

# VBA

## Excel, Access, Word Makro Programmierung



# Vorab

- Kursablauf & Organisation
- Vorstellungsrunde
  - Firma, Vorkenntnisse, Erwartungen
- Bei Unklarheiten: **Sofort** unterbrechen!
- Präsentation und Beispiele sind später online verfügbar



# ppedv AG

- Firmensitz in Burghausen
- Schulungszentren
- Schulungen für nahezu alle Microsoft-Technologien
- Konferenzen, Camps, Verlag (VisualStudioOne / VSOne)

[blog.ppedv.de](http://blog.ppedv.de)



# Überblick



# VBA - Allgemein

- **Visual Basic for Applications**
  - Ermöglicht den Zugriff auf Office-Anwendungen
  - Unterschiede zwischen VB und VBA:
    - VB – Entwicklung eigenständiger Windows-Anwendungen
    - VBA – Automatisierung von Office-Anwendungen
- Möglichkeiten von VBA:
  - bestehende Funktionen ergänzen
  - benutzerdefinierte Dialoge generieren
  - Daten aus anderen Anwendungen nutzen

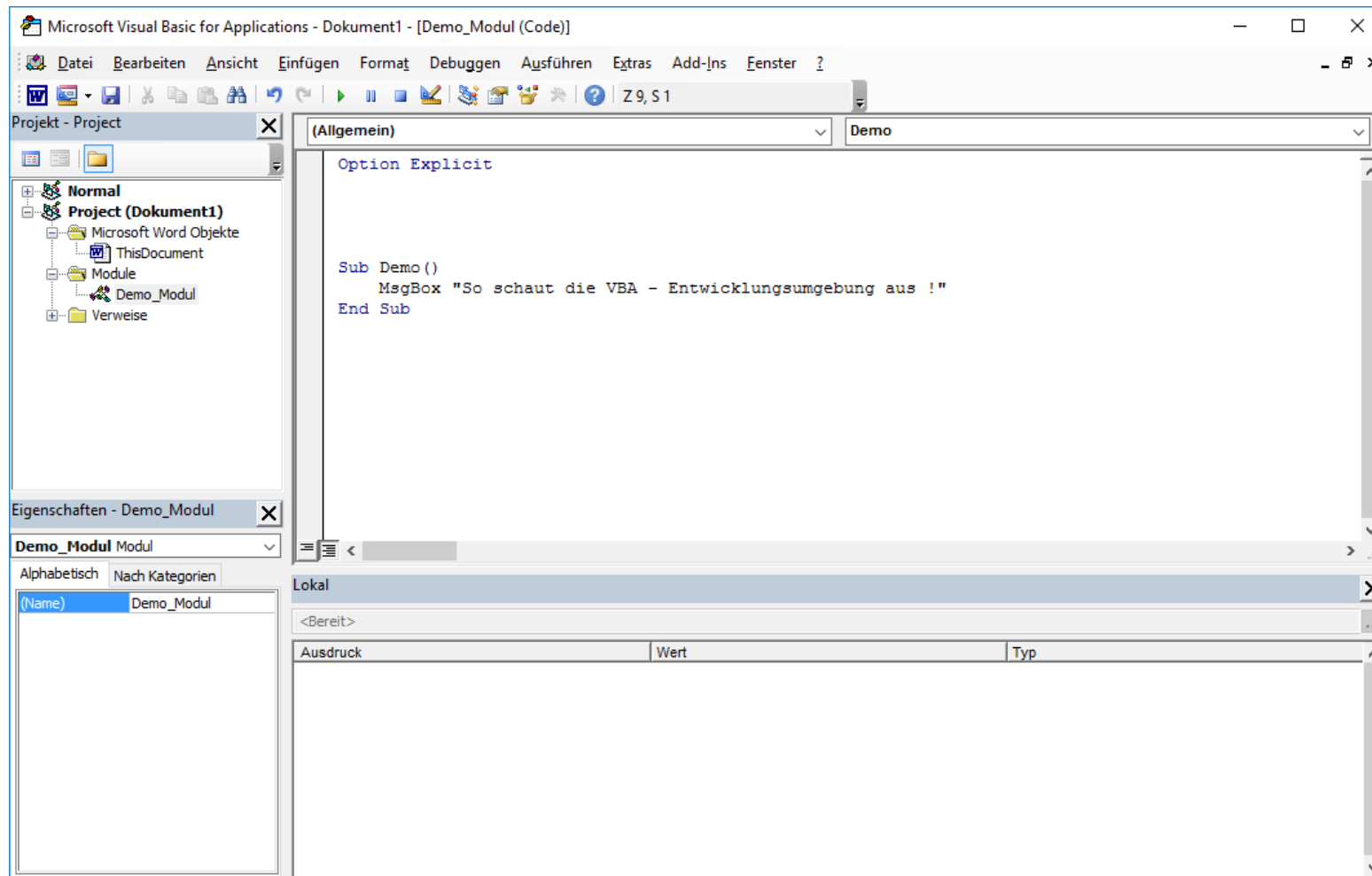


# Office - Paket

- Word
  - Textverarbeitung
- Excel
  - Tabellenkalkulation
  - Formeln
  - Diagramme
- Access
  - Datenspeicherung und -verwaltung

