Workshop 2: VBA-Programmierung mit MS Excel

1 Die Entwicklungsumgebung kennen lernen

In dieser Lektion lernen Sie

- den Projekt-Explorer
- das Eigenschaftenfenster
- den Objektkatalog und
- einige weitere Fenster der Entwicklungsumgebung

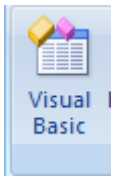
kennen.

Workshop 2: VBA-Programmierung mit MS Excel

Beim ersten Betrachten erscheint die sogen. Entwicklungsumgebung von Visual Basic for Applications (VBA) mit den verschiedenen Ordnern und den vielen Eigenschaften als ziemlich verwirrend. Deshalb sollen in dieser Lektion die einzelnen Bestandteile der Entwicklungsumgebung vorgestellt werden, bevor auf die Grundlagen der VBA-Programmierung eingegangen wird.

1.1 In die Entwicklungsumgebung wechseln

Wie Sie bereits gelernt haben, können Sie mit der Tastenkombination **Alt + F11** in die Entwicklungsumgebung wechseln. Alternativ dazu können Sie mittels Menüpunkt ENTWICKLERTOOLS und der Schaltfläche VISUAL BASIC die Entwicklungsumgebung aufrufen.



1.2 Der Projekt-Explorer

In der linken oberen Ecke sehen Sie den Projekt-Explorer, der alle geöffneten Arbeitsmappen sowie die darin befindlichen Tabellenblätter anzeigt. Standardmäßig besteht ein VBAProject aus einem Ordner (MICROSOFT EXCEL OBJEKTE), das 4 Elemente umfasst:

- ein Symbol für die Arbeitsmappe selbst und
- drei weitere für jeweils ein Tabellenblatt der Arbeitsmappe ((siehe Abb. 1).

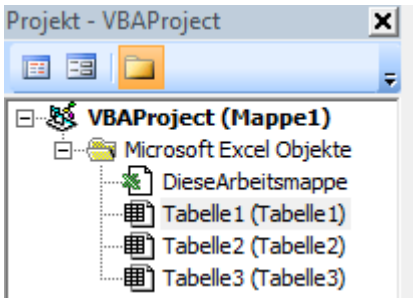


Abb. 1: Standard-Elemente des Projekt-Explorers

Möchten Sie ein neues Makro oder eine neue Prozedur erstellen, doppelklicken Sie auf das jeweilige Element. Damit öffnen Sie das zugehörige Code-Fenster.

Der Name der Excel-Dateien erscheint direkt hinter der Bezeichnung VBAProject. Haben Sie in Excel bereits die einzelnen Tabellenblätter benannt, so finden Sie diese benutzerdefinierten Namen in Klammern hinter TABELLE1, TABELL2, usw.

Sowohl das Projekt selbst als auch seine Elemente können Sie nach Belieben umbenennen. Dazu benötigen Sie das Eigenschaftsfenster (siehe unten).

1.3 Das Eigenschaftenfenster

Das Eigenschaftenfenster wird mit dem Befehl ANSICHT -> EIGENSCHAFTENFENSTER oder die Funktionstaste **F4** geöffnet, falls es noch nicht aktiviert sein sollte. Verschiedene Merkmale zum aktuell ausgewählten Element werden dargestellt (siehe Abb. 2)

