Workshop 4: VBA-Programmierung mit MS Excel

1 Prozeduren und Funktionen in Excel

Sachlogisch zusammengehörige Anweisungen werden in Prozeduren und Funktionen zusammengefasst. In dieser Lerneinheit erfahren Sie,

- was Prozeduren von Funktionen unterscheidet,
- wie Sie mit Prozeduren und Funktionen arbeiten,
- wie Sie Werte an Prozeduren und Funktionen übergeben.

1.1 Prozedur

Eine Prozedur wird durch folgende Syntax deklariert:

```
[Public|Private ]Sub Prozedurname([ByVal|ByRef] Parameter1 [As Typ1][, ...])
...
   [Exit Sub]
...
End Sub
```

- Der Prozedurkopf beginnt mit dem Schlüsselwort `Sub`, gefolgt von dem Prozedurnamen und einem Klammerpaar `()`.
- Der Prozedurname muss sich an die Regeln für Bezeichner halten.
- Die Prozedur kann mit der Anweisung `Exit Sub` verlassen werden.
- Die Programmausführung wird mit der ersten Anweisung fortgesetzt, die dem Prozeduraufruf folgt.
- Die Deklaration der Prozedur wird mit `End Sub` beendet.
- Im Prozedurkopf kann `Public` oder `Private` vorangestellt werden, um den Gültigkeitsbereich der Prozedur festzulegen. Prozeduren mit den Schlüsseldort `Public` besitzen globale Gültigkeit, solche mit `Private` nur lokale. Wird keines dieser beiden Schlüsselwörter angegeben, dann handelt es sich standardmäßig um eine globale Prozedur.
- Optional kann die Prozedur mit Parameterübergabe aufgerufen werden. Dabei können die Parameter entweder als Wert (`ByVal`) oder als Verweis (`ByRef`) übergeben werden.

Beispiel: Die Prozedur `Aufrufer` ruft die Prozedur `Aendern` mit dem Parameter `intZahl` auf, wobei dieser Parameter zum einen als Wert (ByVal) und zum anderen als Verweis (ByRef) übergeben sind. Sonst wird nichts am Programmcode der beiden Prozeduren geändert.

```
Sub Aufrufer()
   Dim intZahl As Integer
   intZahl = 10
   Call Aendern(intZahl)
   If intZahl = 10 Then
      MsgBox "ByVal: intZahl wurde nicht geändert!"
   Else
      MsgBox "ByRef: intZahl wurde geändert!"
   End If
End Sub
```

Die beiden unterschiedlichen Parameterübergaben wirken sich wie folgt aus:

Parameter wird als **Wert** übergeben	Parameter wird als **Verweis** übergeben

`Sub Aendern(ByVal intZ As Integer)`
`  intZ = intZ + 10`
`End Sub`
	`Sub Aendern(ByRef intZ As Integer)`
	`  intZ = intZ + 10`
	`End Sub`

Wirkung: übergebener Wert wird *nicht* geändert	**Wirkung:** übergebener Wert wird geändert

Die Syntax für den Aufruf einer Prozedur ist sehr einfach: Schreiben Sie eine Anweisung, die nur aus dem Namen der aufzurufenden Prozedur besteht oder setzen Sie optional noch das Schlüsselwort `Call` davor. Optional enthält der Aufruf der Prozedur noch eine Liste der Werte, Konstanten oder Ausdrücke, die an diese übergeben werden sollen.

```
[Call ]Prozedurname (Ausdruck [, ...])
```

Eine Prozedur, die von einer anderen Prozedur aus aufgerufen wird, bezeichnet man im Allgemeinen als Subprozedur, Subroutine oder Unterprogramm.

Eine Prozedur, von der aus eine oder mehrere Subprozeduren aufgerufen werden und selbst von keinem Programm aus ausgerufen wird, wird als Hauptprogramm bezeichnet.

1.2 Funktion

Funktionen sind Prozeduren, die einen Wert zurückgeben. Sie können sogen. benutzerdefinierte Funktionen schreiben, die auf einem Tabellenblatt wie eine in Excel eingebaute Funktion über den Funktionsassistenten aufgerufen werden kann.

Allgemein wird eine Funktion definiert durch:

```
[Public|Private ]Function Name([ByVal|ByRef] Parameter1 [As Typ1][, ...]) [As Typ]
   ...
   [Exit Function]
   [Name = Rückgabewert]
End Function
```

- Die Übergabe der Parameter kann als Wert (`ByVal`) oder als Verweis (`ByRef`) erfolgen. Fehlt diese Angabe wird standardmäßig `ByRef` angenommen.
- Innerhalb der Klammer () wird von der aufrufenden Prozedur eine Parameterliste an die Funktion übergeben. Wird die Deklaration des Datentyps weggelassen, wird standardmäßig von Datentyp `Variant` ausgegangen.
- Mehrere übergebene Parameter werden jeweils durch ein Komma getrennt.