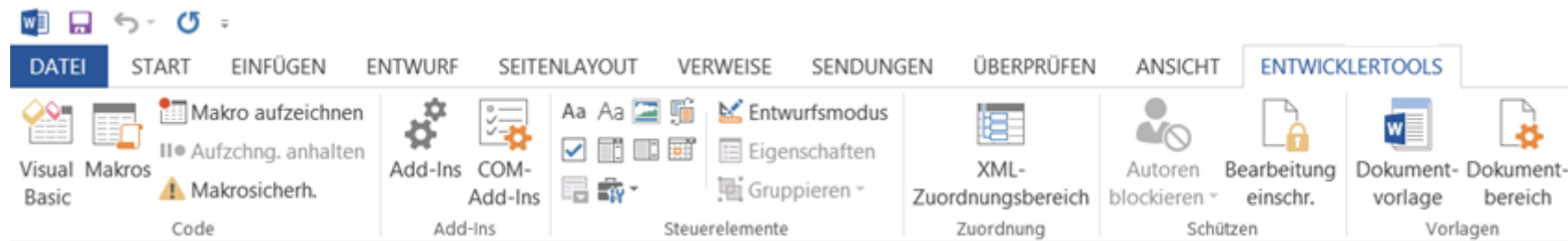


# Die VBA - Entwicklungsumgebung

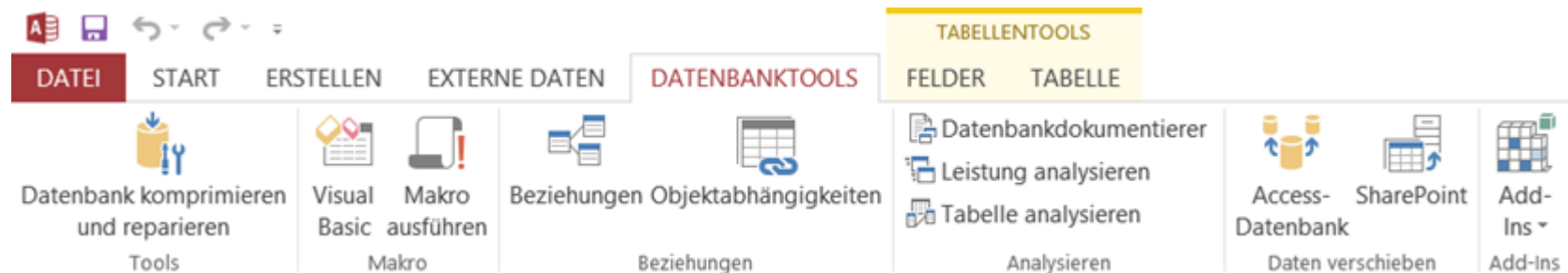


# Starten der VBA-Entwicklungsumgebung

- Word, Excel, Outlook und PowerPoint:
  - Registerkarte *Entwicklertools*:



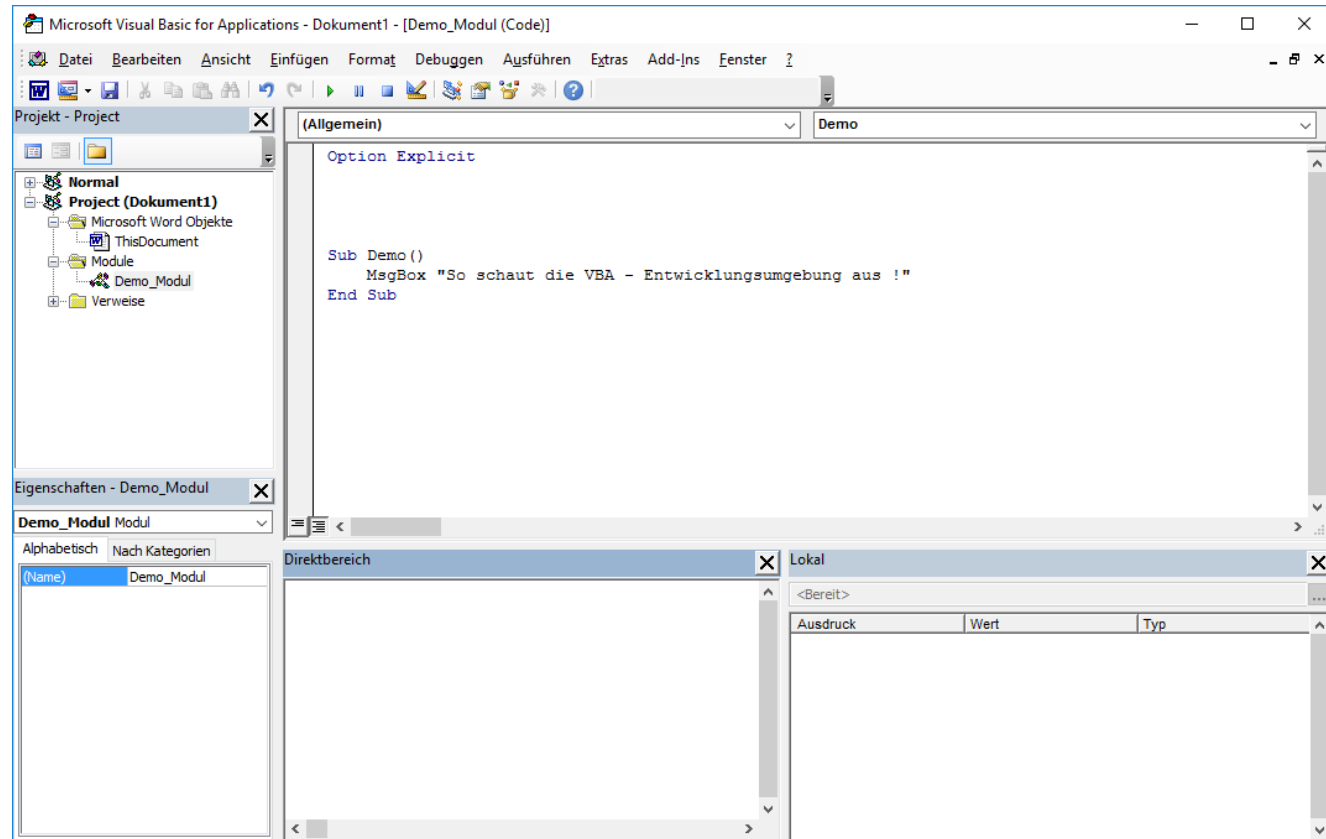
- Access:
  - Registerkarte *Datenbanktools*:





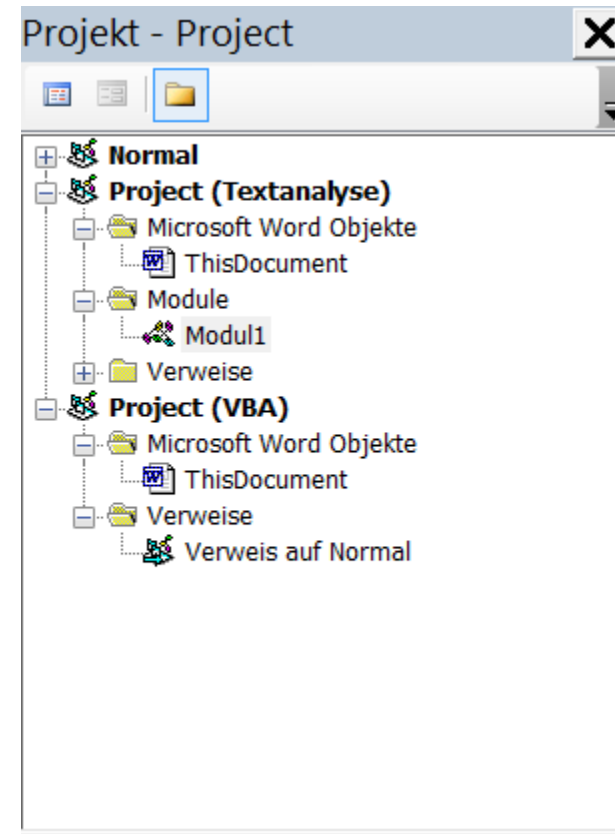
# Bestandteile der Entwicklungsumgebung

- Projekt-Explorer
- Eigenschaftsfenster
- Code-Fenster
- Direktbereich
- Lokal-Fenster