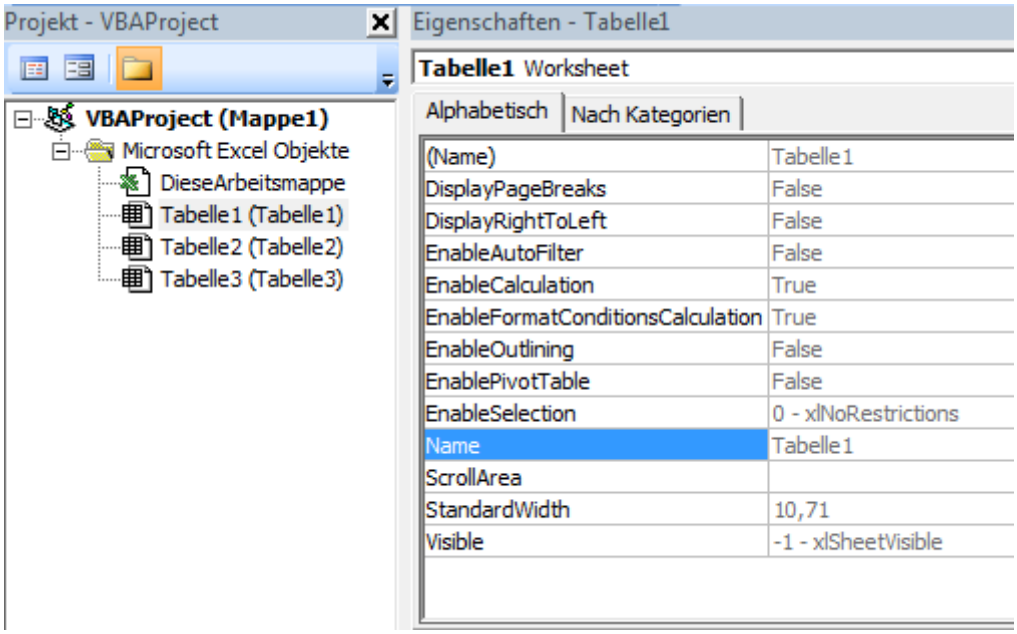


Abb. 2: Eigenschaftenfenster zu TABELLE1

Der bisherige Name (hier Tabelle1) seht hinter (NAME). Überscheiben Sie ihn nach Belieben.

1.4 Sonstige Elemente im Projekt-Explorer

Nach dem Erstellen eines Mitschreibmakros in Ihrer Excel-Datei, erscheint neben dem Ordner MIRROSOFT EXCEL OBJEKTE ein weiterer, nämlich MODULE. Zudem lassen sich u. a. auch eigene Dialogfelder – oft auch Formulare genannt - im Projekt-Explorer erstellen.

Ist der Ordner MODULE bereits angelegt, erhält er mindestens 1 Element, das standardmäßig als MODUL1 bezeichnet wird. Möchten Sie eine neues Modul anlegen, vollziehen Sie einen der folgenden Arbeitsschritte:

- Verwenden Sie die Befehlsfolge EINFÜGEN MODUL
- Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf das entsprechende Projekt und wählen Sie im Kontextmenü EINFÜGEN MODUL aus.

Auf ähnliche Weise lassen sich Formulare (USERFOM) in den Projekt-Explorer aufnehmen. Darüber später mehr.

Möchten Sie ein angelegtes MODUL oder eine vorhandene USERFORM löschen, klicken Sie das entsprechende Element mit der rechten Maustaste an und wählen im Kontextmenü die Option ENTFERNEN VON … aus. Auf die dann folgende Frage, ob Sie das Element exportieren möchten, antworten Sie mit NEIN.

1.5 Der Objektkatalog

Im Objektkatalog finden Sie alle verfügbaren Objekte mit ihren zugeordneten Eigenschaften und Methoden. Dazu muss man wissen, dass alle Komponenten von Excel als Objekte angesehen werden. Somit sind Arbeitsmappen und Tabellenblätter ebenso Objekte wie eine Schaltfläche oder ein Diagramm. Sie öffnen den Objektkatalog über die gleichnamige Schaltfläche, also den Befehl ANSICHT -> OBJEKTKATALOG oder mit der Funktionstaste F2.

1.6 Weitere Fenster

Das Direkt-, das Lokal- sowie das Überwachungsfenster dienen zur Fehlersuch in VBA-Programmen. Alle drei lassen sich über den Menüpunkt ANSICHT anzeigen. Bezüglich VBA-Programmierung ist das Direktfenster u. a. aus folgenden Gründen sehr nützlich:

- Dort lässt sich der Inhalt von Variablen während der Programmausführung abfragen.
- Einzelne Programmzeilen können dort eingegeben und getestet werden.
- Mittels `Debug.Print` können dort Programmausgaben aufgelistet werden.

2 Grundlagen der Programmierung mit VBA

Sie werden in dieser Lektion lernen,

- wie ein VBA-Programm erstellt wird,
- wie Variablen in Programmen deklariert und verwendet werden,
- was Datentypen sind,
- was bedingte Anweisungen im Programm bewirken,
- wie unterschiedliche Arten von Schleifen programmiert werden.

2.1 VBA-Programm erstellen

In VBA wird als Prozedur bezeichnet, was bislang (Mitschreib-) Makro genannt wurde. Prozeduren werden durch folgende Schlüsselwörter eingerahmt:

```
Sub Name
      Anweisungsblock innerhalb der Prozedur
End Sub
```

Um eine Prozedur zu erstellen, gehen Sie so vor:

- Benutzen Sie die Tastenkombination Alt + F11, um das VBA-Fenster zu aktivieren oder
- Doppelklicken Sie im Projekt-Explorer auf TABELLE1 (TABELLE1).

Nach dem Erscheinen des VBA-Editor geben Sie folgenden Code ein:

```
Sub MeineErsteProzedur()
   MsgBox "Die Prozedur funktioniert!"
End Sub
```

`MsgBox` ist ein Schlüsselwort. Schlüsselwörter bezeichnen von Excel reservierte Wörter. Wenn Sie hinter `MsgBox` ein Leerzeichen eingeben, erscheint die sogen. QuickInfo für diesen Befehl. Darin wird der jeweilige Aufbau eines Befehls angezeigt.