Eine Funktion kann mit einem optionalen Parameter aufgerufen werden. Die Syntax dafür lautet:

```
Optional Parameter [As Typ] [= Standardwert]
```

Da eine Funktion einen Wert zurückgibt, ist es sinnvoll, im Funktionsaufruf den Datentyp des Wertes von `Name` hinter der schießenden Klammer der Parameterliste festzulegen. Innerhalb der Funktion wird der Rückgabewert der Funktion durch die Zeile Name =… bestimmt. Der Funktionsaufruf erfolgt durch:

```
Variable = Name([Variable1], ...)
```

Beispiel: Die Prozedur `Osterdatum` soll für die Jahre 2009 bis 2011 jeweils das Datum des Ostersonntags bestimmen. Dafür wird die Funktion `Ostersonntag` eingesetzt. An diese Funktion wird das jeweilige Jahr als Inhalt der Variablen `intJahr` übergeben, und zwar im Rumpf einer Zählschleife von 2009 bis 2011. Die Funktion `Ostersonntag` gibt das berechnete Osterdatum an die aufrufende Prozedur zurück. In der Funktion selbst werden die in Excel eingebauten Datumsfunktionen `DateAdd`, `DateSerial` und `Weekday` eingesetzt. Dadurch wird die Komplexität der Berechnung des Osterdatums reduziert.

```
Sub Osterdatum()
   Dim intJahr As Integer
   Debug.Print "Jahr" & vbTab & "Ostersonntag"
   Debug.Print "----" & vbTab & "------------"
   For intJahr = 2009 To 2011
      Debug.Print intJahr & vbTab & Ostersonntag(intJahr)
   Next intJahr
End Sub
```

```
Public Function Ostersonntag(intYr As Integer) As Date
   Dim intDays As Integer
   Dim dtmTmp  As Date
   intDays = (19 * (intYr Mod 19) + 24) Mod 30
   dtmTmp = DateAdd("d", intDays, DateSerial(intYr, 3, 22))
   dtmTmp = DateAdd("d", (8 - Weekday(dtmTmp)) Mod 7, dtmTmp)
   Ostersonntag = DateSerial(intYr, Month(dtmTmp), Day(dtmTmp))
End Function
```

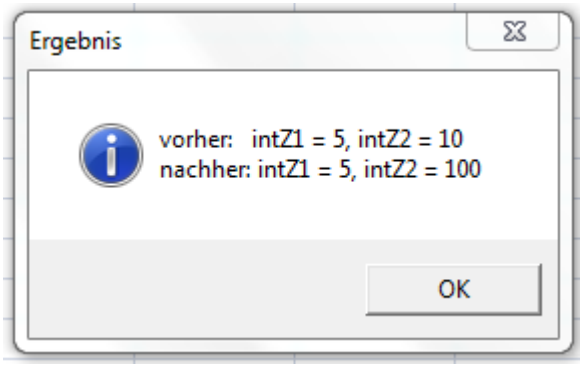
1.3 Schnellübersicht

Was bedeutet	
Prozedur (`Sub`)	Ein Programm, das kein Ergebnis zurückliefert.
Funktion (`Function`)	Ein Programm, das einen Wert zurück gibt.
Rückgabewert (engl. return value)	Ergebnis einer Funktion, das an das aufrufende Programm zurückgegeben wird.
Parameter (Argument)	An Prozeduren oder Funktionen übergebener Wert
Parameterübergabe als Verweis (`ByRef`)	Wenn ein übergebener Werte in einer Funktion geändert wird, ändert sich auch der entsprechende Wert im aufrufenden Programm.
Parameterübergabe als Wert (`ByVal`)	Änderungen des Übergabewertes in einer Funktion wirken sich im aufrufenden Programm nicht aus.

1.4 Übungen

1.4.1 Parameterübergabe

Die Prozedur `Treiber` ruft die Prozedur `Esel` mit zwei Parametern auf. Der erste Parameter (intZ1) mit dem Anfangswert 5 wird als Wert, der zweite Parameter (intZ2) mit dem Anfangswert 10 als Verweis übergeben. In der Prozedur `Esel` wird jeder übergebene Parameter quadriert. Nach dem Empfang der Rückmeldung sollen in der Prozedur `Treiber` die als Parameter übergebenen Variablen in einer `MsgBox` wie folgt angezeigt werden:



1.4.2 Funktionsaufruf

Schreiben Sie eine Prozedur `Advent`, die den Parameter *Jahreszahl* (`intJahr`) an die Funktion `VierterAdvent` übergibt. Berechnen Sie den 4. Advent für die Jahre 2009, 2010 und 2011. Der vierte Advent ist der letzte Sonntag vor dem 25.12. eines Jahres. Schreiben Sie die Ergebnisse in das Direktfenster. Zur Codierung der Funktion `VierterAdvent` verwenden Sie zweckmäßiger Weise die eingebauten Datumsfunktionen `DateSerial` und

`Weekday`. In der Online-Hilfe erhalten Sie nähere Informationen über diesen beiden vordefinierten Excel-Datumsfunktionen.

1.5 Lösungen

1.5.1 Parameterübergabe

```
Sub Treiber()
   Dim intZ1 As Integer
   Dim intZ2 As Integer
   Dim strVorher As String
   intZ1 = 5
   intZ2 = 10
   strVorher = "vorher:   intZ1 = " & intZ1 & ", intZ2 = " & intZ2
   Call Esel(intZ1, intZ2)
   MsgBox strVorher & vbNewLine & _
      "nachher: intZ1 = " & intZ1 & ", intZ2 = " & intZ2,
vbInformation, "Ergebnis"
End Sub
```

```
Sub Esel(ByVal intP1 As Integer, ByRef intP2 As Integer)
   intP1 = intP1 ^ 2
   intP2 = intP2 ^ 2
End Sub
```

1.5.2 Funktionsaufruf

```
Sub Advent()
   Dim intJahr As Integer
   Debug.Print "Jahr" & vbTab & "4. Advent"
   Debug.Print "----" & vbTab & "---------"
   For intJahr = 2009 To 2011
      Debug.Print intJahr & vbTab & VierterAdvent(intJahr)
   Next intJahr
End Sub
```

```
Public Function VierterAdvent(intYr As Integer) As Date
   ' Der vierte Advent ist der letzte Sonntag vor dem 25.12.
   Dim dtmChristmas As Date
   dtmChristmas = DateSerial(intYr, 12, 25)
   VierterAdvent = dtmChristmas - Weekday(dtmChristmas, vbMonday)
End Function
```

Workshop 4: VBA-Programmierung mit MS Excel

2. Objekte in Excel

In Excel sind mehr als 150 Objekte definiert. Das umfasst im Wesentlichen alles, was programmiert und kontrolliert werden kann. In dieser Lektion erfahren Sie,

- was Auflistungsobjekte sind,
- wie die Excel-Objekte hierarchisch aufgebaut sind,
- wie auf Objekte zugegriffen wird,
- wie mit Objektvariablen gearbeitet wird.