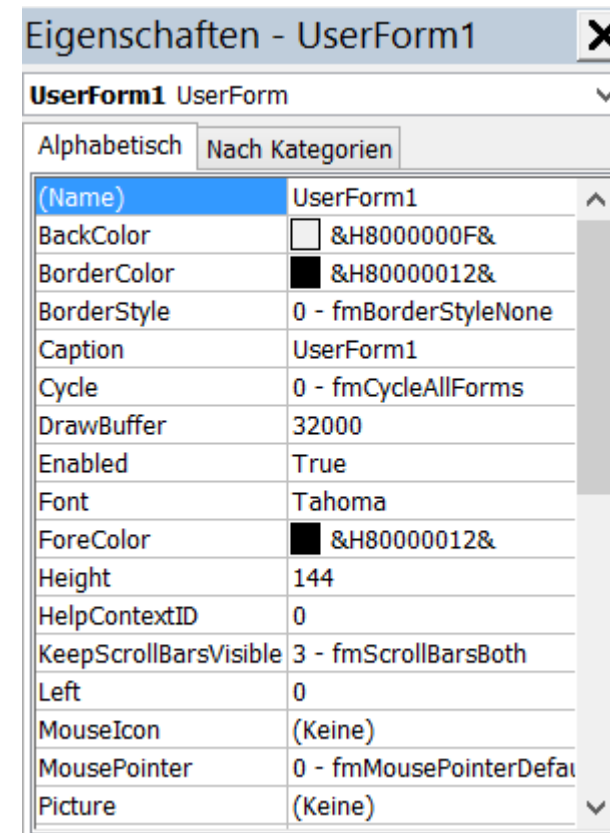
# Projektexplorer

- Anzeige für alle offenen Dokumente
- Gliederung von Makros in Modulen
- Baumstruktur
  - z.B. Unterordner für Makros



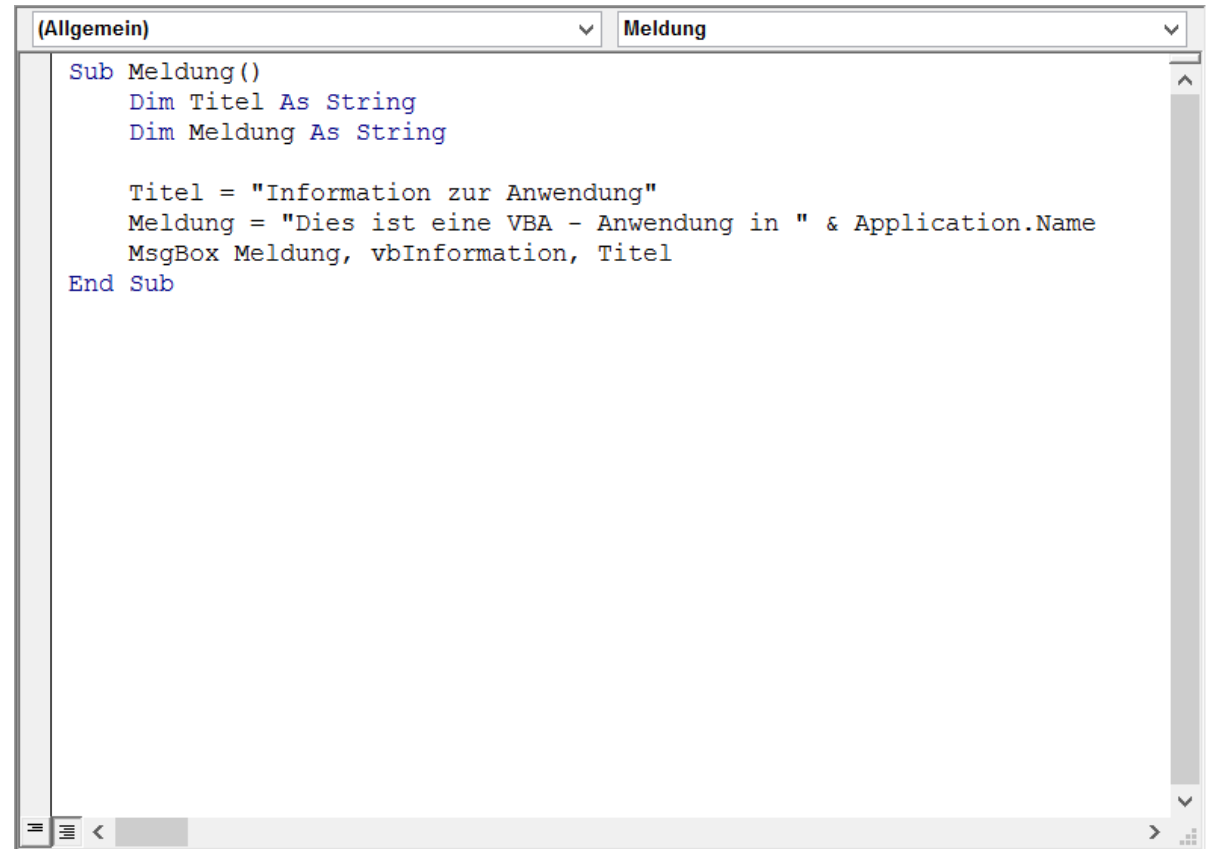
# Eigenschaftsfenster

- Eigenschaften des gewählten Moduls
- Eigenschaften eines Steuerelements im Formulardesigner
- Sortierung der Eigenschaften
  - alphabetisch
  - kategorisch