Zum Start von `MeineErsteProzedur` betätigen Sie einfach die Funktionstaste F5 oder klicken Sie auf die Schaltfläche SUB/USERFOM AUSFÜHREN in der Symbolleiste DEBUGGEN. (siehe Abb. 3).

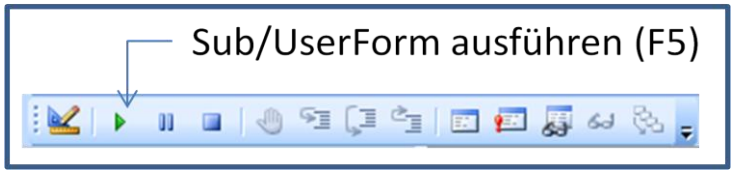


Abb. 3: Symbolleiste DEBUGGEN

Auf dem Bildschirm erscheint dann die definierte Erfolgsmeldung:

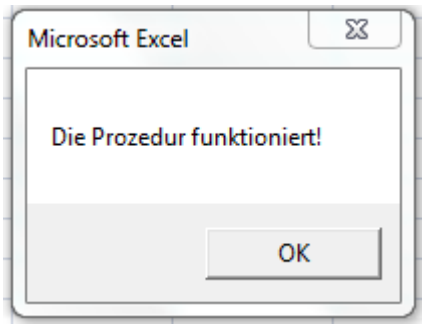


Abb. 4: Ausgabe der Meldung

Die Meldung verschwindet erst wieder, wenn die OK Schaltfläche angeklickt wird.

Weitere Gestaltungs- und Verwendungsmöglichkeiten einer `MsgBox` entnehmen Sie bitte der Online-Hilfe.

Die Prozedur `MeineErsteProzedur` soll jetzt noch durch das Einfügen einer `InputBox` etwas flexibler gestaltet werden. Eine `InputBox` erwartet eine Eingabe des Benutzers, die einer Variablen vom Typ Variant übergeben wird. Für die in Abb. 5 gezeigte `InputBox` wurde die Titelzeile, der Beschreibungstext und der Standardtext im Eingabefeld in der Prozedur definiert. Den Standardtext kann der Benutzer beliebig überschreiben.

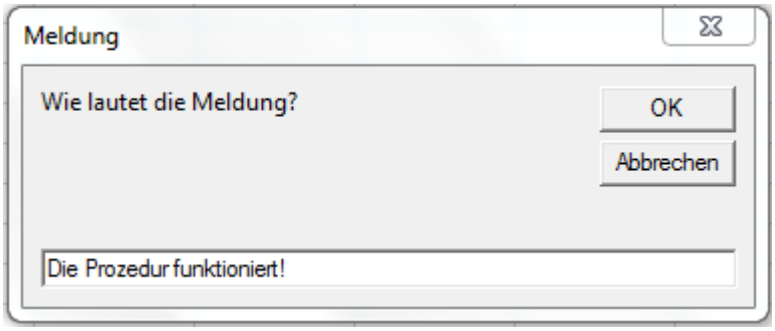


Abb. 5: Inputbox mit vordefinierter Eingabe

Die geänderte und kommentierte Prozedur `MeineErsteProzedur` sieht dementsprechend wie folgt aus:

```
Sub MeineErsteProzedur()
   ' Variable vom Typ Variant deklarieren
   Dim varMsg As Variant
   ' Meldung erfassen
   varMsg = InputBox("Wie lautet die Meldung?", "Meldung", _
      "Die Prozedur funktioniert!")
   ' Meldung ausgeben
   MsgBox varMsg
End Sub
```

Die Syntax von Anweisungen beruht auf folgenden Regeln :

- Eine einfache Anweisung wird durch das Ende einer Codezeile abgeschlossen.

- Ein Anweisungsblock besteht aus mehreren einfachen Anweisungen.
- Anweisungen, die sich innerhalb spezieller Schlüsselwörter befinden, werden als Anweisungsblock bezeichnet. Beispiel: `Sub … End Sub`
- Anweisungsblöcke können geschachtet wwerden.
- Eine Anweisung kann in der nächsten Zeile forgesetzt werden:
  - Ein Unterstrich (_)gilt als Fortsetzungszeichen (siehe obiges Beispiel)
  - Fortsetzungszeilen werden überlicherweise mit einem Tabstopp eingerückt.

2.2 Reservierte Wörter in VBA

Alle Wörter die fester Bestandteil von VBA sind, werden als reservierte Wörter bezeichnet. Dazu zählen u. a. alle

- Schlüsselwörter wie `As, New, True, False`
- Wörter die Anweisungen einleiten wie `Dim, Public, If, For, Next`
- Datentypen (siehe dazu Tab. 1)

2.3 Variablen deklarieren

2.3.1 Bezeichner

Bezeichner sind alle vom Anwender frei wählbaren Namen, mit denen im VBA-Code Prozeduren, Funktionen, Variablen und Konstanten bezeichnet werden müssen. Der Name der Prozedur `MeineErsteProzedur` ist beispielsweise ein frei gewählter Bezeichner.