2.1 Eigenschaften, Methoden und Ereignisse

Jedes Excel-Objekt wird durch seine Eigenschaften (engl. *properties*) und Methoden (engl. *methods*) beschrieben. Ein wichtiges Objekt ist beispielsweise eine Arbeitsmappe. Sie hat einen bestimmten Namen, einen Autor oder sie ist durch ein Passwort geschützt. Sie besitzt standardmäßig drei Tabellenblätter. Sie kann auch Diagrammblätter enthalten.

Auf Excel-Objekte lassen sich Methoden anwenden. Methoden, die auf Arbeitsmappen (engl. `workbooks`) angewandt werden können, sind zum Beispiel: Aktivieren, schließen, speichern, drucken oder löschen.

Die folgende Tabelle beinhaltet einen kleinen Überblick über die am häufigsten benutzten Objekte in Excel:

Objekttyp[1]	**Kurzbedeutung**
`Application`	Die aktuelle Anwendung, praktisch Excel selbst
`Cells`	Zellen auf in einem Arbeitsblatt
`Chart`	Stellt ein Diagramm in einer Arbeitsmappe dar
`Charts`	Eine Auflistung der in einer Arbeitsmappe enthaltenen Diagrammblätter
`Dialog`	Stellt ein einzelnes Dialogfeld dar.
`Dialogs`	Eine Auflistung der in einer Arbeitsmappe enthaltenen Dialog-Objekte
`Module`	Ein einzelnes Modulblatt
`Modules`	Eine Liste der in einer Arbeitsmappe enthaltenen Modulblätter
`Names`	Gibt eine Auflistung aller arbeitsblattspezifischen Namen zurück.
`Range`	Umfasst eine einzelnen Zelle, eine ganze Zeile oder Spalte, einen Zellbereich, usw.
`Sheets`	Eine Auflistung der augenblicklich verfügbaren Tabellen-, Diagramm- und sonstige Blätter in der angegebenen Arbeitsmappe.
`Window`	Ein einzelnes Fenster
`Windows`	Eine Auflistung verschiedener augenblicklich geöffneter Fenster
`Workbook`	Eine einzelne Arbeitsmappe
`Workbooks`	Eine Auflistung aller augenblicklich geöffneten Arbeitsmappen
`Worksheet`	Stellt ein einzelnes Arbeitsblatt dar.
`Worksheets`	Eine Auflistung aller in einer Arbeitsmappe enthaltenen Tabellenblätter
[1]) Benutzen Sie bitte die Online-Hilfe, um Näheres über die aufgezählten Objekttypen zu erfahren.	

Tab. 1: Häufig benutzte Objekte in Excel

2.2 Auflistungen von Objekten

Mehrere Objekte desselben Typs werden zu Auflistungen (engl. collections) zusammengefasst. So werden beispielsweise die drei `Worksheet`-Objekte einer Arbeitsmappe als Auflistung *`Worksheets`* verwaltet. Alle geöffneten `Workbook`-Objekte werden in der Auflistung `Workbooks`, alle Diagramme einer Arbeitsmappe als `Charts` und alle Namen für Zellen und Zellbereiche einer Arbeitsmappe in der `Names`-Auflistung zusammengefasst.

Um eine Auflistung zu kennzeichnen, wird in der Regel ein „**s**“ an den Namen des Objekts angehängt. Eine Ausnahme davon ist das `Range`-Objekt (siehe Tab. 1).

Auch Auflistungen selbst sind Objekte und können Eigenschaften und Methoden aufweisen. Diese können sich von denen der einzelnen Objekte in der Auflistungen durchaus unterscheiden.

2.3 Objekte benennen

Um Objekte zu bearbeiten, müssen sie genau benannt (d. h. *referenziert*) werden. Beispiel: Sie haben sowohl in der Arbeitsmappe `Rechnung` als auch in der Arbeitsmappe `Auftrag` jeweils ein Tabellenblatt namens `TABELLE1`. Eine eindeutige Referenzierung der Zelle `B1` in `Tabelle1` der Arbeitsmappe `Rechnung` ist nur unter Angabe der Arbeitsmappe möglich.

```
Workbooks("Rechnung").Worksheets("Tabelle1").Range("B1").Value = "Datum"
```

Wird allerdings nur mit einer einzigen Arbeitsmappe gearbeitet, dann ist die Referenzierung der Arbeitsmappe überflüssig. Obige Anweisung verkürzt sich dann zu:

```
Worksheets("Tabelle1").Range("B1").Value = "Datum"
```

Ist `Tabelle1` bereits das aktive Tabellenblatt, dann genügt sogar die Anweisung

```
Range("B1").Value = "Datum"
```

um die richtige Zelle im richtigen Tabellenblatt zu adressieren.