# Code-Fenster

- Quellcodeeditor
- Umfangreiche Eingabehilfen:
  - Automatische Syntaxüberprüfung
  - Anzeige von Objektmitgliedern
  - Autovervollständigung
  - farbliche Hervorhebung von Schlüsselwörtern



The screenshot shows the VBA Code Window with the 'Meldung' module selected. The code is as follows:

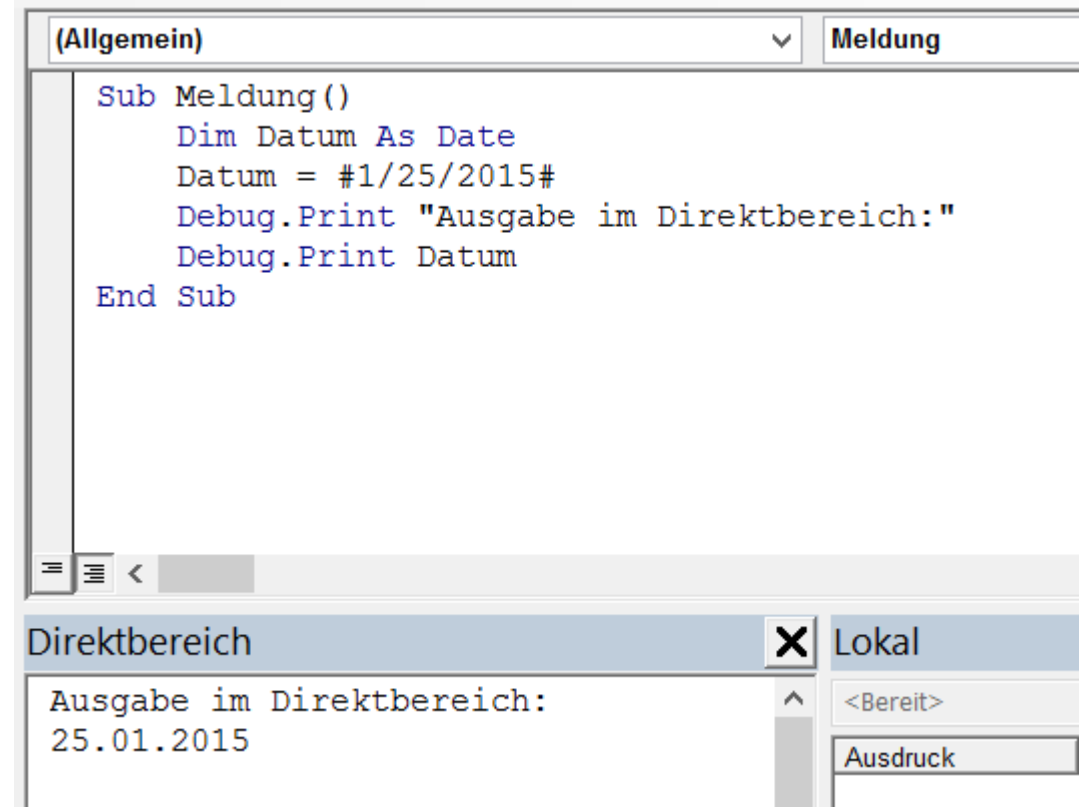
```
Sub Meldung()  
    Dim Titel As String  
    Dim Meldung As String  
  
    Titel = "Information zur Anwendung"  
    Meldung = "Dies ist eine VBA - Anwendung in " & Application.Name  
    MsgBox Meldung, vbInformation, Titel  
End Sub
```

The code is color-coded: 'Sub' and 'End Sub' are in blue, 'Dim' is in blue, and variable names and strings are in black. The window has a tab labeled 'Meldung' and a dropdown menu showing '(Allgemein)'.



# Direktbereich

- Direktes Ausführen von einzelnen Anweisungen
- Anzeige für Statusmeldungen während des Debuggings
  - `Debug.Print`



# Lokal-Fenster

- Stellt Informationen zu allen lokalen und globalen Variablen dar
- Hilfreich beim Debugging

(Allgemein) Meldung

```
Sub Meldung()  
    Dim Datum As Date  
    Dim Text As String  
    Text = "Diesen Text sieht man im Lokal-Fenster"  
    Datum = #1/25/2015#  
    Debug.Print "Ausgabe im Direktbereich:"  
    Debug.Print Datum  
End Sub
```

Lokal

Project.ThisDocument.Meldung

Ausdruck	Wert	Typ
Me		ThisDocument
Datum	#25.01.2015#	Date
Text	"Diesen Text sieht man im Lokal-Fenster"	String