2.3.2 Variablen

Die Deklaration der Variablen sollte bei der VBA-Programmierung immer erfolgen. In der Entwicklungsumgebung im Menü EXTRAS > OPTIONEN, Register EDITOR, Rahmen CODE-EINSTELLUNGEN wird deshalb das Kontrollkästchen VARIABLENDEKLARATION ERFORDERLICH aktiviert. Dadurch wird automatisch im Deklarationsbereich eines jeden Moduls der Befehl `Option Explicit` eingefügt. VBA zwingt einem damit, alle Variablen mit der `Dim` Anweisung zu deklarieren, bevor sie benutzt werden. Die explizite Variablendeklaration hilft ganz enorm dabei, Programmierfehler zu vermeiden. Außerdem wird die Ausführung des Codes beschleunigt.

2.3.3 Datentypen

Gebräuchliche Standarddatentypen sind:

Datentyp	Präfix für Variable	Art	Kurzbe-zeichnung	Wertebereich
Boolean	bln oder bol	Wahrheitswert	keines	TRUE oder FALSE
Byte	byt	Ganze Zahl	keines	0 bis 255
Currency	cur	Festkommazahl (für Währung)	@	15 Vor- und 4 Nachkommastellen
Date	dat oder dtm	Datums-/Zeitwert	keines	1.1.100 bis 31.12.9999
Double	dbl	Dezimalzahl (Fließkommazahl)	#	Zahlen mit insgesamt (Vor- und Nachkommastellen) 16 Stellen
Integer	int	Ganze Zahl	%	-32.768 bis 32.767
Long	lng	Ganze Zahl	&	-2.147.483.648 bis 2.147.483.647
Object	obj	Objektvariable	keines	Verweist auf ein Objekt
Single	sng	Dezimalzahl (Fließkommazahl)	!	Zahlen mit insgesamt (Vor- und Nachkommastellen) 8 Stellen
String	str	Text	$	Zeichenketten
Variant	var	Alle Typen	keines	zunächst unbestimmt

Tab. 1: Gebräuchliche Standarddatentypen in Excel

2.3.4 Variablennamen

Der Gebrauch von Variablen in VBA-Anweisungen ist vergleichbar mit dem Einsatz von Variablen in mathematischen Formeln. Für die Bildung von Variablennamen gelten bestimmte Regeln:

- Der Name des entsprechenden Bezeichners muss mit einem Buchstaben beginnen.
- Danach kann jede mögliche Kombination von Zahlen und Buchstaben sowie das Unterstreichungszeichen (_) folgen.
- Bezeichner dürfen keine Leerstellen, Punkte (.) oder andere Zeichen enthalten, die bei VBA für mathematische Berechnungen oder Vergleichsoperationen (=, +, – usw.) eingesetzt werden.
- Sonderzeichen wie #, %, &, ! oder ? sind nicht erlaubt.
- Variablennamen und andere Bezeichner dürfen maximal 255 Zeichen enthalten.
- Variablennamen dürfen nicht mit bestimmten VBA-Schlüsselwörtern identisch sein. Wird ein Variablenname gewählt, der mit einem reservierten Wort identisch ist, meldet VBA einen Syntaxfehler.
- Bezeichner müssen in ihrem Gültigkeitsbereich eindeutig sein. Das heißt, dass der Name innerhalb der Prozedur oder des Moduls, in dem die Variable eingesetzt wird, nur einmal deklariert sein darf. Wenn versehentlich zwei Variablen in einer Prozedur den gleichen Namen haben oder der Name der Variablen mit dem Namen einer Prozedur in dem gleichen Modul identisch ist, meldet VBA bei der Ausführung der Prozedur mit einem Laufzeitfehler.
- Der Name einer Variablen sollte ihre Aufgabe so gut wie möglich verdeutlichen.
- Variablen sollten nicht größer als erforderlich dimensioniert werden. Das gilt insbesondere für Datenfelder (siehe weiter unten).
- Zwischen Groß- und Kleinschreibung von Bezeichnern unterscheidet VBA nicht.

