VBA mit Excel

VBA steht für *Visual Basic for Applications.*
VBA ist eine vollwertige Programmiersprache, die hinter allen Programmen aus dem Microsoft Office Paket (und einigen anderen mehr, z. B. *CorelDraw* oder *AutoCad*) zur Verfügung steht. Mit VBA entwickelte Anwendungen können nicht zu alleine lauffähigen EXE-Programmen kompiliert werden – sie laufen immer nur in der zugeordneten Programmumgebung: Wenn Sie ein VBA-Programm also unter Excel entwickelt haben, dann benötigt ein User Excel, um es ausführen zu können (der kostenlose Excel-Viewer von Microsoft reicht dazu nicht aus).

⇨ Hinweis: *OpenOffice* ist ein für Privatanwender kostenloses Officepaket, das auch eine ähnliche Basic-Programmiersprache beinhaltet. Rufen Sie den OpenOffice.org-Basic-Editor über die Menüleiste auf: *Extras/ Makros/ Makros Verwalten/OpenOffice.org Basic …* wählen Sie anschließend MAIN aus und klicken Sie auf *bearbeiten.*

VBA startet man über *Extras/ Makro/ Visual Basic Editor* oder mit dem Shortcut: Alt+F11

In Direktfenster (*Ansicht/ Direktfenster* oder Strg+G) kann man einzelne Code-Zeilen eingeben, die beim Verlassen mit *Enter* ↵ ausgeführt werden.

⇨ Hinweis: In *OpenOffice Basic* gibt es kein Direktfenster. Hier geben Sie die Code-Zeilen bitte einzeln in die *Sub Main* ein und klicken anschließend auf den grünen *Ausführen*-Button in der Symbolleiste.

Wichtige Befehle:

- **Debug.print** oder **?**
  ‚Druckt' das Folgende in das Direktfenster
  Beispiel:
  ? "Hallo Welt" zeigt *Hallo Welt* in der folgenden Zeile (Text bitte immer in Anführungszeichen)
  ? 5+2*8^2 (ergibt 133) (Zahlen bitte immer ohne Anführungszeichen)

⇨ Hinweise
Rechenoperatoren sind:

^ Caret-Zeichen	Potenzierung:	5^2	entspricht 5^2 (also 5*5)	ergibt 25	} Potenz (^)
	Wurzel	25^(1/2)	entspricht $\sqrt[2]{25}$	ergibt 5	
* Asterix	Multiplikation	5*8	entspricht $5 \cdot 8$	ergibt 40	} Punkt (* und /)
/ Slash	Division	32/8	entspricht $\frac{32}{8}$ oder **32:8**	ergibt 4	
+ Plus	Addition	32+8		ergibt 40	} Strich (+ und -)
- Minus	Subtraktion	32-8		ergibt 24	

Die **Reihenfolge** der Abarbeitung ist *Potenz vor Punkt vor Strich* ^ ⇨ * / ⇨ + -
Will man diese Reihenfolge ändern, dann muss man **runde Klammern** setzen: ? ((5+2)*8)^2 (ergibt 3136)
Das **Dezimaltrennzeichen** ist der Punkt, nicht das Komma: 3.5 statt 3,5

& kaufmännisches UND Verknüpfung "Vorname" & " " & "Nachname" ergibt Vorname Nachname
(Achtung: Achten Sie darauf, dass vor und nach dem &-Zeichen **immer** eine Leerstelle ist)

Vergleichsoperatoren sind:
> größer als < kleiner als = gleich <> ungleich >= größergleich <= kleinergleich

- **Msgbox** *MsgBox* Prompt, [Buttons], [Title] (Parameter in eckigen Klammern sind *nicht* verpflichtend)

Beispiel:
MsgBox "Hallo Welt" *(linkes Bild)*
MsgBox "Hallo Welt",vbCritical+vbYesNo,"Neuer Titel" *(rechtes Bild oben)*
MsgBox 5+5,vbInformation+vbOKCancel,"Titel 2" *(rechtes Bild unten)*

⇨ Hinweise
Die Groß-Kleinschreibung ist gleichgültig.
Subprozeduren und Funktionen werden mit nachgestellten *Parametern* gesteuert, deren Reihenfolge erheblich ist. Die MsgBox-Prozedur erwartet als ersten Parameter den *Prompt*, also den Meldungstext. Ein Komma zeigt als Listentrennzeichen an, dass danach der zweite Parameter folgt, nach dem zweiten Komma folgt der dritte Parameter …
Text (in VBA heißt der *String*) wird immer in Anführungsstriche gesetzt – Text ohne Anführungsstriche wird von VBA als eine Variable interpretiert (wenn diese nicht anders ‚gefüllt' wurde, dann hat sie den Anfangswert *NULL*, ist also leer und hat den numerischen Wert 0)

- **Inputbox** *InputBox (*Prompt, [Title], [Default]*)*

Beachten Sie die runden Klammern um die Parameter der Inputbox-Funktion: sie zeigen an, dass die Inputbox-Funktion ein Ergebnis zurückgibt – nämlich die Eingabe des Users. *Default* ist ein voreingestellter Standardwert, der dem User vorgeschlagen wird.

Beispiel:

? InputBox("Wie heißen Sie?", "Ihr Name", "Mr. X") erzeugt das nebenstehende Eingabe-Fenster. Das Ergebnis dieser Inputbox ist die Eingabe des Users – in diesem Beispiel würde also *Mr. X* im Direktfenster ausgegeben werden (der dritte Parameter nach Prompt und Titel ist der *Default*-Wert).

MsgBox InputBox("Ihr Name bitte!", "Namenseingabe") erzeugt das zweite Eingabe-Fenster und gibt das, was der User in es eingegeben hat mit einer *Messagebox* wieder aus.

MsgBox InputBox("Ihr Name bitte!", "Namenseingabe"),vbAbortRetryIgnore+vbQuestion,"Username"
würde dasselbe Eingabefenster wie eben erzeugen. Die Eingabe des Users würde aber in einem veränderten Meldungsfenster ausgegeben werden *(rechtes Bild)*.

MsgBox InputBox("Erste Zahl", "Faktor1") InputBox("Zweite Zahl", "Faktor2"), vbInformation,"Produkt"*
*MsgBox InputBox("Erste Zahl", "Faktor1")**1** + InputBox("Zweite Zahl", "Faktor2"), vbInformation,"Produkt"*

⇨ Hinweise
Der Titel wird von *inputbox* als zweiter Parameter erwartet, während er bei *Msgbox* an dritter Stelle steht.
Bei *Inputbox* kann man keine Buttons oder Icons einstellen.
Die *Inputbox-Funktion* (mit Klammern) gibt den vom User eingegebenen Text als Ergebnis zurück – die *Msgbox-Funktion* (mit Klammern) gibt den Zahlenwert des vom User angeklickten Buttons als Ergebnis zurück.

⇨ Hinweise
OpenOffice-Basic akzeptiert den *1-Trick nicht. Hier müssen Sie den *String*, den die Inputbox-Funktion zurückgibt, ausdrücklich in eine Zahl umwandeln, bevor sie damit rechnen können. Verwenden Sie dazu die *CDbl-Funktion* (die folgende Code-Zeile funktioniert auch mit VBA):
*MsgBox **CDbl**(InputBox("Erste Zahl", "Faktor1"))+ **CDbl**(InputBox("Zweite Zahl", "Faktor2")), vbInformation,"Produkt"*

Prozeduren:

Funktion

- **Funktionen** und Subprozeduren müssen in einem *Modul* stehen.
- *Subprozeduren* sind wie *Makros*, die der Reihe nach Anweisungen ausführen.
- *Funktionen* sind *Subprozeduren* mit einem *Rückgabewert* (=Ergebnis). Funktionen haben also einen Rückgabewert – Subprozeduren haben keinen. Deshalb kann man Funktionen drucken, was mit Subprozeduuren nicht funktioniert: *? fktpi()*