2.4 Benennung von Objekten in einer Auflistung

Im Folgenden wird das korrekte Ansprechen von Objekten in einer Auflistung demonstriert. Betrachten Sie dazu bitte die in Abb. 1 gezeigten Tabellenblätter in einer Arbeitsmappe:

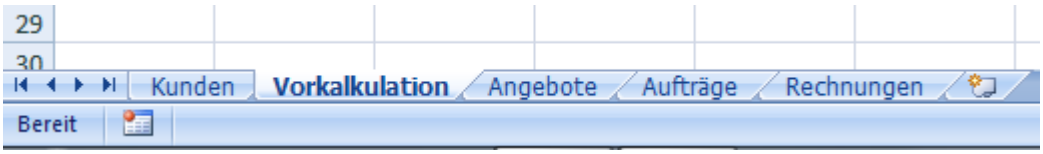


Abb. 1: Benannte Arbeitsblätter in einer Arbeitsmappe

Das zweite Arbeitsblatt der Auflistung `Worksheets` soll ausgedruckt werden. Zum Ansprechen eines bestimmten Objekts einer Auflistung gibt es zwei Möglichkeiten:

- Referenzierung mit seinem Namen (hier `Vorkalkulation`, siehe Abb. 1)
- Referenzierung mit seinem Index (hier `2`, siehe Abb. 1)

Tab. 2 enthält eine Gegenüberstellung der beiden Möglichkeiten.

Tabellenblatt ansprechen mit …	
Namen	**Index**

`Sub Solution1()`
`  Worksheets("Vorkalkulation").PrintOut`
`End Sub`
	`Sub Solution2()`
	`  Worksheets(2).PrintOut`
	`End Sub`

Tab. 2: Gegenüberstellung der Referenzierung mit Namen bzw. Index

Der Index nummeriert alle Tabellenblätter vom Anfang bis zum Ende fortlaufend durch,

- unabhängig von der angezeigten Nummer des Tabellenblatts auf dem Register und
- unabhängig von der ursprünglichen Position des Tabellenblattes.

Werden die Tabellenblätter vertauscht (siehe Abb. 2), dann wird mit

```
Worksheets(2).PrintOut
```

das zweite Tabellenblatt gedruckt mit dem Namen `TABELLE1`.

Abb. 2: Bedeutung der relativen Position beim Zugriff auf ein Tabellenblatt

Für den Ausdruck aller verwendeter Tabellenblätter einer Arbeitsmappe wird einfach die Auflistung ohne Argumente verwendet:

```
Worksheets.PrintOut
```

2.5 Beispiele zum Ansprechen von Objekten

Die Prozedur `ObjekteReferenzieren` ermittelt die Zahl der geöffneten Arbeitsmappen und den Wert der Zelle B1 im zweiten Arbeitsblatt der aktiven Arbeitsmappe:

```
Sub ObjekteReferenzieren()
   Dim intMappen As Integer
   Dim strText  As String
   intMappen = Workbooks.Count
   MsgBox "Zahl der geöffneten Arbeitsmappen: " & intMappen, vbInformation
   strText = ActiveWorkbook.Worksheets(2).Cells(1, 2).Text
   MsgBox "Inhalt der Zelle B1: " & strText
End Sub
```

Mit Hilfe der `With`-Anweisung kann mehrmals auf ein Objekt zugegriffen werden, ohne dass jedes Mal der Name des relevanten Objekts erneut referenziert werden muss. Die Syntax der `With`-Anweisung lautet:

```
With Objekt
   .Anweisung
End With
```

`With`-Anweisungen können geschachtet werden. Beispiel:

```
With Objekt1
      .Anweisung1
      With Objekt2
            .Anweisung2
      End With
      .Anweisung3
End With
```

In den beiden Prozeduren `SchriftAendern1` und `SchriftAendern2` werden einige Eigenschaft der Zelle B1 im zweiten Arbeitsblatt der aktiven Arbeitsmappe geändert. Das geschieht einmal mit und einmal ohne die `With`-Anweisung:

```
Sub SchriftAendern1()
   ActiveWorkbook.Worksheets(2).Cells(1, 2).Font.Name = "Arial"
   ActiveWorkbook.Worksheets(2).Cells(1, 2).Font.Size = 20
   ActiveWorkbook.Worksheets(2).Cells(1, 2).Font.ColorIndex = 5
   MsgBox "Fertig", vbExclamation
End Sub
```

```
Sub SchriftAendern2()
   With ActiveWorkbook.Worksheets(2).Cells(1, 2).Font
      .Name = "Arial"
      .Size = 20
      .ColorIndex = 5
   End With
   MsgBox "Fertig", vbExclamation
End Sub
```

Das Ergebnis ist gleich, aber die zweite Lösung ist kürzer und übersichtlicher!

2.6 Eigenschaften von Objekten

Eigenschaften (engl. properties) beschreiben Objekte. Eigenschaften von Objekten können abgefragt und häufig auch geändert werden. Einige Eigenschaften können nur gelesen, aber nicht geändert werden.