2.3.5 Präfix

Sinnvollerweise wird ein Präfix (vgl. Tab. 1) vor den eigentlichen Variablennamen gesetzt, aus dem der Datentyp hervorgeht. Das erleichtert die Lesbarkeit des Codes:

Datentyp	Variableninhalt	Beispiel für Variablennamen
String	Text (Zeichenkette)	strVorname, strNachname, strMsg
Integer	Ganze Zahlen von -32.768 bis 32.767	intZaehler, intPersonen, intSpalte
Date	Datum oder Zeit	dtmStart, dtmEnde, dtmHeute

Tab. 2 Beispiele für die Bildung von Variablennamen

Wenn Sie Ihre Variablennamen übersichtlich und aussagekräftig definieren möchten, empfiehlt sich folgende Schreibweise mit Präfix und Festlegung des Datentyps:

```
Dim dtmStart As Date
```

Zur expliziten Deklaration einer Variablen mit Festlegung des Datentyps wird die `Dim`-Anweisung benutzt. Sie bewirkt, dass VBA eine Variable vom Typ Datum erzeugt. Variablen eines Typs sollten möglichst in einer Codezeile deklarieren werden. Dazu einige repräsentative Beispiele:

```
Dim arrWoche(1 To 52) As Integer
Dim bolNeuKunde As Boolean
Dim curBrutto As Currency
Dim dblNetto As Double
Dim dblZeile As Double
Dim intZaehler As Integer, intPersonen As Integer, intSpalte As Integer
Dim lngEinwohner As Long
Dim objKunde As Object
Dim sngMwst As Single
Dim strVorname As String, strNachname As String, strMsg As String
Dim varEingabe As Variant
Dim wksQuelle As Worksheet, wksZiel As Worksheet
```

- Eine Variable, die innerhalb einer Prozedur deklariert wird, ist auch nur innerhalb dieser Prozedur gültig.
- Innerhalb des Gültigkeitsbereichs einer Variablen muss diese eindeutig benannt sein.
- In VBA ist es möglich, Variablen zu deklarieren, die in mehreren Prozeduren gültig sind.
- Wenn Variablen mit gleichem Namen sich in unterschiedlichen Gültigkeitsbereichen befinden, benutzt VBA die Variable mit der lokalsten Gültigkeit.

Die Gültigkeitsbereiche und –dauer von Variablen sind folgender Tabelle zusammengefasst:

Variablentyp	Gültigkeitsbereich	Gültigkeitsdauer	Deklaration
Globale Variablen	Sie müssen nur einmal deklariert werden und gelten dann in allen Prozeduren eines Projekts.	Während der gesamten Laufzeit der Anwendung.	Werden durch das Schlüsselwort `Public` im allgemeinen Deklarationsteil des Moduls deklariert.
Lokale Variablen	Gelten nur in der jeweiligen Prozedur, in der sie deklariert werden.	Bestehen nur solange, wie die jeweilige Prozedur abläuft.	Wenn mit dem Schlüsselwort `Dim` in der jeweiligen Prozedur deklariert, in der sie benutzt werden.
Statische Variablen	Sind lokale Variablen, die nur in der jeweiligen Prozedur gültig sind.	Während der gesamten Laufzeit der Anwendung.	Werden mit dem Schüsselwort `Static` in der jeweiligen Prozedur deklariert, in der sie benutzt werden.
Variablen auf Modulebene	Stehen allen Prozeduren des Moduls zur Verfügung, in dem sie deklariert werden.	Während der gesamten Laufzeit der Anwendung.	Werden mit dem Schlüsselwort `Private` im allgemeinen Deklarationsteil des Moduls deklariert.

Tab. 3: Gültigkeitsbereiche und Gültigkeitsdauer von Variablen

2.4 Konstanten

2.4.1 Benutzerdefinierte Konstanten

Verwenden Sie Werte in einem VBA-Programm, die sich nicht ändern, können diese als Konstanten mit dem Schlüsselwort `Const` deklariert werden. Gemäß empfohlener Namenskonvention werden Konstanten mit den Präfix `con` gekennzeichnet. Dazu drei Beispiele:

```
[Public|Private ] Const conBasisBetrag As Currency = 49.5
[Public|Private ] Const conAnzahl As Integer = 20
[Public|Private ] Const conEndung As String = ".xls"
```

2.4.1 Integrierte Konstanten

VBA stellt eine große Zahl von integrierten Konstanten zur Verfügung. Am Präfix des Namens dieser Konstanten lässt sich erkennen, zu welche Anwendung oder zu welchem Bereich sie gehören:

vb	VBA-Konstante, z. B. `vbYesNo`
xl	Excel-Konstante, z. B. `xlColumn`
fm	MSForms für Formulare, z. B. `fmActionPaste`

Tab. 4: Integrierte Konstanten

Tipp: Wenn integrierte Konstanten vorhanden sind, sollten nicht stattdessen benutzerdefinierten Konstanten deklariert werden.