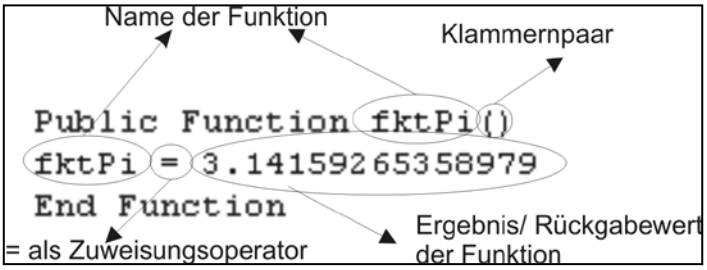
Beispiel (im Modul *Modul1*): *(Einfügen/ Modul Einfügen/ Prozedur:* Name: *fktPi Funktion Public)*

```
Public Function fktPi()
fktPi = 3.14159265358979
End Function
```

Diese Funktion kann an beliebiger Stelle in der aktuellen Mappe aufgerufen werden, z. B. in einer Zelle mit: *=fktPi()* oder im Direktfenster mit *? fktPi()* oder *MsgBox fktPi()*

fktPi() entspricht dem Ergebnis dieser Funktion, also dem Zahlenwert *3.14159265358979*.
Man kann mit *fktPi()* ebenso rechnen wie mit der Zahl, also z. B.: ? 5 * *fktPi()* + 2

Aufbau einer *Function*:
Im Funktionskopf stehen das Schlüsselwort *Public* und das Schlüsselwort *Function* mit einem Leerzeichen getrennt. Danach folgt nach einem weiteren Leerzeichen der Name der Funktion, gefolgt von einem Klammernpaar.



Im Körper der Funktion wird dem Namen der Funktion mit dem Gleichheitszeichen ein Ergebnis zugewiesen. Das ist der Rückgabewert der Funktion.
Den Fuß der Funktion bildet *End Function*.

Beispiel für eine Sub-Prozedur: (*Einfügen/ Prozedur:* Name: *subPi* *Sub* *Public)*

```
Public Sub subPi()
MsgBox "Pi ist " & 3.14159265358979
End Sub
```

Diese Subprozedur kann an beliebiger Stelle in der aktuellen Mappe aufgerufen werden, z. B. in Excel mit *Extras/ Makro/ Makros* Makroname auswählen *Ausführen*-Knopf anklicken oder im Direktfenster mit: *subPi*

⇨ Hinweise
Bei OpenOffice-Basic schreiben Sie die Funktion/ Sub-Prozedur unter die Sub Main. Anschließend rufen Sie sie in der Sub Main auf:

```
Sub Main
msgbox fktKreisumfang(CDbl(inputbox("Geben Sie einen Radius ein")))
End Sub
Public Function fktKreisumfang(varRadius)
fktKreisumfang =varRadius*2*3.14159265358979
End Function
```

Man kann Funktionen und Subprozeduren **Parameter** übergeben:

Beispiel: (*Einfügen/ Prozedur:* Name: *fEcHo* *Function* *Public)*

```
Public Function fEcHo(varEcho)
fEcHo =varEcho
End Function
```

Beachten Sie, dass *Function* mit *c* und nicht mit *k* geschrieben wird!

Diese Funktion gibt den Wert zurück, der ihr übergeben wurde: =fEcHo(2) in einer beliebigen Excel-Zelle ergibt also 2 *=fEcHo("hallo")* ergibt *hallo.*
Im Direktfenster gibt *? fecho(date()))* das aktuelle Systemdatum als Ergebnis aus.

Subprozeduren und Funktionen können mit den Parametern auch *rechnen*:

Beispiel 2: (*Einfügen/ Prozedur:* Name: *fPotenz* *Function* *Public)*

```
Public Function fPotenz(varBasis, varExponent)
fPotenz = varBasis ^ varExponent
End Function
```

Diese Funktion kann an beliebiger Stelle in der aktuellen Mappe aufgerufen werden, z.B. in einer Zelle mit: *= fPotenz (5;2)* **mit Semikolon ;** oder im Direktfenster mit *? fPotenz (5,2)* **mit Komma ,**

Beispiel 3: (*Einfügen/ Prozedur:* Name: *sPotenz* *Sub* *Public)*

```
Public Sub sPotenz(Romeo, Julia)
Range("A1") = Romeo ^ Julia
End Sub
```

Diese Subprozedur kann aufgerufen werden, indem Sie im Direktfenster einfach ihren Namen gefolgt von einer Leerstelle eingeben. Danach müssen Sie den Wert des Parameters *Romeo* und nach einem Komma den Wert des Parameters *Julia* eingeben. Wenn Sie anschließend die Enter-Taste betätigen, wird das Ergebnis von Romeo hoch Julia in die Zelle A1 der aktiven Tabelle eingetragen. *sPotenz 5,2* in A1 wird 25 eingetragen

Mit einem Parameter kann einer Subprozedur ein beliebiger Wert übergeben werden.
Dieser Parameter kann (fast) beliebig[1] genannt werden. Er muss in dem Klammernpaar hinter dem Namen der Funktion im Funktionskopf stehen. Die Reihenfolge der Parameter ist wichtig: Steht der Parameter *Romeo* an erster Stelle und *Julia* mit einem Komma getrennt an der zweiten, dann wird immer der zuerst übergebene Parameter der Variablen *Romeo* zugewiesen und der zweite der Variablen *Julia*. Der Aufruf *sPotenz 5,2* im Direktfenster bewirkt, dass die 5 der Variablen *Romeo* und dass die 2 der Variablen *Julia* zugewiesen wird. In der Subprozedur *sPotenz* wird gerechnet, indem die Variable Romeo (also der ihr zugewiesene Wert 5) mit Julia (also mit dem ihr zugewiesen Wert 2) potenziert wird. Das Ergebnis dieser Berechnung ($5^2 = 25$) wird anschließend in die Zelle A1 eingetragen.

[1] Vermeiden Sie alle Sonderzeichen, wie deutsche Umlaute sowie Leerzeichen, und verzichten Sie auf Zahlen am Anfang

Beispiel 4: (*Einfügen/ Prozedur:* Name: *kreisumfang Function Public)*

```
Public Function kreisumfang(radius)
kreisumfang = 2 * radius * 3.14159265358979
End Function
```

Diese Funktion wird in einer Excel-Zelle aufgerufen durch: *=kreisumfang(10)* Als Ergebnis erhält man den Umfang eines Kreises mit einem Radius von 10
Geben Sie in B1 ein: *=kreisumfang(A1) Als Ergebnis erhalten Sie den Umfang eines Kreises mit einem Radius von der Größe, die in A1 steht.*

⇨ Hinweis: In *OpenOffice Calc* können Sie eine eigene Funktion wie in Excel aufrufen mit: *= fktKreisumfang(100)*
Das ergäbe den Umfang eines Kreises mit einem Radius von einem Meter.

Die folgende Funktion erzeugt eine (ganze) Zufallszahl, die höchstens so groß ist, wie die als *Zahlmax* übergebene Integer-Variable (also Ganzzahl-Variable) und mindestens 0 ist:

```
Public Function fktZufallszahl(Zahlmax As Integer)
fktZufallszahl = Int((Zahlmax * Rnd))
End Function
```

Die INT-Funktion schneidet den Nach-Komma-Anteil der ihr übergebenen Zahl ab, rundet also immer auf 0 Dezimalstellen ab. Die RND-Funktion (*Random*) erzeugt eine Zufallszahl zwischen 0 und kleiner als 1.

Bezüge:

⇨ Hinweis: Das Objektmodell von *OpenOffice Basic* unterscheidet sich von jenem in Excel sehr. Wenn Sie wissen möchten, wie man dort Zellbereiche adressiert, dann zeichnen Sie ein Makro auf und schauen Sie sich anschließend den automatisch erzeugten Code an.