Die folgende Prozedur `SpeicherStatus` prüft, ob die aktive Arbeitsmappe bereits gespeichert wurde und gibt eine entsprechende Meldung aus:

```
Sub SpeicherStatus()
   Dim bolSaved As Boolean
   Dim strMsg As String
   bolSaved = ActiveWorkbook.Saved
   If bolSaved Then
      strMsg = "Die Arbeitmappe wurde gespeichert."
   Else
      strMsg = "Die Arbeitsmappe wurde noch nicht gespeichert."
   End If
   MsgBox strMsg, vbInformation, "Speicherstatus"
End Sub
```

Im Gegensatz zu Methoden werden für Eigenschaften *keine* Argumente übergeben. In der folgenden Anweisung wird nach einem Punkt der Wert der neuen Eigenschaft gesetzt:

```
Worksheets("Tabelle1").Range("B1").HorizontalAlignment = xlCenter
```

Mit dieser Anweisung wird der Wert in Zelle B1 horizontal zentriert. Soll nicht eine Eigenschaft gesetzt, sondern der Wert der Eigenschaft ermittelt werden, ist der Aufruf ähnlich. Angenommen, Sie möchten die aktuelle Schriftgröße in der Zelle B1 herausfinden:

```
Sub SchriftGroesse()
   Dim dblSchrift As Double
   dblSchrift = Worksheets("Tabelle1").Range("B1").Font.Size
   Debug.Print "Schriftgröße: " & dblSchrift
End Sub
```

2.7 Methoden von Objekten

Als Methode wird ein Vorgang bzw. eine Tätigkeit bezeichnet, die mit einem Excel-Objekt ausgeführt werden kann. Um eine Methode anzuwenden, wird ein Bezug auf das betreffende Objekt benötigt, gefolgt von einem Punkt und dem Namen der Methode, also allgemein so:

```
Objekt.Methode
```

Im folgenden Beispiel wird der Tabellenbereich A1:B2 ausgewählt und kopiert:

```
Sub BereichKopieren()
   With ActiveSheet
      MsgBox "Das aktulle Tabellenblatt heißt " & .Name
      .Range("A1:B2").Select
   End With
   Selection.Copy
End Sub
```

Nochmals, eine Methode wird immer auf das davor stehende Objekt angewendet und mit einen Punkt verbunden, siehe oben: **`Selection.Copy.`**

Viele Methoden übergeben Argumente, um ihre Ausführung zu spezifizieren. Beispielsweise kann mit der Methode `PrintOut` das Argument `Copies:=1` übergeben werden:

```
ActiveWindow.SelectedSheets.PrintOut Copies:=1
```

Die allgemeine Form der `PrintOut`-Methode wird angezeigt, wenn das Schlüsselwort `PrintOut` im Quellcode angeklickt und dann die Funktionstaste F1 gedrückt wird (s. Abb. 3)

Excel-Entwicklerreferenz

Worksheets.PrintOut-Methode

Druckt das Objekt.

Syntax

Ausdruck.**PrintOut**(***From, To, Copies, Preview, ActivePrinter, PrintToFile, Collate, PrToFileName, IgnorePrintAreas***)

Ausdruck Eine Variable, die ein **Worksheets**-Objekt darstellt.

Parameter

Name	Erforderlich/Optional	Datentyp	Beschreibung
From	Optional	**Variant**	Die Nummer der ersten Seite, die gedruckt werden soll. Wenn Sie dieses Argument nicht angeben, wird von der ersten Seite an gedruckt.
To	Optional	**Variant**	Die Nummer der letzten Seite, die gedruckt werden soll. Wenn Sie dieses Argument nicht angeben, wird bis zur letzten Seite gedruckt.
Copies	Optional	**Variant**	Die Anzahl der zu druckenden Kopien. Wenn Sie dieses Argument nicht angeben, wird eine Kopie gedruckt.
Preview	Optional	**Variant**	Mit dem Wert **True** wird in Microsoft Excel die Seitenansicht aufgerufen, bevor das Objekt gedruckt wird. Mit dem Wert **False** oder bei nicht angegebenem Wert wird das Objekt sofort gedruckt.
ActivePrinter	Optional	**Variant**	Legt den Namen des aktiven Druckers fest.
PrintToFile	Optional	**Variant**	Mit **True** erfolgt die Ausgabe in eine Datei. Wenn *PrToFileName* nicht angegeben ist, werden die Benutzer von Microsoft Excel zur Eingabe des Namens der Ausgabedatei aufgefordert.
Collate	Optional	**Variant**	Mit **True** werden Mehrfachkopien sortiert.
PrToFileName	Optional	**Variant**	Wenn *PrintToFile* auf **True** festgelegt ist, gibt dieses Argument den Namen der Datei an, in die Sie drucken möchten.
IgnorePrintAreas	Optional	**Variant**	Mit **True** werden Druckbereiche ignoriert und das gesamte Objekt gedruckt.

Rückgabewert

Variant

Abb. 3: `Worksheets.PrintOut`-Methode in der Online-Hilfe

In VBA gibt es zwei Möglichkeiten, Argumente an eine Methode zu übergeben:

- über ihre Reihenfolge
- über ihren Namen

Die zuerst genannte Möglichkeit spart gewöhnlich Schreibarbeit, hat aber den Nachteil, dass die in der Syntax vorgeschriebene Reihenfolge (s. Abb. 3) strikt eingehalten werden muss. Eine falsche Reihenfolge erzeugt unter Umständen nicht beabsichtige Wirkungen.

Beispiel:

```
ActiveWindow.SelectedSheets.PrintOut 1, 2, 1, , , , True
```

Die zweite Möglichkeit beruht auf der Übergabe von benannten Argumenten. Fortsetzung des Beispiels:

```
ActiveWindow.SelectedSheets.PrintOut From:=1, To:=2, Copies:=1, Collate:=True
```

Da hierbei die einzelnen Argumente mit ihrem Namen übergeben werden, spielt ihre Reihenfolge keine Rolle.

Beide Möglichkeiten führen also, richtige Anwendung vorausgesetzt, zum gleichen Ergebnis.

2.8. Ereignisse von Objekten

Jede von Benutzer ausgelöste Aktion (etwa ein Tastenanschlag oder ein Mausklick) löst *in Windows* ein Ereignis aus. *Windows* wertet das Ereignis aus und gibt ggf. die Nachricht an das entsprechende Anwendungsprogramm weiter.