# Die Sprachelemente von VBA



# Sprachsyntax

- VBA arbeitet mit Codeblöcken
  - Codeblöcke werden in Prozeduren eingeschlossen
  - Variablen, die innerhalb einer Prozedur definiert werden, sind nur innerhalb dieser Prozedur gültig
  - Variablen, die innerhalb eines übergeordneten Codeblocks definiert werden, sind auch innerhalb aller eingebetteten Codeblöcke gültig
- VBA ist nicht Case-Sensitiv !
- Kommentare
  - Apostroph: ' So schaut ein Kommentar aus





# Sprachsyntax - Beispiel

- Prozedur → `Sub Sprachsyntax()`
- Code → `MsgBox "Hallo Welt"`
- Kommentar → `' Das ist ein Kommentar`  
`End Sub`



# Variablen

- Zwischenspeicher für Daten
  - `Dim` Name `as` Datentyp
- Benennungsregeln:
  - Muss mit einem Buchstaben anfangen
  - Darf nur Buchstaben, Zahlen und Unterstriche enthalten
  - Darf nicht mit einem Schlüsselwort oder einer integrierten Funktion von VBA übereinstimmen
    - Beispiele: `sub`, `as`, `function`, `do`, `while`, `for`, ...



Datentyp:	Zeichen:	Verwendungszweck:	Wertebereich:
Byte	-	Ganze Zahlen	0-255
Integer	%	Ganze Zahlen	-32.768 bis 32.768
Long	&	Ganze Zahlen	-2.147.483.648 bis 2.147.483.647
Single	!	GK-Zahlen (einfache Gen.)	-3.4E+38 bis -1.4E-45 1.4E-45 bis 3.5E38 7 Ziffern Genauigkeit
Double	#	GK-Zahlen (doppelte Gen.)	-1.8E+308 bis -4.94E-324 4.94E-324 bis 1.8E308 15 Ziffern Genauigkeit
Currency	@	GK-Zahlen (Währung)	-9.22E+14 bis 9.22E+14 15 Vor- und 4 Nachkommastellen
String	\$	Zeichenketten	0 bis 65.535 Zeichen
Date	-	Datumswerte	1.1.100 bis 31.12.9999 Zeit: 00:00:00 bis 23:59:59
Boolean	-	Wahrheitswerte	True oder False
Variant	-	Beliebige Daten	Beliebige Daten



# Literale

- Ganze Zahlen: 3, 73, 1223
- Gleitkommazahlen: 3.45, 0.0045, 1234.444
- Zeichenketten: "Hallo", "1234"
- Datumswerte: "1.1.2007", #1/12/2007#,  
#1/12/2007 01:14 PM#
- Wahrheitswerte: True oder False



# Beispiel

```
Sub Variablen()  
  
    Dim text As String      ' Hier wird eine String-Variable initialisiert  
    Dim zahl As Integer    ' Hier wird eine Integer-Variable initialisiert  
    Dim kommazahl As Double ' Hier wird eine Double-Variable initialisiert  
    Dim test As Boolean     ' Hier wird eine Boolean-Variable initialisiert  
  
    text = "Hallo Welt"    ' Der Variable "text" wird der Wert "Hallo Welt" zugewiesen  
    zahl = 9               ' Der Variable "zahl" wird der Wert "9" zugewiesen  
    kommazahl = 3.14       ' Der Variable "kommazahl" wird der Wert "3.14" zugewiesen  
    test = False           ' Der Variable "test" wird der Wert "False" zugewiesen  
  
    MsgBox text            ' Preisfrage  
  
End Sub
```



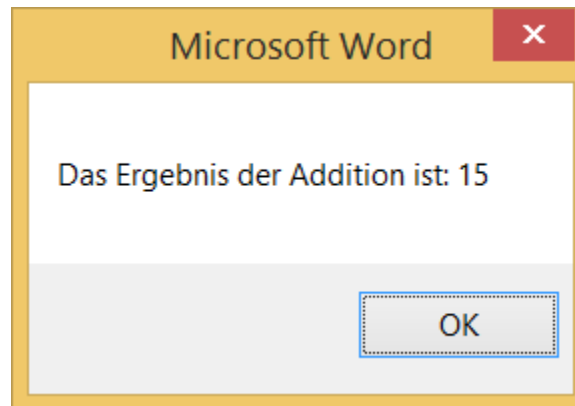
# Operatoren

Operator:	Bedeutung:
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
\	Ganzzahlige Division
^	Potenzierung
&	Stringverkettung
Mod	Restwertdivision (Modulo)



# Übung

- Erstellen Sie ein Programm, welches 2 Integer Zahlen addiert.  
Verwenden Sie dafür 3 Variablen: `zahl1`, `zahl2` und `ergebnis`
- Geben Sie das Ergebnis der Addition in einem Meldungsfenster (MsgBox) aus