Sie können mit *Range(*"Bereich"*)* eine bestimmte Zelle adressieren : *Range("A1:C5")=123* bewirkt, dass in den 15 Zellen des Bereichs A1 bis C5 immer 123 steht. Mit *Range("A1:C5")=""* leeren Sie die angegebenen Zellen wieder ("" sind einfach zwei Gänsefüßchen direkt hintereinander). Mit *Range("Tabelle2!A1")="huhu"* schreiben Sie *huhu* in die Zelle A1 von Tabelle2. *ActiveCell=1* füllt die gerade aktive Zelle in Excel mit einer 1.
Mit der Offseteigenschaft können Sie den Eingabefokus relativ zur geraden aktiven Zelle verschieben: *ActiveCell.Offset(1, 0).Activate* bewirkt, dass die um eine Zeile unter der geraden aktiven Zelle liegende Zelle aktiviert wird. *ActiveCell.Offset(0, -5).Activate* lässt den Eingabefokus um 5 Spalten nach links springen.
Mit *Range("A1:D5").Select* markieren Sie die 20 besagten Zellen.
Zählen Sie diese mit *? selection.count* … es sind wirklich 20.

Eigenschaften:

⇨ Hinweis: In *OpenOffice Basic* ordnen Sie Ereignissen Code zu, indem Sie bei *Extras/ Makros/ Makros Verwalten/OpenOffice.org Basic …* auf den Button *Zuordnen* klicken. Wählen Sie anschießend das Ereignis aus, danach die auszuführende Prozedur.

Excel besteht aus ***Objekten*** wie Zellen, Tabellen, Mappen. Jedes Objekt hat eine bestimmte Anzahl von Eigenschaften.
Das Objekt Zelle hat als Eigenschaft einen Zell-Hintergrund (=*Interior*) und dieser hat u. a. die Eigenschaft einer Farbe (=*Color*). Diesen Wert können Sie mit dem = (Istgleich-Zeichen) als Zuweisungsoperator festlegen:

ActiveCell.Interior.Color = vbRed färbt die aktive Zelle rot ein.
Range("A1").Font.Color = vbGreen färbt die Schrift in Zelle A1 grün.

Erstellen Sie eigene Farben mit der ***RGB-Funktion***: *R* steht für *Rot*, *G* für *Grün*, und *B* für *Blau*: *ActiveCell.Interior.Color =rgb(255,0,0)* Da der höchste Wert für jede Farbe 255 ist und der niedrigste 0 würde obige Anweisung den Zellhintergrund der gerade aktiven Zelle rot einfärben.
Range("A1:C3").Font.Color=RGB(255,255,255) stellt die Schriftfarbe in den 9 Zellen des angegebenen Bereichs auf weiß.
Sie können auch die Rahmen der aktuellen Auswahl einstellen:
Selection.Borders(xlEdgeTop).LineStyle = xlContinuous bewirkt, dass ganz oben über der Auswahl eine Rahmenlinie erstellt wird. Ersetzen Sie *xlEdgeTop*, um andernorts Linien zu erzeugen*: xlEdgeLeft, xlEdgeBottom, xlEdgeRight, xlInsideVertical, xlInsideHorizontal* sowie *xlDiagonalDown* und *xlDiagonalUp*

Statt können Sie auch *xlContinuous, xlDash, xlDashDot, xlDashDotDot, xlDot, xlDouble, xlSlantDashDot* oder *xlLineStyleNone* einsetzen.
Mit *Selection.Borders(xlDiagonalUp).LineStyle = xlContinuous* streichen Sie die ausgewählten Zellen durch. *Selection.Borders(xlDiagonalUp).Color=RGB(0,0,255)* färbt die Diagonalen blau.

Mit einem Doppelpunkt können Sie im Direktfenster auch zwei Anweisungen gleichzeitig ausführen lassen:

```
Selection.Borders(xlDiagonalUp).LineStyle = xlContinuous:Selection.Borders(xlDiagonalUp).Color=RGB(0,0,255)
```

Ereignisse:

Excel besteht aus lauter Objekten: Excel selbst, jedes Tabellenblatt, ja jede Zelle, die Statuszeile, (ab Excel 2000:) der Registerreiter … das alles sind Objekte, die je mit einem bestimmten Satz an möglichen Ereignissen verbunden sind.
Excel führt eine ***Ereignisprozedur*** aus, wenn ein bestimmtes Ereignis eines bestimmten Objektes ausgelöst wird: Ereignisprozeduren liegen sozusagen auf der Lauer und sind bereit in Aktion zu treten, sobald das passende Ereignis ausgelöst wird. Dann werden Sie Zeile für Zeile abgearbeitet und liegen sofort wieder auf der Lauer.
Betrachten Sie einmal alle möglichen Ereignisse eines Tabellenblattes (englische Bezeichnung: *Worksheet*): Dazu wechseln Sie in das Code-Fenster indem Sie auf eine der Tabellen im Projekt-Explorer doppelklicken (oder Tabelle auswählen und *Ansicht/ Code* F7) Ganz oben über dem Code-Fenster sehen Sie zwei Kombinationsfelder: im linken *Objekt-Kombinationsfeld* wählen Sie bitte das Objekt *Worksheet* aus. Dann werden im rechten *Prozedur-Kombinationsfeld* alle Ereignisse dieses Objektes zur Auswahl gestellt.
Wählen Sie eines dieser Ereignisse aus, dann wird automatisch der Rumpf einer Ereignisprozedur erstellt, die genau dann ausgeführt wird, wenn das ausgewählte Ereignis des ausgewählten Objekts ausgelöst wird. Ein Beispiel:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
MsgBox Target.Address, vbInformation, "Zieladresse"
End Sub
```

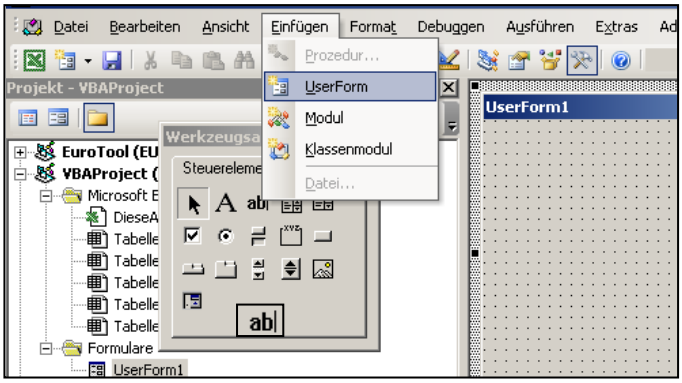
Obige Ereignisprozedur *Worksheet_SelectionChange(ByVal Target As Range)* wird ausgeführt, wenn das Ereignis *SelectionChange* einer Tabelle *Worksheet* ausgelöst wurde, wenn also der Eingabefokus einer oder mehrerer Zellen sich ändert, z. B. weil eine andere Zelle mit der Maus angeklickt wurde, weil man mit der Tabulator-Taste (oder der Eingabe-Taste oder einer Pfeiltaste) eine oder mehrere andere Zellen ausgewählt hat.
Wenn dieses Ereignis eintritt, dann wird eine Meldungsbox erzeugt, welche die Adresse des Zielbereichs *Target* ausgibt.

Sie können die Ereignisse eines Objektes auch im Objektkatalog (*Ansicht/ Objektkatalog* F2) nachschlagen. Dazu geben Sie als Suchtext *Worksheet* ein und klicken dann auf das Fernglas. Ereignisse sind mit einem Blitz gekennzeichnet.

Objekte:

Sie können in Excel ganz einfach eigene Objekte erstellen: Zeichnen Sie zum Beispiel einen Kreis (Shift-Taste ⇧ gedrückt halten) und klicken Sie anschließend mit der rechten Maustaste darauf. Wählen Sie aus dem aufspringenden Kontextmenü *Makro zuweisen*. Es öffnet sich ein Formular, auf dem alle Subprozeduren der Mappe aufgeführt werden. Wählen Sie die Subprozedur *Hallo* aus (die Sie natürlich vorher erstellt haben müssen). Wenn Sie nun den Kreis deselektieren und dann anschließend mit der Maus darauf klicken, dann wird die *Hallo*-Subprozedur ausgeführt.
Verwenden Sie ein beliebiges Bild, und machen Sie daraus einen Knopf, der beim Betätigen Excel beendet: *Application.Quit*

Formulare:



Erzeugen Sie ein eigenes Formular: *Einfügen/ Userform* in den Eigenschaften des neuen Formulars stellen Sie die Eigenschaft *ShowModal* auf *False* (sonst ist die Tabelle nicht mit der Maus erreichbar, solange das Formular geöffnet ist). Die *Caption*-Eigenschaft verändert den Formulartitel: *Dollarumrechner*. Plazieren Sie aus der Toolbox ein *Textfeld* und eine *Befehlsschaltfläche* auf dem Formular: *CommandButton1*: Caption auf *Umrechnen* -> ein Doppelklick auf den Button erstellt den Rumpf der Ereignisprozedur *Private Sub CommandButton1_Click()* , die immer ausgeführt wird, wenn auf den Commandbutton1 geklickt wird. Dort geben Sie als body ein:

*TextBox1 = TextBox1 * 1.1985* Diese Anweisung multipliziert den Wert in der Textbox mit dem Umrechnungskurs des Dollars (http://dollar-kurs.de). Wenn also vorher ein EUR-Wert in dem Textfeld gestanden war, so steht danach ein Dollarwert drin. Damit der Wert aus einer angeklickten Zelle in dem Textfeld *TextBox1* erscheint, erstellen Sie dafür eine Ereignisprozedur: Klicken Sie im linken Projekt-Explorer-Fenster doppelt auf *Tabelle1* und wählen Sie dort aus dem linken *Objekt*-Kombinationsfeld *Worksheet* und aus dem rechten *Prozedur*-Kombinationsfeld *SelectionChange* aus. VBA erstellt automatisch den Rumpf der Ereignisprozedur *Private Sub Worksheet_SelectionChange(ByVal Target As Range)*, die immer dann automatisch ausgeführt wird, wenn man eine andere Zelle in *Tabelle1* auswählt. Damit das Formular *UserForm1* angezeigt wird, geben Sie als erste Anweisung ein: *UserForm1.Show* In der zweiten Zeile weisen Sie der *TextBox1* den Wert der gerade aktiven Zelle zu:

UserForm1.TextBox1 = Target

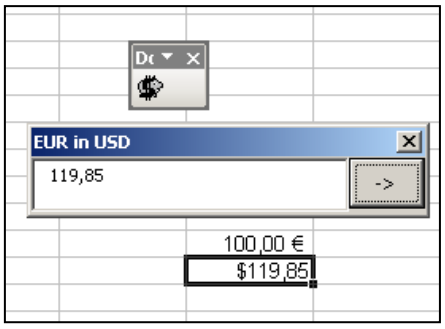
Target ist in der Ereignisprozedur *Private Sub Worksheet_SelectionChange(ByVal Target As Range)* diejenige Zelle, die neu ausgewählt wurde.

Immer, wenn Sie jetzt auf eine neue Zelle in Tabelle 1 klicken, öffnet sich unser Formular mit dem Wert der angeklickten Zelle im Textfeld. Mit einem Klick auf die Befehlsschaltfläche wird der Wert in Dollar umgerechnet. Sie können sich den Button auch ersparen und die Worksheet-Ereignisprozedur wie folgt ändern:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
UserForm1.Show
UserForm1.TextBox1 = Target * 1.1985
End Sub
```