In Excel können Programme geschrieben werden, die geplant auf Ereignisse der folgenden Art reagieren.

- Mausklick auf ein Symbol, ein Steuerelement oder einen Menüpunkt,
- Tastatureingaben,
- Öffnen oder Schließen einer Arbeitsmappe oder eines Fensters.

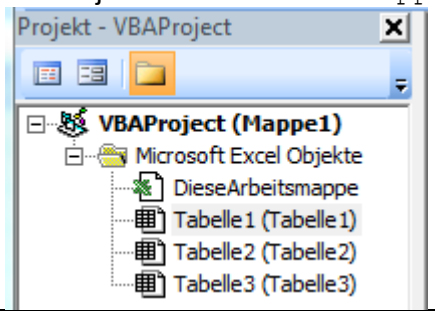
Die entsprechenden Programme werden *Ereignisprozeduren* genannt.

Wird zum Beispiel in Excel auf eine Befehlsschaltfläche namens `Vorkalkulation` geklickt, wird ein sogen. `Click`-Ereignis ausgelöst. Entsprechend heißt die dazugehörige die Ereignisprozedur `Vorkalkulation_Click()`.

Um Ereignisprozeduren den syntaktischen Regeln gemäß zu benennen, werden das Objekt und das zugehörige Ereignis durch einen Unterstrich (_) hintereinander geschrieben, wie die folgenden Beispiele verdeutlichen:

```
Schaltfläche_Click()
Arbeitsmappe_Open()
Tabellenblatt_Activate()
Diagramm_Change()
```

Das Erstellen einer Ereignisprozedur für die Arbeitsmappe `DieseArbeitsmappe`, die beim Ereignis `Open` ausgelöst werden soll, erfordert folgende Arbeitsschritte (siehe Tab. 3)

❶	Klicken Sie in Projekt-Explorer doppelt auf das Objekt `DieseArbeitsmappe` 	Für dieses Objekt wird das Code-Fenster geöffnet.
❷	Geben Sie `Option Explicit` ein, falls noch nicht vorhanden.	
❸	Wählen Sie im linken Listenfeld des Code-Fensters den Eintrag `Workbook`.	Es wird automatisch die leere Prozedur `Workbook_Open` für das Ereignis `Open` erstellt.
❹	Fügen Sie an der Cursorposition der eingefügten Ereignisprozedur die gewünschten Anweisungen ein.	

Tab. 3: Arbeitsschritte zur Erstellung einer leeren `Workbook_Open` Ereignisprozedur

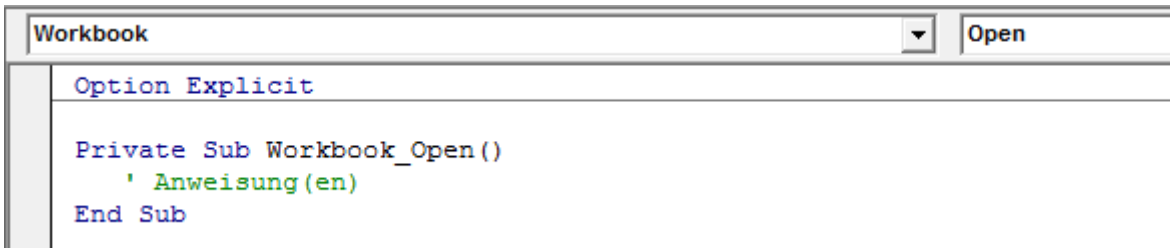


Abb. 4: Die leere Ereignisprozedur `Workbook_Open`

Die Anweisungen in der Ereignisprozedur `Workbook_Open` könnten zum Beispiele so lauten:

```
Private Sub Workbook_Open()
   Dim varEingabe As Variant
   Do ' fußgesteuerte Schleife
      varEingabe = InputBox("Geben Sie bitte Ihren Namen ein: ", "Willkommen")
      If varEingabe = vbNullString Then
         MsgBox "Name ist erforderlich!", vbExclamation, "Workbook Open"
      End If
   Loop Until varEingabe <> vbNullString
   ActiveWorkbook.Worksheets("Tabelle1").Cells(1, 1) = varEingabe
End Sub
```

Diese Ereignisprozedur trägt in Zelle A1 des Tabellenblatts `Tabelle1` den über `InputBox` angeforderten Namen ein.

Neben `Workbook_Open` ist `Workbook_BeforeClose` ein weiteres wichtiges Ereignis. Es tritt automatisch ein, sobald die Arbeitsmappe geschlossen wird. Aber es gibt noch weitaus mehr Ereignisse für Arbeitsmappen. Ausgewählte Ereignisse für Arbeitsmappen enthält Tab. 4:

Ereignis (`WorkBook_...`)	Beschreibung (Tritt ein, …
`Activate`	sobald eine Arbeitsmappe aktiviert wird.
`BeforePrint`	vor dem Druck einer Arbeitsmappe oder einzelner Tabellenblätter.
`BeforeSave`	vor dem Speichern.
`Deactivate`	wenn eine Arbeitsmappe deaktiviert wird.
`NewSheet`	wenn ein neues Tabellenblatt in die Arbeitsmappe eingefügt wird.
`SheetActivate`	wenn ein beliebiges Tabellen- oder Diagrammblatt in der Arbeitsmappe aktiviert wird.
`SheetBeforeDoubleClick`	wenn mit der linken Maustaste ein Doppelklick auf eine beliebige Stelle des Arbeitsblatts ausgeführt wird.
`SheetBeforeRightClick`	wenn mit der rechten Maustaste ein Klick auf eine beliebige Stelle des Arbeitsblatts ausgeführt wird.
`SheetCalculate`	bei Neuberechnung eine beliebigen Tabellenblatts.
`SheetDeactivate`	sobald ein beliebiges Tabellenblatt verlassen wird.
`SheetChange`	bei Änderung eines beliebigen Tabellenblatts.
`SheetSelectionChange`	wenn sich die Markierung auf einem Tabellenblatt ändert
`WindowActivate`	wenn ein Fenster der Arbeitsmappe aktiviert wird.
`WindowDeactivate`	wenn ein Fenster der Arbeitsmappe deaktiviert wird.
`WindowResize`	wenn ein die Größe eines Fenster der Arbeitsmappe geändert wird.

Tab. 4: Ausgewählte Ereignisse für Arbeitsmappen

Ereignisprozeduren für Tabellenblätter oder Diagrammblätter können auf ganz ähnliche Weise erstellet werden.

Einige wichtige Ereignisse für Tabellenblätter enthält Tab. 5:

Ereignis (`WorkSheet_...`)	Beschreibung (Tritt ein, …
`Activate`	sobald eine Tabellenblatt aktiviert wird.
`Change`	wenn sich der Wert einer Zelle ändert.
`Calculate`	wenn ein Tabellenblatt neu berechnet wird.
`Deactivate`	wenn ein Tabellenblatt deaktiviert (z. B. verlassen) wird.
`SelectionChange`	wenn sich die Markierung auf einemTabellenblatt ändert.

Tab. 5: Ausgewählte Ereignisse für Tabellenblätter

Die Ereignisse von Arbeitsmappen und Tabellenblättern bilden die Grundlage für die ereignisorientierte Programmierung mit VBA in Excel.

2.9 Hilfe zu Objekten

Hilfe zu bestimmten Objekten, die sie in einer Prozedur oder Funktion verwendet haben, erhalten Sie am einfachsten, indem Sie in den Namen des Objekts klicken und mit der Funktionstaste F1 die dazugehörige Online-Hilfe aufrufen.

Außerdem besteht die Möglichkeit, vom VBA-Editor aus den **Objektkatalog** aufzurufen. Um ihn zu aktivieren, verwenden Sie die Funktionstaste F2.

In Excel hat jedes Auflistungsobjekt denselben Namen wie die Eigenschaft, die eine Referenz auf das entsprechende Objekt liefert (siehe Abb. 4). Zum `Workbooks`-Objekt ❶ gibt es die Eigenschaft `Workbooks` ❷, die eine Referenz auf das Workbooks-Objekt ❸ liefert.

In Abb. 4 bezeichnet die Spaltenüberschrift *Klasse* eine abstrakte Beschreibung für Objekte desselben Typs. Eine Klasse (engl. class) legt Methoden und Eigenschaften fest, die jedes Objekt der Klasse aufweisen soll.

