBereich.Cells
                If xlZelle.Value > 0 Then
                    dblZ = dblZ + xlZelle.Value
                Else
                    dblZ = dblZ - xlZelle.Value
                End If
            Next
        Next
    Next

    SummeAbs = dblZ
End Function
```

So könnte man folgendes Problem lösen: Eine Funktion soll bestimmte Zahlen summieren, die auf eine bestimmte Art formatiert sind, beispielsweise fett. Dies kann per VBA leicht überprüft werden:

```
Function Summe_Fett (ParamArray Bereich()) As Double
    Dim xlMarkierung
```

```
Dim xlBereich As Range
Dim xlZelle As Range
Dim dblS As Double
Application.Volatile True
dblS = 0

For Each xlMarkierung In Bereich()
    For Each xlBereich In xlMarkierung.Areas
        For Each xlZelle In xlBereich.Cells
            If IsNumeric(xlZelle.Value) = True Then
                If xlZelle.Font.Bold = True Then
                    dblS = dblS + xlZelle.Value
                End If
            End If
        Next xlZelle
    Next xlBereich
Next xlMarkierung

Summe_Fett = dblS
End Function
```

12.2.6 Funktionen, die mehrere Werte zurückgeben

In Excel die Matrixfunktionen TREND(Y_Werte, X_Werte, Neue_x_Werte), MTRANS(Matrix), hier: PRIMFAKTOREN(Zahl) und TEILER(Zahl). Werden zu wenige Zellen markiert, dann finden sich nur einige der Ergebnisse wieder, werden zu viele Zellen markiert, so stehen in den überflüssigen Zellen Fehlermeldungen (#NV).

So könnte die Funktion ARRWOCHENTAGE die Wochentage anzeigen:

```
Function ARRWOCHENTAGE() As Variant
    ARRWOCHENTAGE = Array("Montag", "Dienstag", "Mittwoch", _
        "Donnerstag", "Freitag", "Samstag", "Sonntag")
End Function
```

End Function

Will man die Anzahl der Ergebniswerte ermitteln, ist eine zweite Funktion nötig, welche die korrekte Anzahl der Ergebnisse ermittelt (PrimfaktorAnzahl und AnzahlDerTeiler). Der korrekte Rückgabewert einer Matrixfunktion ist As Variant.

```
Function PrimfaktorAnzahl(Zahl As Double) As Double
    ' Diese Funktion liefert die Anzahl der Faktoren
    ' Sie wird für die nächste Funktion benutzt.
    Dim dblAnzahl As Double
    Dim dblZähler As Double

    dblZähler = 2
    Do
        If Zahl Mod dblZähler = 0 Then
            dblAnzahl = dblAnzahl + 1
            Zahl = Zahl / dblZähler
        Else
            dblZähler = dblZähler + 1
        End If

    Loop Until dblZähler > Zahl
    PrimfaktorAnzahl = dblAnzahl
End Function
```

```
Function Primfaktoren(Zahl As Double)
    Dim dblWerte() As Double
    Dim intZähler As Integer
    Dim intPKZähler As Integer
    Dim dblPFZahl As Double
    Dim dblÜZahl As Double
```

Benutzerdefinierte Funktionen

```
dblÜZahl = Zahl
dblPFZahl = PrimfaktorAnzahl(dblÜZahl)

ReDim dblWerte(dblPFZahl - 1, 0)

intZähler = 2
Do
    If Zahl Mod intZähler = 0 Then
        dblWerte(intPKZähler, 0) = intZähler
        Zahl = Zahl / intZähler
        intPKZähler = intPKZähler + 1
    Else
        intZähler = intZähler + 1
    End If

Loop Until intZähler > Zahl

Primfaktoren = dblWerte()
End Function

Function AnzahlDerTeiler(Zahl As Double) As Double
    Dim lngZähler As Long
    Dim intAnzahl As Integer
    Dim dblWurzel As Double
    ' diese Funktion wird für die Matrix-
    ' funktion TEILER benötigt.

    intAnzahl = 0
    dblWurzel = Sqr(Zahl)
    For lngZähler = 1 To Int(dblWurzel)
```

```
        If Zahl Mod lngZähler = 0 Then
            intAnzahl = intAnzahl + 1
        End If
    Next lngZähler

    intAnzahl = intAnzahl * 2
    If dblWurzel = Int(dblWurzel) Then
        intAnzahl = intAnzahl - 1
    End If
    AnzahlDerTeiler = intAnzahl
End Function