Wenn Sie das Formular nicht bei jeder neuen Zellenauswahl zu sehen bekommen möchten, dann können Sie es auch über eine Subprozedur öffnen. Dazu schreiben Sie in ein Modul die folgende Subprozedur:

```
Public Sub DollarformularStarten()
UserForm1.Show
End Sub
```



Dieses Makro können Sie in Excel über *Extra/ Makro/ Makros* (oder Alt + F8) auswählen und mit *Ausführen* starten. Alternativ erstellen Sie eine Schaltfläche in einer eigenen Symbolleiste: Klicken Sie mit der rechten Maustaste auf eine Symbolleiste und wählen Sie den untersten Menüpunkt *Anpassen*. Es erscheint das Symbolleisten-Anpassungs-Fenster *Anpassen* – solange dies geöffnet ist, können Sie Ihre Symbolleisten verändern. Erstellen Sie im Register *Symbolleisten* eine eigne Symbolleiste mit einem Klick auf den Button *Neu*. Wählen Sie nun im Register *Befehle* den Eintrag *Makros* und dort *Benutzerdefinierte Schaltfläche*. Ziehen Sie mit gedrückter linker Maustaste das Smiley-Symbol in Ihre Symbolleiste. Mit

einem rechten Mausklick darauf können Sie anschließend das *Schaltflächensymbol ändern* und dann *bearbeiten* um ihm dann auch ein *Makro zuzuweisen*: Wählen Sie *DollarformularStarten* OK.

Elaboration:
Wenn Sie wollen, dass der Wert der neu ausgewählten Zelle sofort in unserem Formular in Dollar umgerechnet wird, dann multiplizieren Sie ihn doch gleich mit dem entsprechenden Kurs, wenn Sie ihn auslesen. Außerdem sollten Sie den Fehler abfangen, der auftritt, wenn der User mehrere Zellen auswählt:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
On Error GoTo FehlerMLDG
UserForm1.TextBox1 = Target * 1.1985
Exit Sub
FehlerMLDG:
UserForm1.TextBox1 = "Fehler"
End Sub
Und
Private Sub CommandButton1_Click()
On Error GoTo FehlerMSG
ActiveCell = Me.TextBox1 * 1
ActiveCell.NumberFormat = "[$$-409]#,##0.00"
Exit Sub
FehlerMSG:
Me.TextBox1 = "Fehler"
End Sub
```

Mit der *On Error GoTo* Sprungmarke - Routine erreichen Sie, dass die Prozedur im Falle eines Fehlers zu der Sprungmarke (mit Doppelpunkt am Ende) verzweigt.
Fügen Sie unbedingt die *Exit Sub*-Zeile ein, damit Ihnen die Prozedur nicht in die Fehlerbehandlung hinein läuft: *Exit Sub* unterbricht die Ausführung der Prozedur. Im Fehlerfall überspringen Sie diese Anweisung.

Verzweigungen mit Excel

Eine Verzweigung ist ein grundlegendes Element bei der Programmierung: je nachdem, ob eine Bedingung erfüllt wurde oder nicht, wird entweder dieser oder jener Code ausgeführt. Dadurch können Programme auf Usereingaben reagieren.
In Excel wird hierzu die *Wenn*-Funktion verwendet:

=Wenn(		A1>0	;	“positiv“	;	“negativ“)
= Wenn	(	Bedingung	;	Dann	;	Sonst)
= Wenn	(	entweder *wahr* oder *falsch*	;	wenn Bedingung wahr	;	Wenn Bedingung falsch)

Genau diese Ihnen von Excel vertraute Syntax wird von der IIF-Funktion verwendet:
Geben Sie im Direktfenster ein: *iif(varB>0,"Positiv","Negativ")* (Groß-Kleinschreibung ist irrelevant, es werden Kommas statt Strichpunkten verwendet, Strings [=Text] steht in Anführungsstrichen).
Sie werden *negativ* erhalten, wenn die Variable *varB* noch nicht von Ihnen definiert wurde (in diesem Fall ist ihr Wert 0 also nicht größer als 0). Füllen Sie nun *VarB* mit 5: *VarB = 5*
Wenn Sie nun die iif-Zeile erneut ausführen, wird *positiv* zurückgegeben.
Die IIF-Funktion erwartet die Übergabe von drei Parametern: der erste Parameter ist die Bedingung, die entweder *wahr* oder *falsch* sein kann. Der zweite Parameter wird mit einem Semikolon getrennt eingegeben und ist der, den die Funktion zurückgibt, falls die Bedingung wahr ist. Der dritte Parameter, der nach einem erneuten Komma steht, wird zurückgegeben, wenn die Bedingung falsch gewesen sein sollte.

Beispiel:

```
VarName=inputbox("Geben Sie Ihren Namen ein!","Ihr Name"):? iif(VarName="","Bitte geben Sie einen Namen ein","Hallo " & VarName)
```

Anderes Beispiel mit großer If-Funktion:

```
Public Sub Rabatt1(varRabattab, varRabatt)
If ActiveCell >= varRabattab Then     'Bedingung: Wenn die gerade ausgewählte Zelle einen Wert von mindestens der Rabattgrenze besitzt
   ActiveCell = ActiveCell - ActiveCell * varRabatt     'dann wird ihr der Rabatt abgezogen
End If                                                  'Ende des Verzweigungsblocks
End Sub
```

(Aufruf im Direktfenster mit *Rabatt1 50,0.3* der Rabatt beträgt 0,3 also 30%; er wird ab 50 gewährt)

Verzweigungen werden in VBA mit der noch größeren *If-Then-Else*-Funktion ausgeführt:

Beispiel:

```
Public Sub NegativezahlenGross()
If ActiveCell < 0 Then              'Bedingung: Wenn die gerade ausgewählte Zelle einen negativen Wert besitzt, dann
   ActiveCell.Font.Size = 16        'soll die Schriftgröße auf 16 eingestellt werden
   ActiveCell.Interior.Color = vbRed    'soll die Hintergrundfarbe der Zelle auf rot eingestellt werden
   ElseIf ActiveCell = 0 Then       'Bedingung 2: Oder Wenn die gerade ausgewählte Zelle den Wert 0 besitzt, dann
```

```
   ActiveCell.Font.Size = 12          'soll die Schriftgröße gleich 12 sein und
   ActiveCell.Interior.Color = vbYellow  'der Zellhintergrund soll gelb sein.
   Else                               'andernfalls, wenn keine Bedingung erfüllt wurde und der Wert also weder negativ noch 0 ist
   ActiveCell.Font.Size = 12          'soll die Schriftgröße auf 12 eingestellt werden
   ActiveCell.Interior.Pattern = 0    ' soll keine Hintergrundfarbe zugewiesen werden
End If                                'Ende des Verzweigungsblocks
End Sub
```

Eine andere Möglichkeit zu verzweigen ist die Select-Case-Anweisung, die weiter hinten an der Funktion *GeburtsWochentag* demonstriert wird.

Wichtige Funktionen:

Stringfunktionen (Textfunktionen)

Text heißt bei VBA *String*. Strings werden in Anführungszeichen geschrieben: *"Das ist ein Text"*
VBA von Excel kennt die gleichen Funktionen wie Excel – dort müssen sie aber immer unter ihrem *englischen Originalnamen* aufgerufen werden.

VBA	Excel
Left (String, Anzahl der Ziffern)	*=Links (String; Anzahl der Ziffern*
Right(String, Anzahl der Ziffern)	*= Rechts(String; Anzahl der Ziffern)*
Mid (String,Startziffer,Länge)	*=TEIL("Elvira";2;3) -> lvi*
Len(String)	*=Länge("Julius") -> 6*
InStr(DurchsuchtString, Suchstring)	*=FINDEN(Suchstring; DurchsuchtString) -> 13* ↺!
UCase(String)	*=Gross("test") -> TEST*
LCase(String)	*=Klein("TeSt") -> test*
String(Anzahl, String)	*=WIEDERHOLEN("a";10) -> aaaaaaaaaa* ↺!

↺! bedeutet, dass die Reihenfolge der Argumente verändert ist

Datums- und Zeitfunktionen

VBA kann wie Excel auf eine Reihe mächtiger Datums- und Zeitfunktionen zugreifen.
VBA verfolgt das gleiche Datumskonzept wie Excel: Jedem Datum entspricht ein Zahlenwert. Dabei bedeutet 1 einen Tag, so dass ein Datum -1 dem Datum des Vortages von diesem Datum entspricht. Die Funktion *fktgestern* ergibt das gestrige Datum (vom Systemdatum aus gesehen):

```
Public Function gestern()
gestern = Date - 1
End Function
```

Im Direktfenster kann man sich mit *? date()* das heutige und mit *? gestern()* das gestrige Datum ausgeben lassen.

Date	=heute()
Now	=jetzt()
Weekday	=Wochentag()

```
Public Function GeburtsWochentag(Geburtstag)
Dim var1 As Integer
var1 = Weekday(Geburtstag)
Select Case var1                    'Select Block beginnt. var1 wird als die Verzweigung bestimmender Wert festgelegt
   Case 1                           'Fall dass var1 dem Wert 1 entspricht
     GeburtsWochentag = "Sonntag"   'wenn var1 den Wert 1 besitzt, dann hat die Funktion Geburtswochentag das Ergebnis Sonntag
   Case 2                           'Fall var1 hat den Wert 2
      GeburtsWochentag = "Montag"   wenn var1 den Wert 2 besitzt, dann hat die Funktion Geburtswochentag das Ergebnis Montag
   Case 3                           'Fall var1 hat den Wert 3
      GeburtsWochentag = "Dienstag" 'Ergebnis: Dienstag
   Case 4                           'Fall: var1 ist 4
     GeburtsWochentag = "Mittwoch"  'Ergebnis: Mittwoch
   Case 5                           '…
      GeburtsWochentag = "Donnerstag"
   Case 6
      GeburtsWochentag = "Freitag"
   Case 7
      GeburtsWochentag = "Samstag"
End Select                          'Ende des Select-Blocks
End Function
```

Obige Funktion gibt den Wochentag als Text zurück, an dem jemand geboren wurde.
Im Direktfenster kann man diese Funktion aufrufen mit:
? GeburtsWochentag(#28/1/1964#)
In Excel übrigens hieße obige übersichtliche Funktion:
=WENN(WOCHENTAG(A1)=1;"Sonntag";WENN(WOCHENTAG(A1)=2;"Montag";WENN(WOCHENTAG(A1)=3;"Dienstag";WENN(WOCHENTAG(A1)=4;"Mittwoch";WENN(WOCHENTAG(A1)=5;"Donnerstag";WENN(WOCHENTAG(A1)=6;"Freitag";"Samstag")))))) alles in *einer* Zeile.

Die Wenn-Funktion kann sieben mal in sich verschachtelt werden – verliert aber schnell an Übersichtlichkeit…

DateSerial(*Jahr, Monat Tag*) timeserial(Stunde, Minute,Sekunde)	=Datum(*Jahr;Monat;Tag*) Zeit(Stunde; Minute;Sekunde)
Dateadd(Intervall,Einheiten,Datum)	
DateDiff(Intervall, Datum1, Datum2)	
DatePart(Intervall, Datum)	
Second, Minute, Hour, Day, Month, Year	Sekunde, Minute, Stunde, Tag, Monat, Jahr
Time	

? datediff("d",date(),dateserial(Year(date()),12,24)) gibt aus, wie viele Tage es noch bis um diesjährigen Weihnachten sind (bzw. waren).

? datepart("ww",date()) im Direktfenster gibt die aktuelle Woche zurück.

? hour(time()) gibt den Stundenanteil der aktuellen Systemzeit aus.

Schleifen mit Excel

Sie können einen Vorgang mit einer Schleife wiederholt ausführen und dabei einen *Zeiger* wandern lassen.

Die *For Each-Schleife*

Mit einer *For-Each-Schleife* können Sie eine Gruppe an *Objekten* durchlaufen, das sind z.B. alle geöffneten Arbeitsmappen, alle Tabellen einer Arbeitsmappe oder alle Zellen einer Auswahl.

Beispiele:

```
Public Sub subTabellenNamen()
   Dim tn As Worksheet
   Dim i As Integer
   For Each tn In ThisWorkbook.Worksheets
      i=i+1
      MsgBox tn.Name,,“Tabelle Nr.: “ & i
   Next tn
End Sub
```

Obige Subprozedur (im Modul1) bringt für jede Tabelle in der aktuellen Arbeitsmappe eine Meldungsbox mit dem Namen dieser Tabelle.

In den beiden Zeilen mit dem Schlüsselwort ***Dim*** werden Variablen deklariert. Die Variable *tn* steht demnach als Stellvertreter für Tabellen (=Worksheet). Die Variable *i* steht dagegen für eine Integer-Zahl, also eine ganze Zahl bis höchstens 32.767. Diese soll hier als Zähl-Variable eingesetzt werden.

Nun folgt die *For-Each*-Schleife: Übersetzt bedeutet diese Zeile: Für jede Tabelle (hier repräsentiert durch das die Tabellen-Variable *tn*) in den Tabellen (=Worksheets) der der aktuellen Arbeitsmappe (=ThisWorkbook) soll das Folgende bis zum Schlüsselwort *Next* ausgeführt werden. Der Objekt-Vartiablen *tn* wird das erste Objekt, hier also die erste Tabelle der aktuellen Arbeitsmappe zugewiesen. *tn* steht ab nun also als Abkürzung für die erste Tabelle.

In dem mathematisch widersinnigen Ausdruck i=i+1 ist das ‚=’-Zeichen ein Zuweisungsoperator. Das Istgleichzeichen weist dem i (auf der linken Seite) einen um 1 höheren Wert als vorher (i auf der rechten Seite) zu. Nach dieser Zeile ist i also um 1 größer als vor der Zeile. Dadurch wird i bei jedem Schleifendurchgang um 1 erhöht und zählt also die Schleifendurchgänge mit. Da eine Integer-Variable anfänglich den Wert 0 besitzt, hat das i nach dem ersten Durchgang den Wert 1.

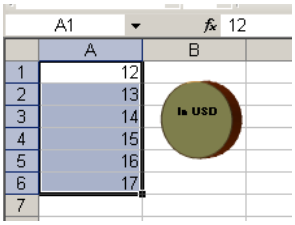
In der folgenden Zeile wird eine Meldungsbox mit dem Namen der Tabellen (tn.Name) erzeugt, die als Titel (dritter Parameter der Msgbox-Funktion nach dem zweiten Komma) auch die Zählvariable i ausgibt.

Die folgende Zeile mit dem Schlüsselwort *Next* bewirkt, dass die *For-Each*-Schleife noch mal von vorne beginnt. Nun wird das nächste Objekt, hier also die Objekt-Variable *tn* mit dem nächsten Objekt gefüllt, so dass die Variable tn nun für die zweite Tabelle der aktuellen Arbeitsmappe steht.

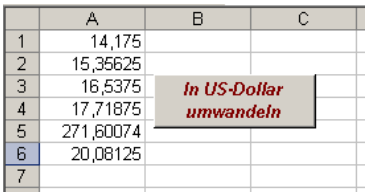
Beispiele:

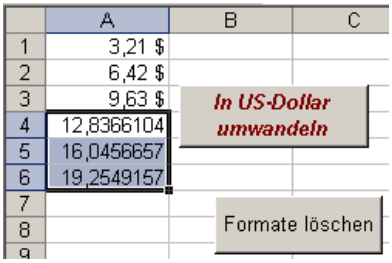
```
Public Sub subEURinUSD()
On Error Resume Next
Dim varA As Range
For Each varA In Selection
   varA.Value = varA.Value * 1.18125
Next varA
End Sub
```

Obige Subprozedur (im Modul1) rechnet den Wert jeder Zelle (=Range) in der aktuellen Auswahl (=Selection) von EUR in USD um (Kurs vom 10.12.2005).



In der ersten Zeile wird die Prozedur angewiesen, bei Fehlern so tun, als wäre nichts passiert, also nicht mit einer Fehlermeldung abzubrechen: bei einem Fehler mach mit dem nächsten weiter (das sollte an dieser Stelle gemacht werden, weil sich in der Auswahl später auch Zellen mit Text befinden könnten, was bei dem Versuch sie umzurechnen zu einem Abbruch der Prozedur führen würde). In der Zeile mit dem Schlüsselwort ***Dim*** wird die Variable *varA* als Zell-Bereich (=Range) deklariert. Die Variable *varA* steht demnach als Stellvertreter für eine oder mehrere Zellen.
Nun folgt die *For-Each*-Schleife, die alle Zellen in der aktuellen Auswahl durchläuft.
Die Variable varA steht beim ersten Schleifendurchlauf für die erste Zelle der Auswahl (links oben). Ihr Wert (=Value) wird nun mit dem Zuweisungsoperator ‚=' ersetzt durch das Ergebnis der Multiplikation des ihres bisherigen Wertes mit dem Dollarkurs. Statt des bisherigen Wertes steht also nach Abarbeitung dieser Code-Zeile der um 1,18125 höhere Wert in der ersten Zelle der Auswahl.
Die folgende Zeile stellt das Format
Mit dem Schlüsselwort *Next* wird die Schleife zur nächsten Runde angewiesen. VBA springt wieder zu der *For-Each*-Zeile und füllt die Zell-Variable *varA* mit der zweiten Zelle der Auswahl (nächste Spalte, dann nächste Zeile). *varA* steht nun also als Stellvertreter für die zweite Zelle der Auswahl.
Die Schleife wird so lange wiederholt, bis alle Objekte der aktuellen Auswahl abgearbeitet sind.
Um dem User anzuzeigen, dass die aktuellen Werte nunmehr Dollar-Werte sind, können Sie Ihre Prozedur noch um die folgende Code-Zeile ergänzen, die Sie bitte als 6. Zeile unmittelbar über der *Next varA* - Zeile einfügen: *varA.NumberFormat = "#,##0.00 [$$-409]"*
Diese Zeile formatiert die bearbeitete Zelle als Dollar.





Belegen Sie nun noch einen Button mit der Subprozedur, so dass sie sie mit einem Klick auf den Button auslösen können:
Erstellen Sie einen Kreis (*Ansicht/ Symbolleisten/ Zeichen/ Ellipse*), Rechter Mausklick und im Kontextmenü *Makro zuweisen* auswählen, im Listenfeld dann *subEURinUSD* auswählen und mit OK bestätigen. Wenn Sie nun irgendwo hinklicken und anschließend auf den Button klicken, dann wird der Wert derjenigen markierten Zellen, die eine Zahl enthalten, um den Faktor 1,8125 erhöht. Wenn Sie nun noch in den Kreis schreiben „In USD", dann haben Sie in Ihrer Tabelle einen EUR-in-USDollar-Umwandlungsknopf.
Oder verwenden Sie einen ‚echten' Button (eine Schaltfläche) aus der Symbolleiste *Formular*: Wenn Sie eine Schaltfläche aufgezogen haben, dann werden Sie automatisch nach einer zugeordneten Sub gefragt: wählen Sie dann *subEURinUSD* aus.

Erzeugen Sie mit *Symbolleiste/ Formular -> Schaltfläche* einen neuen Button, wählen Sie im Folgefenster *Neu* und geben Sie den folgenden Code ein:

```
Sub Schaltfläche2_BeiKlick()
Dim Zelle As Range
For Each Zelle In Selection
   Zelle.NumberFormat = ""
Next Zelle
End Sub
```

Benennen Sie den Button sinnvoll. Wenn Sie anschließend auf den neuen Button klicken, löscht er die Formate der ausgewählten Zellen.

Anderes Beispiel:

```
Public Sub Rabatt2(varRabattab, varRabatt)
Dim x As Range
For Each x In Selection        ' Schleifenbeginn: alle Zellen der Auswahl werden durchlaufen. x steht für die je aktuelle Zelle
   If x >= varRabattab Then    'Bedingung: Wenn die gerade ausgewählte Zelle einen Wert von mindestens der Rabattgrenze besitzt
      x = x - x * varRabatt    'dann wird ihr der Rabatt abgezogen
   End If                      'Ende des Verzweigungsblocks
Next x                         'Ende der Schleife
End Sub
```

Die *For Next-Schleife*

Mit einer *For-Next-Schleife* können Sie eine Schleife beliebige Male durchlaufen lassen.

```
For i = 1 To 10 : MsgBox i : Next i
```

(Der Doppelpunkt bewirkt, dass Sie mehrere Zeilen in eine Zeile schreiben können)
Obige Schleife im Direktfenster gibt 10 Meldungsfenster aus, die nacheinander die Zahlen 1 bis 10 melden.