# Ergebnis

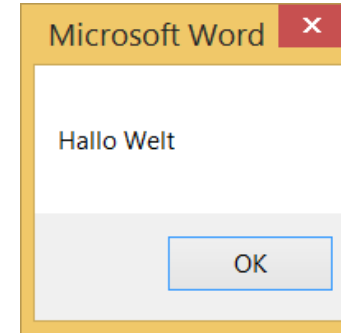
```
Sub Addition()  
Dim zahl1 As Integer  
Dim zahl2 As Integer  
Dim ergebnis As Integer  
  
zahl1 = 5  
zahl2 = 10  
ergebnis = zahl1 + zahl2  
  
msgbox "Das Ergebnis der Addition ist: " & ergebnis  
  
End Sub
```



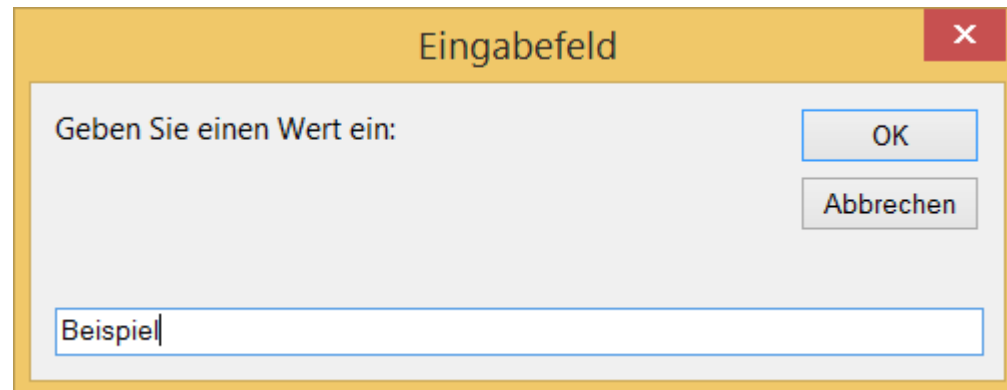


# Einfache Eingabe-/Ausgabefunktionen

- `MsgBox`: Ausgabe einfacher Meldungen

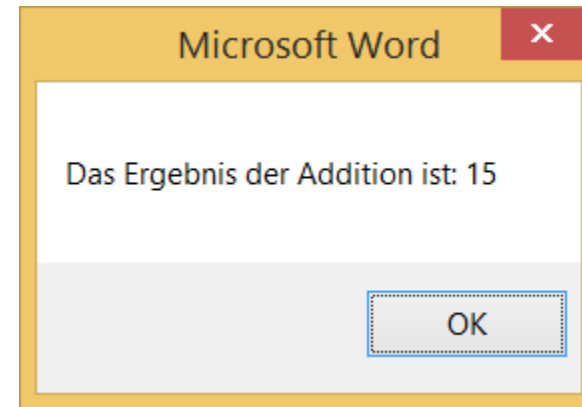
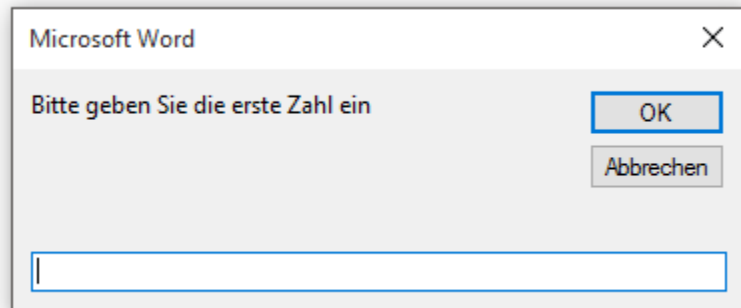


- `InputBox`: einfaches Einlesen von Werten



# Übung

- Schreiben Sie das vorherige Beispiel so um, dass die zwei Zahlen mithilfe einer `InputBox` eingelesen werden



# Ergebnis

```
Sub Addition2()  
    Dim zahl1 As Integer  
    Dim zahl2 As Integer  
    Dim ergebnis As Integer  
  
    zahl1 = InputBox("Bitte geben Sie die erste Zahl ein")  
    zahl2 = InputBox("Bitte geben Sie die zweite Zahl ein")  
  
    ergebnis = zahl1 + zahl2  
  
    MsgBox "Das Ergebnis der Addition ist: " & ergebnis  
End Sub
```



# Konstanten

- Schlüsselwort `Const`, gefolgt vom Namen
- Danach optional `As`, gefolgt vom Datentyp
- Zuweisung eines Wertes
- Integrierte Konstanten
  - `vb`, `wd`, `xl`, `ac`, `pp`, ...
- Beispiel:

```
Dim zahl1 As Integer
Dim zahl2 As Integer, zahl3 As Integer, wort1 As String
Const PI As Double = 3.14159
```