Function Teiler(Zahl As Long)
    Dim lngWerte() As Long
    Dim intZähler As Integer
    Dim intTeiler As Integer

    ReDim lngWerte(AnzahlDerTeiler(Zahl) - 1, 0)

    intTeiler = 0
    For intZähler = 1 To Zahl
        If Zahl Mod intZähler = 0 Then
            lngWerte(intTeiler, 0) = intZähler
            intTeiler = intTeiler + 1
        End If
    Next intZähler

    Teiler = lngWerte()
End Function
```

Hinweis

Achtung beim Testen: Vergessen Sie nicht, dass Matrixfunktionen mit [Shift]+[Strg]+[Enter] beendet werden.

{=Teiler(F1)}		
D	E	F
	Zahl:	664547
	Anzahl der Teiler:	16
	Teiler:	1
		13
		17
		31
		97
		221
		403
		527
		1261
		1649
		3007
		6851
		21437
		39091
		51119
		664547

Abbildung 12.8 Die Anzahl der Teiler und die Teiler (mehrere Werte werden zurückgegeben)

12.2.7Volatile

Die Funktionen:

- BEREICH.VERSCHIEBEN
- HEUTE
- INDIREKT
- INFO
- JETZT
- ZELLE("Dateiname")
- ZUFALLSBEREICH
- ZUFALLSZAHL

Sind volatil, das heißt: bei jeder Aktion innerhalb von Excel werden sie neu berechnet. Benutzerdefinierte Funktionen dagegen sind nicht volatil. Das bedeutet: die Funktion

```
Function Blattname() As String
```

```

    Blattname = ActiveSheet.Name
End Function

```

Zeigt nach der Eingabe den korrekten Funktionsnamen an, nach Änderung des Blattname jedoch nicht mehr. Die Funktion liefert noch den alten Tabellenblattnamen – sie wird erst nach Editieren der Zelle ([F2]) neu berechnet. Die kann mit Hilfe der Methode `Volatile` verbessert werden:

```

Function TABELLENNAMEVOLATILE() As String
    Application.Volatile True
    TABELLENNAMEVOLATILE = ActiveSheet.Name
End Function

```

Dies könnte auch beispielsweise mit einem optionalen Parameter eingesetzt werden:

```

Function EinZiehen(InBereich As Range, _
    Optional Neuberechnen = 0)
    ' Diese Funktion wählt zufällig eine Zelle
    ' des Bereichs aus.
    ' Wenn Neuberechnen = 1 ist wird die
    ' Funktion als veränderlich gekennzeichnet,
    ' so dass das Arbeitsblatt neu berechnet wird
    If Neuberechnen <> 0 Then Application.Volatile
    ' Eine zufällige Zelle bestimmen
    EinZiehen = InBereich(Int((InBereich.Count) * Rnd + 1))
End Function

```

12.3 Benutzerfehler in der Eingabe

Gibt der Benutzer nun einen fehlerhaften Wert ein (beispielsweise eine Zeichenkette statt einer Zahl oder mehrere Zellen statt einer einzigen), so wird der entsprechende Excel-interne Fehler aktiviert: `#WERT`, `#NAME`, `#DIV/0`, und so weiter. Wollen Sie dagegen explizit Fehlermeldungen bei bestimmten Eingaben erzeugen, so können Sie die VBA-Funktion `CVErr` verwenden. Sie benötigt eine Fehlernummer. Das folgende Beispiel erzeugt alle möglichen Fehlernummern (die „Umkehrfunktion“ von `FEHLER.TYP`):

Die Anzeige im Funktionsassistenten

```
Public Function XLFehler(Index As Byte)
    If Index = 1 Then
        XLFehler = CVErr(2000) ' #NULL
    ElseIf Index = 2 Then
        XLFehler = CVErr(2007) ' #DIV/0
    ElseIf Index = 3 Then
        XLFehler = CVErr(2015) ' #WERT
    ElseIf Index = 4 Then
        XLFehler = CVErr(2023) ' #BEZUG
    ElseIf Index = 5 Then
        XLFehler = CVErr(2029) ' #NAME
    ElseIf Index = 6 Then
        XLFehler = CVErr(2036) ' #ZAHL
    ElseIf Index = 7 Then
        XLFehler = CVErr(2042) ' #NV
    End If
End Function
```

Achtung

eine benutzerdefinierte Funktion darf nicht „Fehler“ genannt werden! Möglicherweise kollidiert der Name mit der Excel-Funktion FEHLER.TYP.

Hinweis

Soll eine Funktion nicht im Funktionsassistenten angezeigt werden, dann muss sie private deklariert werden.

12.4 Die Anzeige im Funktionsassistenten

Soll eine Funktion in Excel nicht im Funktionsassistenten angezeigt werden, dann muss sie `Private` deklariert werden. Jede Funktion, die mit `Function` beginnt (oder explizit mit `Public Function`), erscheint im Funktionsassistenten.

Um einen Kommentar anzuzeigen oder die Funktion in einer anderen Kategorie auftauchen zu lassen, muss der Befehl `MacroOptions` aktiviert werden. Beispielsweise so:


```

Sub Function_Verschieben()
    Application.MacroOptions _
    Macro:="Ostern", _
    Description:="Diese Funktion berechnet den Ostersonntag"
    & " nach der Formel von Gauß", _
    Category:=2
End Sub

```

Dabei ergibt sich die Nummer nach der Reihenfolge der Kategorien im Funktionsassistenten:

Tabelle 12.12 Liste der Funktionskategorien für die Methode MacroOptions

Wert	Kategorie
1	Finanzmathematik
2	Datum & Uhrzeit
3	Mathematik & Trigonometrie
4	Statistik
5	Matrix
6	Datenbank
7	Text
8	Logische Funktionen
9	Information
10	Commands
11	Customizing
12	Macro Control
13	DDE/External
14	Benutzerdefinierte Funktionen
15 – 32	Eigene Kategorien

Sie könne also auch eine eigene Kategorie definieren:

```

Sub Function_Verschieben()
    Application.MacroOptions _

```

Die Anzeige im Funktionsassistenten

```
Macro:="RechtwinklDreieck", _  
Description:=_  
"Diese Funktion berechnet die Hypotenuse " & _  
"eines rechtwinkligen Dreiecks " & _  
"nach dem Satz von Pythagoras", _  
Category:="Geometrie"  
  
End Sub
```

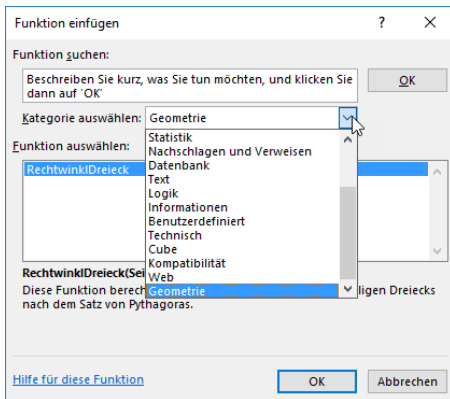


Abbildung 12.9 Eine neue Kategorie wird für neue Funktionen generiert.

Zu den einzelnen parametern können mit Hilfe eines Arrays ebenso kleine Hilfetexte generiert werden:

```
Sub EinordnenUndHilfstexte()  
    Dim arrHilfe(1 To 2) As String  
    arrHilfe(1) = "Die Breite des Rechtecks"  
    arrHilfe(2) = "Die Höhe des Rechtecks"  
    Application.MacroOptions Macro:="dblflaecherechteck", _  
        Description:="Berechnet die Fläche eines Rechtecks", _  
        Category:="Geometrie", ArgumentDescriptions:=arrHilfe  
End Sub
```

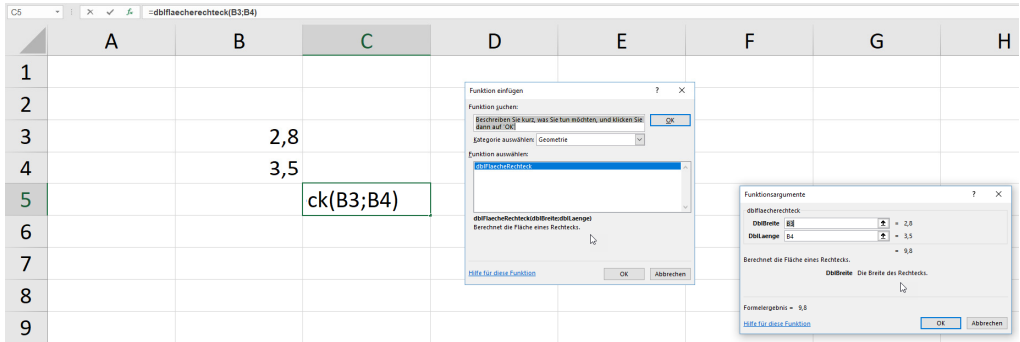


Abbildung 12.10 Die Erklärungstexte der Parameter.

Hinweis

Leider werden weder die Parameternamen noch die Hilfetexte angezeigt, wenn die Funktion in eine Zelle eingetragen und nicht über den Funktionsassistenten ausgewählt wird.

12.5 Speichern der VBA-Funktionen

Achtung

Benutzerdefinierte Funktionen können nur dann verwendet werden, wenn die Arbeitsmappe, in der sie gespeichert sind, offen ist. Soll die Funktion für alle Excel-Mappen zur Verfügung stehen, so kann die Datei als Add-In gespeichert werden (*.xla oder *.xlam).

Im Explorer können Eigenschaften der Datei festgelegt werden, die im Dialog als Alias-Name und als Kommentar angezeigt werden.

Nun stehen diese Funktionen allen Programmen zur Verfügung, wenn im Add-In-Manager das entsprechende Add-In aktiviert ist. Liegt das Add-In in einem anderen Ordner als die übrigen Add-Ins, dann muss mit dem Schalter „Durchsuchen“ der richtige Ordner gewählt werden.

Achtung

Wenn Sie die Excel-Datei als Add-In speichern, so haben Sie die ursprüngliche Datei nicht mehr vollständig zur Verfügung! Diese sollten Sie sicherheitshalber auch als Excel-Mappe speichern.

12.6 Zusammenfassung

So gehen Sie vor, wenn Sie eine oder mehrere Funktionen erstellen möchten, die Sie dauerhaft verwenden wollen:

- Öffnen Sie eine leere Mappe, wechseln in den Visual-Basic-Editor und erstellen in einem Modul die Funktion(en).

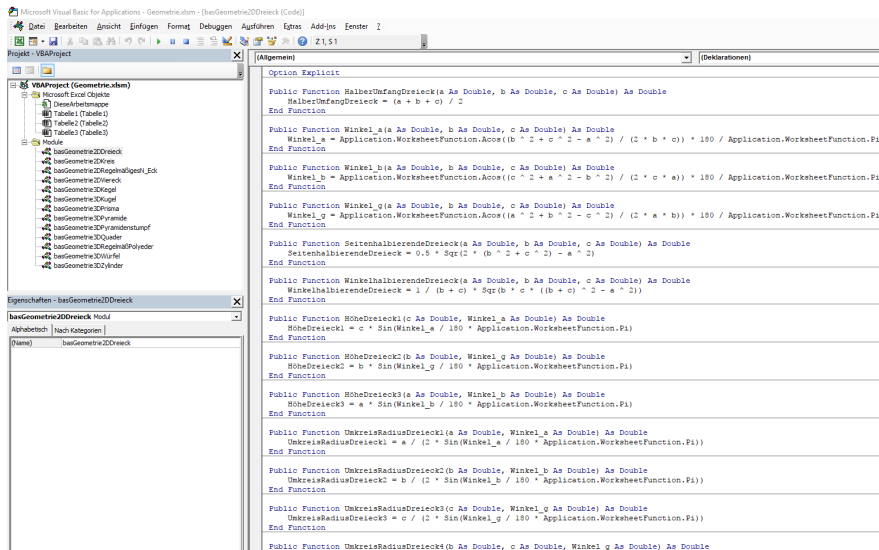


Abbildung 12.11 Neue benutzerdefinierte Funktionen werden erstellt.

- Testen Sie die Funktionen gut in Excel

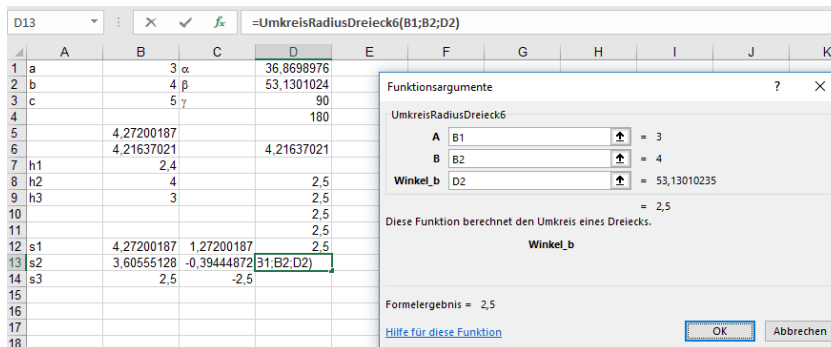


Abbildung 12.12 Die Funktionen stehen in Excel zur Verfügung.

■ Erstellen Sie die Startroutine:

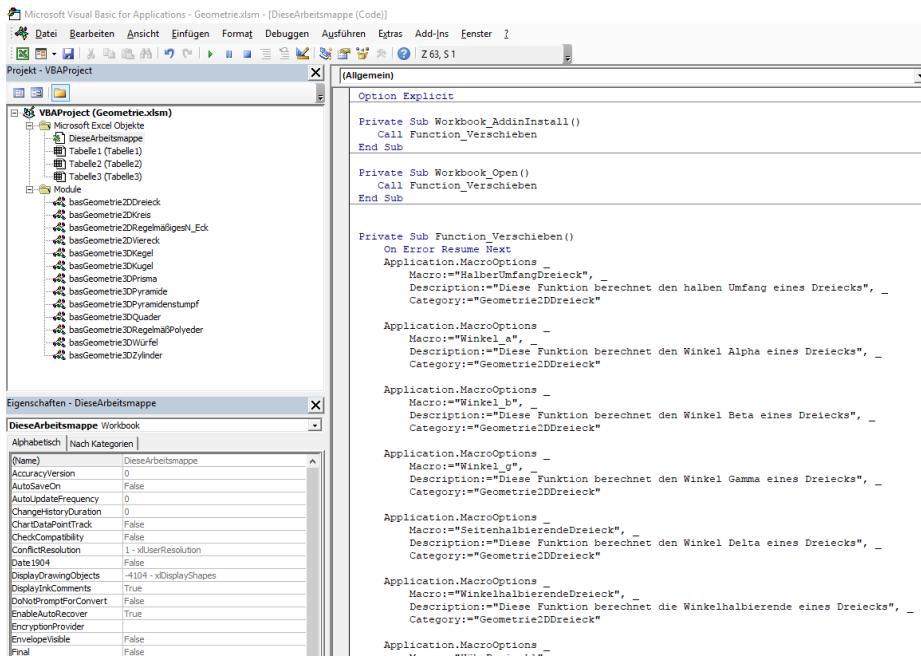


Abbildung 12.13 Den Makros werden verschiedene Kategorien zugewiesen.

- Speichern Sie die Datei als Add-In.
- Installieren Sie das Add-In.
- Nun stehen Ihnen die Funktionen in den neuen Kategorien zur Verfügung.

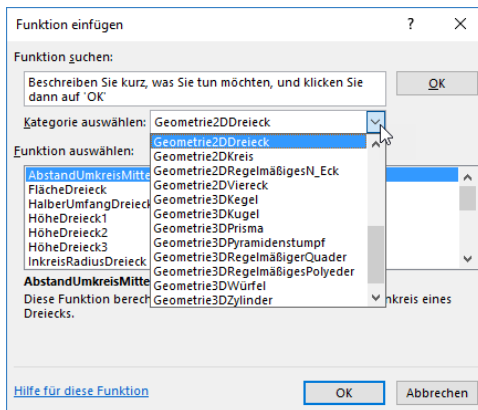


Abbildung 12.14 Neue Funktionen in neuen Kategorien

Zusammenfassung

Und so können eigene Funktionen auch in Formularen verwendet werden:

	A	B	C	D	E	F	G	H	I
1									
2									
3		Nur gültig für 2018							
4		Berechnung der Unterstützungsleistung nach B6, B7, C2, C3 d. Richtlinien							
5		für das Jahr 2018							
6									
7									
8		Name	(nur bei Bedarf und evt. Ausdruck)						
9									
10		Pers.Nr.	(nur bei Bedarf und evt. Ausdruck)						
11									
12									
13		Zur Ermittlung des Unterstützungsbetrages die mit					hinterlegten Felder ausfüllen		
14									
15			bei 0,5 Personen "x" eintragen						
16		Zu unterstützende Personen / Familienmitglieder	2						
17									
18		Zu berücksichtigendes Nettoeinkommen	1.000,00						
19									
20		Betriebszugehörigkeit	25		Jahre				
21									
22		Eigenanteil	150,00						
23									
24									
25									
26		Errechneter Unterstützungsbetrag					91	EUR	
27									
28									
29									

Abbildung 12.15 Die Unterstützungsleistung – eine komplexe Berechnung