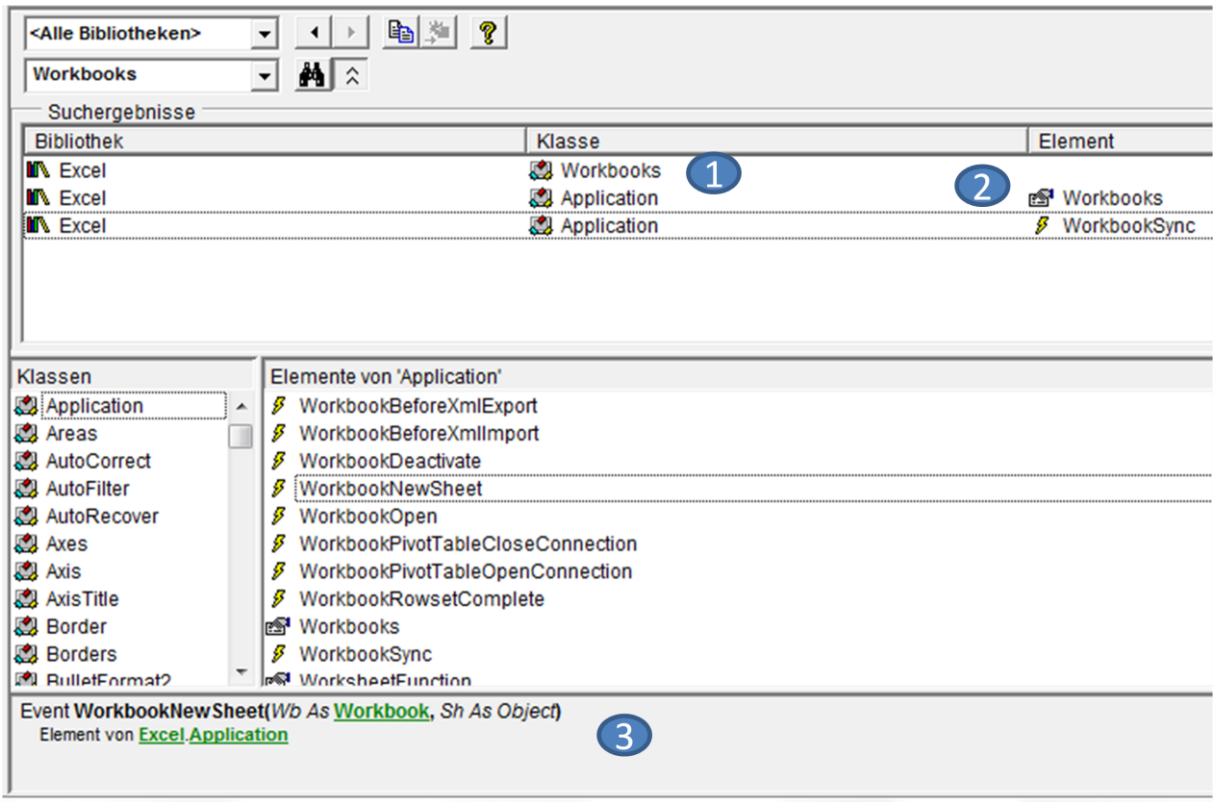


Abb. 4: Informationen zum Objekt `Workbooks` im Objektkatalog

2.10 Schnellübersicht

Was besagt …	
Klasse (bzw. Objekttyp)	Bauanleitung für Objekte desselben Typs
Objekt	Instanz (Ausprägung) einer Klasse
Eigenschaft	Jedes Objekt besitzt bestimmte Eigenschaften. Damit kann das Objekt angesprochen oder verändert werden.
Methode	Methoden stellen Prozeduren dar, die auf bestimmte Objekte spezialisiert sind und mit deren Daten etwas bewirken.
Auflistung	Zusammenfassung der Objekte eines bestimmten Typs
Ereignisprozedur	Führt benutzerdefinierte Anweisungen aus, die an ein Ereignis gebunden sind. Tritt das Ereignis ein, wird sie aufgerufen und die darin enthaltenen Ausweisungen ausgeführt.

Sie wollen …	
ein Objekt ansprechen	Geben Sie eine Objekthierarchie an, deren Objekte ggf. durch Punkte getrennt sind. Typisches Beispiel: `ActiveWorkbook.Worksheets(1).Range("A1").Font.Bold`

ein Objekt in einer Auflistung ansprechen	Geben Sie entweder den Namen des Objekts oder dessen Index in Klammern sein: Beispiel:
	`Dim wrkBk As Excel.Workbook`
	`Set wrkBk = Workbooks(1)`
	oder:
	`Set wrkBk = Workbooks("Mappe1")`

auf alle Objekte einer Auflistung zugreifen	Verwenden Sie den Namen der Auflistung ohne Klammern, z. B. `Workbooks`
eine Objektvariable deklarieren	`Dim Variablenname As [New] Objekttyp`
einen Verweis auf eine Objektvariable setzen	`Set Variablenname As [New] Objekt`
den Verweis auf eine Objektvariable löschen	`Set Variablenname = Nothing`
mehrmals auf ein Objekt zugreifen	`With Objekt ...Anweisung ...End With`
eine Ereignisprozedur erstellen	1. Code-Fenster für das betreffende Objekt öffnen 2. im linken Listenfeld das Objekt auswählen 3. im rechten Listenfeld das Ereignis auswählen

2.11 Übungen

1. Wie referenzieren Sie die Tabelle `Tabelle1` in einer aktiven Arbeitsmappe?
2. Wie machen Sie die Tabelle `Tabelle2` einer zwar geöffneten, aber zurzeit nicht aktiven Mappe `Test.xlsx` zur aktiven Tabelle?
3. Wie geben Sie den Namen des folgenden Tabellenblatts hinter dem zurzeit aktiven Tabellenblatt in einer `MsgBox` aus?
4. Weisen Sie der Zelle C5 in der momentan nicht aktiven Tabelle `Tabelle1` der Mappe `Test.xlsx` mit Hilfe der `With`-Anweisung die Schrittgröße 12 und das Schriftattribut kursiv zu!
5. Lösen Sie Aufgabe 4 unter Verwendung einer Objektvariablen namens `objTemp`.
6. Schreiben Sie eine Ereignisprozedur (ggf. mit Fehlerbehandlung), die das Tabellenblatt `Tabelle3` beim Öffnen der Arbeitsmappe `Test.xlsm` aktiviert:

2.12 Lösungen

❶
```
Worksheets("Tabelle1").Name
```

❷
```
Workbooks("Test.xlsx").Worksheets("Tabelle2").Activate
```

❸
```
MsgBox ActiveSheet.Next.Name
```

❹
```
With Workbooks("Test.xlsx").Worksheets("Tabelle1").Cells(5, 3).Font
   .Size = 12
   .Italic = True
 End With
```

❺
```
Dim objTemp As Object    ' Objektvariable deklarieren
Set objTemp = Workbooks("Test.xlsx").Worksheets("Tabelle1").Cells(5, 3).Font
With objTemp             ' Mit der Objektvariablen ...
   .Size = 12            ' Schriftgröße bestimmen
   .Italic = True        ' Kursiv einschalten
End With
Set objTemp = Nothing    ' Objektvariable freigeben
```

❻

1. Drücken Sie die Tastenkombination Alt + F11, um in die Entwicklungsumgebung zu kommen.
2. Klicken Sie im Projekt-Explorer den Eintrag DIESE ARBEITSMAPPE doppelt an.
3. Klicken Sie im linken Listenfeld des Codefensters den Eintrag WORKBOOK an. Excel stellt Ihnen standardmäßig eine vorgefertigte Ereignisprozedur zu Verfügung.
4. Ergänzen Sie diese Prozedur mit den erforderlichen Anweisungen, ggf. mit Fehlerbehandlung.

```
Private Sub Workbook_Open()
   Const conTabName As String = "Tabelle3"
   On Error GoTo Fehler
   Sheets(conTabName).Activate
   Exit Sub
Fehler:
   MsgBox "Tabellenblatt" & conTabName & " nicht gefunden", _
      vbCritical, "Workbook Open"
End Sub
```