2.5 Datenfelder

Zu einem Datenfeld können Werte gleicher Art in einer Liste zusammengefasst werden. Sie werden alle mit demselben Namen anhand ihres Indexes angesprochen. Oft erhalten Felder noch einen Präfix. Gemäß gängiger Namenskonventionen beginnen sie entweder mit „arr“ oder nur mit „a“, womit gekennzeichnet wird, dass es sich im ein Feld (engl. array) handelt. Dazu zwei Beispiele:

```
Dim arrWochenNummer(1 To 52) As Integer
Dim asngMwstSatz(3) As Single
```

Um beispielsweise 7 Wochentage unter dem Namen varWochenTag anzusprechen, wird folgendes Datenfeld benötigt:

varWochenTag	Mo	Di	Mi	Do	Fr	Sa	Sa
Index:	0	1	2	3	4	5	6

Tab. 5: Beispiel für ein Datenfeld mit sieben Elementen

Auf den ersten Wert (Mo) des Datenfeldes muss mit dem Index 0 zugegriffen werden, denn Datenfelder basieren standardmäßig auf null. Soll der Index stattdessen mit 1 beginnen, muss vorher im allgemeinen Deklarationsbereich der Befehl `Option Base 1` eingefügt werden.

Die folgende Prozedur `DatenfeldAuslesen` nutzt das in Tab.5 veranschaulichte Datenfeld `varWochenTag`. Sie verwendet die in VBA eingebauten Datumsfunktionen `WeekDay` und `Date`, um den Wochentag des aktuellen Tagesdatums als ganze Zahl zurückzugeben. Dann wird mit Hilfe der Fallstruktur `Select Case...End Select` das Kürzel des jeweiligen Wochentags ermittelt und anschließend mit `Debug.Print` im Direktfenster ausgegeben.

```
Sub DatenfeldAuslesen()
   Dim varWochenTag As Variant
   Dim strTag As String
   varWochenTag = Array("Mo", "Di", "Mi", "Do", "Fr", "Sa", "So")
   Select Case Weekday(Date, vbMonday) ' gibt Wochentag zurück
      Case 1
         strTag = varWochenTag(0)
      Case 2
         strTag = varWochenTag(1)
      Case 3
         strTag = varWochenTag(2)
      Case 4
         strTag = varWochenTag(3)
      Case 5
         strTag = varWochenTag(4)
      Case 6
         strTag = varWochenTag(5)
      Case 7
         strTag = varWochenTag(6)
   End Select
   Debug.Print "Heute ist " & strTag
End Sub
```

Statt des soeben benutzen `Array`-Befehls kann ein funktionsgleiches Datenfeld `astrTag` wie folgt definiert werden:

```
Sub Wochentage()
   Dim astrTag(1 To 7) As String ' Datenfeld
   Dim strKuerzel As String      ' Ziel-Zeichenkette
   Dim intTag As Integer         ' Schleifenzähler
   astrTag(1) = "Mo"             ' Wertzuweisung
   astrTag(2) = "Di"             ' usw.
   astrTag(3) = "Mi"
   astrTag(4) = "Do"
   astrTag(5) = "Fr"
   astrTag(6) = "Sa"
   astrTag(7) = "So"
   ' Scheifendurchlauf
   For intTag = 1 To 7 ' oder: For intTag = LBound(strTag) To UBound(strTag)
      strKuerzel = strKuerzel & astrTag(intTag) & ", "
   Next intTag
   ' Ausgabe der Ziel-Zeichenkette
   Debug.Print Left(strKuerzel, Len(strKuerzel) - 2)
End Sub
```

Der Index des Datenfelds `astrTag` startet hier mit 1. In der `For...Next`-Zählschleife werden die Kürzel der Wochentage mit `&` verbunden und danach ausgegeben. Das Direktfenster enthält dann folgende Zeichenkette: `Mo, Di, Mi, Do, Fr, Sa, So`

2.6 Literale

Feste Werte (z. B. Zahlen oder Texte), die im Programmcode eingegeben werden, bezeichnet man als Literale. Ihre Schreibweise ist abhängig vom jeweiligen Datentyp (siehe Tab. 1).

Numerische Datentypen	• Gültig sind die Ziffern 0 bis 9 und die Vorzeichen + und – • Fließkommazahlen erhalten einen Punkt (.) als Dezimaltrennzeichen.
Zeichen und Zeichenfolgen	Diese werden in Anführungszeichen (") eingeschlossen
Boolescher Datentyp	Zulässig sind: `True` oder `False`
Datum und Zeit	• Diese werden in Raute-Zeichen (#) eingebettet: #12/31/2011# • Zulässig ist auch die übliche deutsche Schreibweise: "31.12.2011"

Tab. 6: Schreibweise von Literalen im Programmcode

2.7 Zusammenfassendes Beispiel

Die folgende Prozedur berechnet die Zahl der Kalendertage im Jahr 2011. Sie verdeutlicht