```
Public Sub TabellenNamen()
Dim i As Integer, varMsg As String
For i = 1 To ActiveWorkbook.Sheets.Count      ' ActiveWorkbook.Sheets.Count gibt die Anzahl der Tabellen der Mappe an
   varMsg = varMsg & vbCrLf & Sheets(i).Name  'varMsg wird bei jedem Schleifendurchgang um einen Tabellennamen erweitert
Next i                                        'Falls die Zahl von ActiveWorkbook.Sheets.Count noch nicht erreicht wurde: neuer Schleifendurchgang

MsgBox Right(varMsg, Len(varMsg) - 2)         'Ausgabe der Textvariablen varMsg ohne die beiden ersten Ziffern: Leerstelle und Zeilenumbruch

End Sub
```

Obige Subprozedur geht die Schleife so oft durch, wie es Tabellen in dieser Arbeitsmappe gibt. Bei jedem Durchgang liest sie den Namen einer Tabelle aus und ergänzt die Textvariable *varMsg* um diesen Namen. Am Ende gibt sie alle Ziffern von *varMsg* bis auf die ersten beiden (von rechts gesehen also die volle Länge der Textvariable minus 2) aus, denn diese beiden ersten Ziffern stellen eine Leerstelle und einen Zeilenumbruch dar, die beim ersten Schleifendurchgang erzeugt wurden, als *varMsg* noch leer war.

Die *Do-Loop-until-Schleife*

Mit einer *Do-Loop-until-Schleife* führen Sie eine Schleife so oft aus, bis ein bestimmter Zustand erreicht ist:

```
Public Sub ExcelistToll()
Dim i As String
Do
   i = InputBox("Excel ist ...", "Excel ist toll!!!")
Loop Until i = "toll"
End Sub
```

Obige Subprozedur wird so lange ausgeführt, bis die String-Variable i den Wert „toll“ hat. Erst dann, wenn der User also in die Inputbox *toll* eingegeben hat, wird diese Schleife beendet – sonst geht sie endlos weiter (oder lässt sich nur mit Strg+Pause unterbrechen).

Die *Do-While-Loop -Schleife*

Mit einer *Do-While-Loop-Schleife* ist beinahe so, wie die vorherige *Do-Loop-until-Schleife*. Weil die While-Schleife aber Kopfgesteuert ist (die Bedingung steht im Schleifenkopf) kann es sein, dass diese Schleife nicht ein einziges Mal ausgeführt wird.

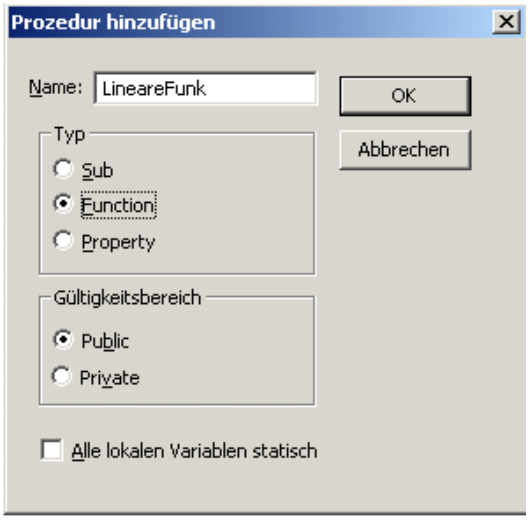
```
Public Sub ExcelistToll2()
Dim i As String
Do While i <> "toll"
   i = InputBox("Excel ist ...", "Excel ist toll!!!")
Loop
End Sub
```

Obige Subprozedur leistet das gleiche wie die *Do-Loop-until-Schleife* davor.
Wenn man unmittelbar nach der Deklaration der Text-Variablen i in beiden Subprozeduren diese auf den Wert *toll* einstellt (i=“toll“), dann wird die *Do-Loop-until-Schleife* ausgeführt – und i wird neu bestimmt - die *Do-While-Loop-Schleife* wird dagegen keinmal aufgerufen.

Benutzung von Excel als Funktionenplotter

Lineare Funktionen

1. Nach dem Start von Excel wechseln Sie über *Extras/ Makro/ Visual Basic-Editor* zum VBA-Editor
2.
3. Im Projektexplorer (linkes Fenster oben) befindet sich ein Eintrag *VBAProjekt(*Mappenname*)*. Wählen Sie das aus.
   Fügen Sie über *Einfügen/ Modul* ein Modul hinzu. Der Eingabefokus wechselt in das Code-Fenster Ihres neuen Moduls.
   Dort können Sie Funktionen definieren, die in der aktuellen Excel-Mappe verfügbar sind.



4. Fügen Sie über *Einfügen/ Prozedur* eine Funktion ein:

   Einer Funktion können innerhalb des Klammernpaares hinter ihrem Namen **Parameter** übergeben werden. Also füllen Sie in die Klammern hinter dem Funktionsnamen die drei Parameter einer linearen Funktion an:
   Für die Funktion $f(x) = mx + n$ m, x und n, also:

   LineareFunk (x,m,n)

5. Die in der Klammer festgelegten Variablen können nun innerhalb der Funktion verarbeitet werden.
   Den Rückgabewert einer Funktion legt man fest, indem man dem Namen der Funktion den Rückgabewert zuweist.
   In VBA-ländisch übertragen bedeutet das:

   *LineareFunk =m*x+n* So ähnlich wie: $f(x) = m \cdot x + n$

6. Überprüfen Sie das Funktionieren Ihrer Funktion, indem Sie sie im Direktfenster aufrufen (dieses befindet sich unter dem Code-Fenster. Falls es nicht da sein sollte, öffnen Sie es mit Strg. + G).
   Geben Sie im Direktfenster ein: *? LineareFunk(1,2,3)*
   Das Fragezeichen bedeutet, dass der Rückgabewert der folgenden Funktion im Direktfenster ausgegeben werden soll. *LineareFunk(1,2,3)* ruft die Funktion *LineareFunk() mit den Parametern 1, 2 und 3* auf. Weil 1 an erster Stelle steht, setzt VBA es als Wert für den ersten Parameter, also für x, ein.
   ? LineareFunk(1,2,3) bedeutet also:

   > Gib im Direktfenster das Ergebnis der Funktion *LineareFunk()* aus, wobei x = 1, m = 2 und n = 3 ist

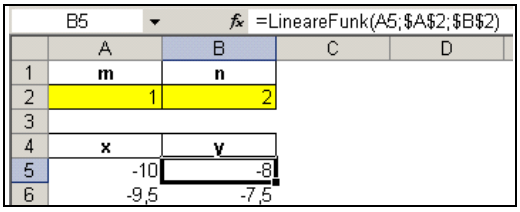
7.
8. Schließen Sie den VBA-Editor und wechseln Sie zu Excel.
9. Geben Sie die folgenden **Überschriften** ein: in A1 *n*, in B1 *m* in A4 *x* und in B4 *y*. Zentrieren. Schalten Sie den Hintergrund der darunter liegenden Felder A2 und B2 auf die **Füllfarbe** Gelb. Schalten Sie für A2 und B2 den **Schutz** ab: die beiden Zellen markieren, dann: *Format/ Zellen/ [Schutz] Geschützt* ohne Häkchen) Blenden Sie die **Gitternetzlinien aus** (*Extras/ Optionen/ [Ansicht] Gitternetzlinien* ohne Häkchen), erstellen Sie **Rahmen** um A1 bis B2 und um A4 bis B45.

10. Geben Sie den Beginn der Werteliste ein:
Schreiben Sie in A5 *-10*, dann darunter in A6 *-9,5*.

11. Geben Sie die Formel zur Berechnung des Funktionswertes ein:
Setzen Sie den Fokus auf B5.

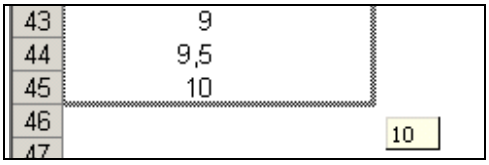
Eine Funktion wird immer mit dem =-Zeichen und ihrem Namen aufgerufen. In Klammern werden die Parameter übergeben: Vorsicht: Während die Parameterliste in VBA mit Kommas getrennt wurde, verwendet das deutsche Excel dafür Strichpunkte.
Obiger Test (Punkt 5) würde in Excel also so aussehen: *=LineareFunk(1;2;3)*
Nun sollen aber keine bestimmten Zahlen, sondern Zellbezüge eingegeben werden.
Man unterscheidet in Excel die normalen relativen Bezüge, die bei Verschiebungen von Zellen mitwandern, von absoluten Beziehungen, die man am $-Zeichen erkennt.

Weil *m* und *n* für die ganze Funktion gleich bleiben sollen, müssen sie absolut referenziert werden Der Bezug auf das X soll dagegen mitwandern. Die Formel in B5 lautet also: ***=LineareFunk(A5;A2;B2)***
Als Ergebnis erhalten Sie 0. Geben Sie in A2 *1* und in B2 *2* ein, dann steht in B5 *-8*



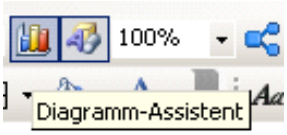
12. Markieren Sie nun B5, bewegen den Cursor auf die rechte untere Ecke der Markierung, so dass der Cursor sich in ein + verwandelt, halten Sie die linke Maustaste gedrückt, ziehen Sie die Maus um eine Zeile nach unten und lassen Sie dann die Maustaste los. In B6 steht nun: **=LineareFunk(A6;A2;B2)** A5 ist also zu A6 geworden, während alles andere gleich geblieben ist. Als Ergebniswert in B6 erhalten Sie *-7,5*

13. Markieren Sie nun die Felder A5 bis B6 und greifen den rechten unteren Rand der Markierung, so dass sich der Cursor wieder in ein + verwandelt. Halten Sie die linke Maustaste gedrückt, und ziehen Sie die Markierung bis B45. Lassen Sie anschließend die Maustaste wieder los.



Sie haben den X-Werten von -10 bis +10 (in 0,5-Schritten) in der Spalte A Y-Werte in der Spalte B zugeordnet. Diese Y-Werte berechnen sich aus der Funktion *LineareFunk()* mit nebenstehendem *x* und dem *m* aus A2 und dem *n* aus B2

14. Lassen Sie den Bereich A5 bis B45 markiert. Klicken Sie auf den Diagramm-Assistenten. Wählen Sie dort: *Punkt(XY)* und *Punkte mit interpolierten Linien und ohne Datenpunkte*
Klicken Sie zweimal auf *weiter*. Stellen Sie dann bei *[Gitternetzlinien]* alle Haupt- und Hilfsgitternetze ein (Häkchen davor) und schalten Sie bei *[Legende]* die Legende ab (Häkchen entfernen). Wählen Sie *fertig stellen*



15. Doppelklicken Sie auf die Gitternetzlinien (einmal auf *Größenachse Y* und dann auf *Größenachse X)* und stellen Sie bei *[Skalierung] Minimum* auf nicht automatisch *-10* und *Maximum* auch nicht automatisch auf *10* ein. Das Hauptintervall sei *5*, das Hilfsintervall *1*
Zur besseren Übersichtlichkeit können Sie die Zeichnungsfläche löschen sowie die X- und die Y-Achse *stärker* einstellen. Auch den Graphen selbst sollten Sie ‚stärken'.

16. Stellen Sie nun bei *Extras/ Schutz* / *Blatt schützen* ein. Übernehmen Sie alle Vorgaben, geben Sie kein Passwort ein. Jetzt können nur noch die ungeschützten Zellen A2 und B2 verändert werden.

17. Experimentieren Sie mit der Steigung *m* und dem Y-Achsenabschnitt *n*.

Quadratische Funktionen

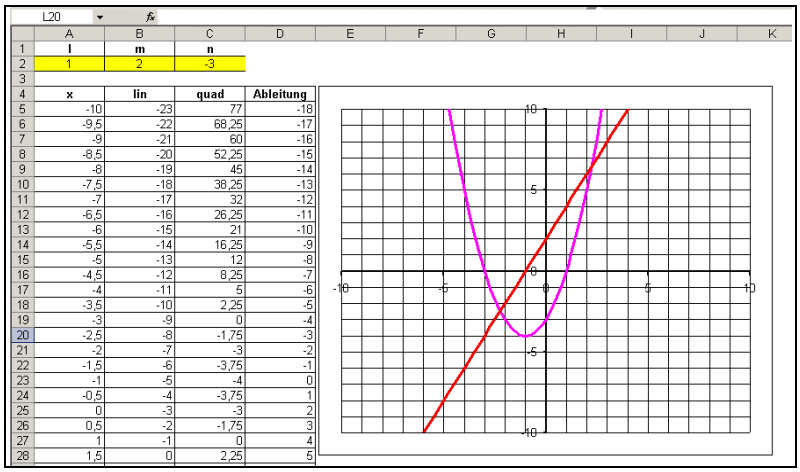
1. Heben Sie den Blattschutz wieder auf: *Extras/ Schutz/ Blattschutz aufheben*
2. Wechseln Sie über *Extras/ Makro/ Visual Basic-Editor* zum VBA-Editor.
   Geben Sie dort die folgende Funktion ein:

   Public Function QuadrFunk(x, l, m, n)
   *QuadrFunk = l * x ^ 2 + m * x + n*
   End Function

   $$f(x) = l \cdot x^2 + m \cdot x + n$$
3. Verfahren Sie wie bei der linearen Funktion. Ermöglichen Sie das Variieren von *l*, *m* und *n* in A1 bis C3
4. Neben der X-Werteliste in A5 bis A45 schreiben Sie in B5 die folgende Formel:

   =QuadrFunk(A5;A2;B2;C2)

   Diese ordnet den Wert in A5 also der Variablen *x* zu (relativer Bezug, der beim Herunterkopieren gleich mitwandert). Die Variable *l* wird mit dem Wert in A2 gefüllt, *m* wird B2 und *n* wird C2.
5. Ziehen Sie B5 wie oben herunter bis zu B45
6. Markieren Sie A5 bis B45 und rufen Sie den Diagrammassistenten auf. Erstellen Sie das bekannte Punkte(XY)-Diagramm und stellen Sie die Werte nach Ihrem Belieben ein. Mit l=1, m=0 und n=0 erhalten Sie die Normalparabel.



	A	B	C	D
1	l	m	n	
2	1	2	-3	
3				
4	x	lin	quad	Ableitung
5	-10	-23	77	-18
6	-9,5	-22	68,25	-17
7	-9	-21	60	-16
8	-8,5	-20	52,25	-15
9	-8	-19	45	-14
10	-7,5	-18	38,25	-13
11	-7	-17	32	-12
12	-6,5	-16	26,25	-11
13	-6	-15	21	-10
14	-5,5	-14	16,25	-9
15	-5	-13	12	-8
16	-4,5	-12	8,25	-7
17	-4	-11	5	-6
18	-3,5	-10	2,25	-5
19	-3	-9	0	-4
20	-2,5	-8	-1,75	-3
21	-2	-7	-3	-2
22	-1,5	-6	-3,75	-1
23	-1	-5	-4	0
24	-0,5	-4	-3,75	1
25	0	-3	-3	2
26	0,5	-2	-1,75	3
27	1	-1	0	4
28	1,5	0	2,25	5

7. Natürlich können Sie in Ihrem Diagramm auch mehrere Graphen gleichzeitig aufnehmen:

Verfeinern: Ergänzen Sie Ihr *Excelsheet* durch bequeme Buttons.

1. Wählen Sie in der Symbolleiste *Formular* (Öffnen mit *Ansicht/ Symbolleisten/ Formular*) das Icon für *Schaltfläche*. Ziehen Sie mit gedrückter linker Maustaste eine kleine Schaltfläche unter A2 auf, die nur etwa halb so breit wie A ist.
2. Bei der Aufforderung *Makro zuweisen* wählen Sie *Neu*
3. Sie landen automatisch im VBA-Editor. In den automatisch erzeugten Prozedur-Rumpf schreiben Sie den folgenden Befehl:

   Range("A2") = Range("A2") - 1

   Dieser bewirkt, dass der Wert der Zelle A2 um 1 erniedrigt wird.
   Geben Sie als Namen für die Schaltfläche einfach ein – ein
   Verfahren Sie analog mit einem +-Button: *Range("A2") = Range("A2") + 1*
   Wenn Sie l lieber in kleineren Schritten verändern wollen, dann schreiben Sie zum Beispiel: *Range("A2") = Range("A2") – 0.1*
   Achtung: In VBA ist der Punkt das Dezimaltrennzeichen, nicht das Komma.