- ihre Bezeichnung (`KalendertageBerechnen`)
- die explizite Deklaration der Variablen (`dtmJahresAnfang, dtmJahresEnde`)
- die Verwendung von Literalen (`#` und `"`) bei Datumsangaben
- den Einsatz der eingebauten Datumsfunktion (`DateDiff`)
- die Verwendung des Fortsetzungszeichens (_) beim Zusammenbau der Meldung des Befehls `MsgBox`
- die Verwendung von integrierten Konstanten (`vbMonday, vbFirstFourDays`)

```
Sub KalendertageBerechnen()
   Dim dtmJahresAnfang As Date
   Dim dtmJahresEnde As Date
   dtmJahresAnfang = "01.01.2011" ' deutsche Schreibweise
   dtmJahresEnde = #12/31/2011#
   MsgBox "Kalendertage im Jahr " & Year(dtmJahresAnfang) & ": " & _
      DateDiff("d", dtmJahresAnfang, dtmJahresEnde, vbMonday, vbFirstFourDays) + 1
End Sub
```

2.8 Schnellübericht

Was bedeutet …	
Anweisung	Nach syntaktischen Regeln erstellter Arbeitsschritt in einem Programm
Array	Datenfeld, das mehrere Elemente eines Datentyps speichern kann, auf die mittels eines Index zugegriffen werden kann.
Datentyp	legt fest, welche Werte eine Variable oder Konstante speichern kann und bestimmt den Speicherplatzbedarf (siehe Tab. 1)
Deklaration	Definition von Variablen oder Konstanten im Programm
Funktion	Eigenständiges Programm, das einen Wert an das aufrufende Programm zurückgibt.
Konstante	Wert im Programm, der sich nicht ändert
Prozedur	Eigenständiges Programm, das aber keinen Rückgabewert liefert.
Syntax	Regeln, nach denen Anweisungen in ein VBA-Programm geschrieben werden müssen.
Variable	Ist im Prinzip ein Platzhalter, der in einem Programm veränderbare Werte speichert

Sie möchten …	Schlüsselwörter
Eine Prozedur deklarieren	`Sub  End Sub`
Eine Funktion deklarieren	`Function  End Function`
Eine Variable deklarieren	`Dim Variablename`
Eine Konstante deklarieren	`Const Konstantenname = Wert`
Einen Datentyp zuweisen (vgl. Tab. 1)	`Dim Variablenname As Datentyp`
Ein Datenfeld deklarieren	`Dim Variablenname ([Untergrenze To ] Obergrenze) As Datentyp`
Einer Variablen einen Wert zuweisen	`Variablenname = Wert`
Arithmetische Operatoren	`+` addiert Zahlen miteinander `-` subtrahiert Zahlen voneinander `*` multipliziert Zahlenwerte `/` dividiert Zahlenwerte `\` Ganzzahlige Division `^` potenziert eine Zahl mit einem Exponenten `Mod` Divisionsrest bestimmen
Vergleichsoperatoren (aufgeführt in der Reihenfolge ihres Vorrangs)	`=` gleich `<>` ungleich `<` kleiner als `>` größer als `<=` kleiner oder gleich `>=` größer oder gleich
Logische Operatoren (aufgeführt in der Reihenfolge ihres Vorrangs)	`Not` logische Negation `And` beide Bedingungen müssen zutreffen `Or` eine Bedingung muss zutreffen `Xor` logische Exklusion
Verkettungsoperatoren	`Zeichenkette1 & Zeichenkette2` `Zeichenkette1 + Zeichenkette2`
Fortsetzungszeichen für Programmzeile	`_`
Einen Eingabedialog anzeigen	`Eingabewert = InputBox(...)`
Einem Ausgabedialog anzeigen	`MsgBox ...`

2.9 Übungen

2.9.1 Kreisumfang berechnen

Erstellen Sie eine benutzerfreundliche Prozedur `KreisumfangBerechnen` zur Ermittlung des Umfangs eines Kreises nach der Formal U = 2 x Kreiszahl x Radius, wobei der Standardwert für Radius = 10 cm beträgt und die Kreiszahl den Zahlenwert 3,142 besitzt. Deklarieren Sie die benötigten Konstanten und Variablen. Verwenden Sie einen Eingabe- und Ausgabedialog. Runden Sie das Ergebnis auf 2 Nachkommastellen.

2.9.2 Alter einer Person bei bekanntem Geburtsdatum berechnen

Arbeitsschritte:

❶	Öffnen Sie eine neue Arbeitsmappe
❷	Navigieren Sie zum VBA-Editor
❸	Legen Sie in der Prozedur `WieAltBistDu` die beiden String-Variablen `strVorname` und `strNachname` an. Die Datums-Variable für das Geburtsdatum soll `dtmGeburtsdatum` genannt werden.
❹	Berechnen Sie das Alter mit der Funktion `Lebensalter`. Übergeben Sie dieser Funktion die genannte Datumsvariable. Die Funktion soll das berechnete Lebensalter an die aufrufende Prozedur zurückgeben.
❺	Zeigen Sie das Alter in einem Ausgabedialog.

❻	Schließen Sie die Arbeitsmappe, ohne diese zu speichern

2.10 Lösungen

2.10.1 Kreisumfang berechnen

```
Sub KreisumfangBerechnen()
   ' Konstanten deklarieren
   Const conPi As Single = 3.142
   Const conTitle As String = "Kreisumfang berechnen."
   ' Variablen deklariern
   Dim varInput As Variant
   Dim dblRadius As Double
   Dim dblUmfang As Double
   ' Eingabedialog anzeigen
   varInput = InputBox("Radius des Kreises in Zentimeter?", conTitle, 10)
   If IsNumeric(varInput) Then            ' Wenn Eingabewert ist numerisch ist, ...
      dblRadius = CDbl(varInput)          ' dann Datentyp von var nach dbl umwandeln
      dblUmfang = 2 * conPi * dblRadius   ' und Kreisumfang bestimmen
      MsgBox "Bei einem Radius von " & dblRadius & _
         " cm beträgt der Umfang des Kreises " & _
         Format(Round(dblUmfang, 2), "###0.00") & " cm."
   Else  ' Wenn Eingabewert nicht numerisch ist, ...
      ' dann Fehlermeldung ausgeben
      MsgBox "Eine numerische Eingabe wird erwartet!", vbExclamation, conTitle
   End If
End Sub
```

2.10.2 Alter einer Person mit bekanntem Geburtsdatum berechnen

```
Sub WieAltBistDu()
   ' String-Variablen deklariern
   Dim strVorname  As String, strNachname As String
   Dim varEingabe As Variant
   ' Variant-Variable deklarieren
   varEingabe = InputBox("Vorname?")
   ' Vorname, Nachname und Geburtsdatum abfragen
   If Len(varEingabe) > 0 Then
      strVorname = CStr(varEingabe)
   End If
   varEingabe = InputBox("Nachname?")
   If Len(varEingabe) = 0 Then
      strNachname = "Anonymus"
   Else
      strNachname = CStr(varEingabe)
   End If
   varEingabe = InputBox("Geburtsdatum?")
   If IsDate(varEingabe) Then
      ' Meldung ausgeben
      MsgBox strVorname & " " & strNachname & " ist " & _
         Lebensalter(CDate(varEingabe)) & " Jahr alt.", _
         vbInformation, "Lebensalter"
   End If
End Sub
```

Die Prozedur `WieAltBistDu` ruft die Funktion `Lebensalter` auf, um das Alter in Jahren zu berechnen. Die Funktion `Lebensalter` gibt das berechnete Alter an die aufrufende Prozedur zurück.

Die gezeigte Lösung ist u. a. deshalb vorteilhaft , weil die Funktion `Lebensalter` gesondert ausgetestet werden kann.

```
Function Lebensalter(ByVal dtmGeburtsdatum As Date) As Integer
   Dim dtmHeute As Date
   Dim intTag As Integer, intMonat As Integer, intJahr As Integer
   Dim intGebTag As Integer, intGebMonat As Integer, intGebJahr As Integer
   ' Fehler abfangen, wenn Datentyp falsch ist
   If IsDate(dtmGeburtsdatum) Then
      ' Altersberechnung
      dtmHeute = DateValue(Now())
      intTag = Day(dtmHeute)
      intMonat = Month(dtmHeute)
      intJahr = Year(dtmHeute)
      intGebTag = Day(dtmGeburtsdatum)
      intGebMonat = Month(dtmGeburtsdatum)
      intGebJahr = Year(dtmGeburtsdatum)
      Lebensalter = intJahr - intGebJahr
      If (intMonat < intGebMonat) Or (intMonat = intGebMonat And intTag < intGebTag) Then
         Lebensalter = Lebensalter - 1
      End If
   End If
End Function
```

Die Prozedur `WieAltBistDu` erzeugt folgenden Ausgabedialog:

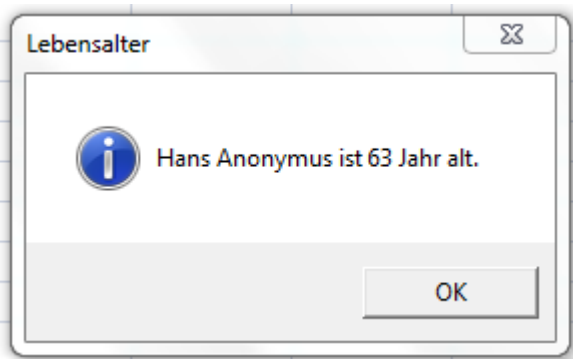


Abb. 6: Message Box bei der Altersberechnung

Die zweite und dritte Lösung sind übersichtlicher und deshalb vorzuziehen.