# Prozeduren

- Sub-Prozeduren
  - Teilaufgaben ohne Wertrückgabe
- Function-Prozeduren
  - Teilaufgaben mit Wertrückgabe
- Property-Prozeduren
  - Zugriff auf Felder in Klassen (siehe Objektorientierung)
- Ereignis-Prozeduren
  - Ereignisse in graphischen Benutzeroberflächen

```
Sub MeineSubprozedur()  
    ' Mein Code  
End Sub
```

```
Function MeineFunktion() As Integer  
    'Mein Code  
    MeineFunktion = 12 ' Rückgabe der Funktion  
End Function
```



# Aufruf von Prozeduren

- Sub-Prozedur:
  - Angabe des Namens ist ausreichend
  - Optional: Voranstellung von `Call`
- Function-Prozedur:
  - Aufruf erfolgt (üblicherweise) in einer Zuweisungsoperation
  - Kann auch wie Sub-Prozedur aufgerufen werden  
`Call` → Rückgabewert verfällt



# Beispiel

```
Sub Start()  
    MsgBox "Start"  
    Aufruf      ' Führt die Prozedur "Aufruf" aus  
    Call Aufruf ' Alternative Schreibweise  
End Sub  
  
Sub Aufruf()  
    MsgBox "Aufruf"  
End Sub
```



# Parameter

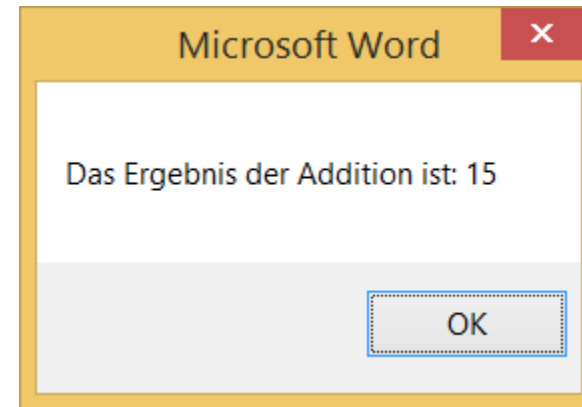
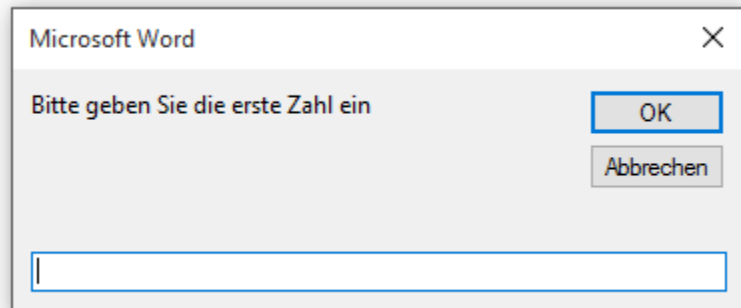
- Angabe nach dem Prozedurnamen in Klammern
- Keine Angabe von Gültigkeitsspezifizierern (wie z.B. `Dim`)
- Trennung mehrerer Parameter: Komma
- Übergabemethoden:
  - Per Wert: Übergabe einer Kopie (`ByVal`)
  - Per Referenz: Übergabe eines Verweises auf das Original (`ByRef`)
    - Standard in VBA: `ByRef`
- Zugriff erfolgt wie auf lokale Variablen





# Übung

- Schreiben Sie das vorherige Beispiel so um, dass die zwei eingegebenen Zahlen in einer Funktion addiert werden.



# Ergebnis

```
Sub Funktionsübung()  
    Dim zahl1 As Integer  
    Dim zahl2 As Integer  
    Dim ergebnis As Integer  
  
    zahl1 = InputBox("Bitte geben Sie die erste Zahl ein")  
    zahl2 = InputBox("Bitte geben Sie die zweite Zahl ein")  
    ergebnis = Addition(zahl1, zahl2)  
  
    MsgBox "Das Ergebnis der Addition ist: " & ergebnis  
End Sub  
  
Function Addition(z1 As Integer, z2 As Integer) As Integer  
    Addition = z1 + z2  
End Function
```



# Parameterübergabe

- Übergabe per Referenz (ByRef)

```
Sub Parameterübergabe()  
    Dim i As Integer  
    i = 10  
  
    Call Ändern(i)  
  
    MsgBox i  
End Sub
```

→

```
Sub Ändern(ByRef zahl As Integer)  
    '...  
    zahl = zahl * 2  
    MsgBox zahl  
End Sub
```

- Übergabe per Wert (ByVal)

```
Sub Parameterübergabe()  
    Dim i As Integer  
    i = 10  
  
    Call Ändern(i)  
  
    MsgBox i  
End Sub
```

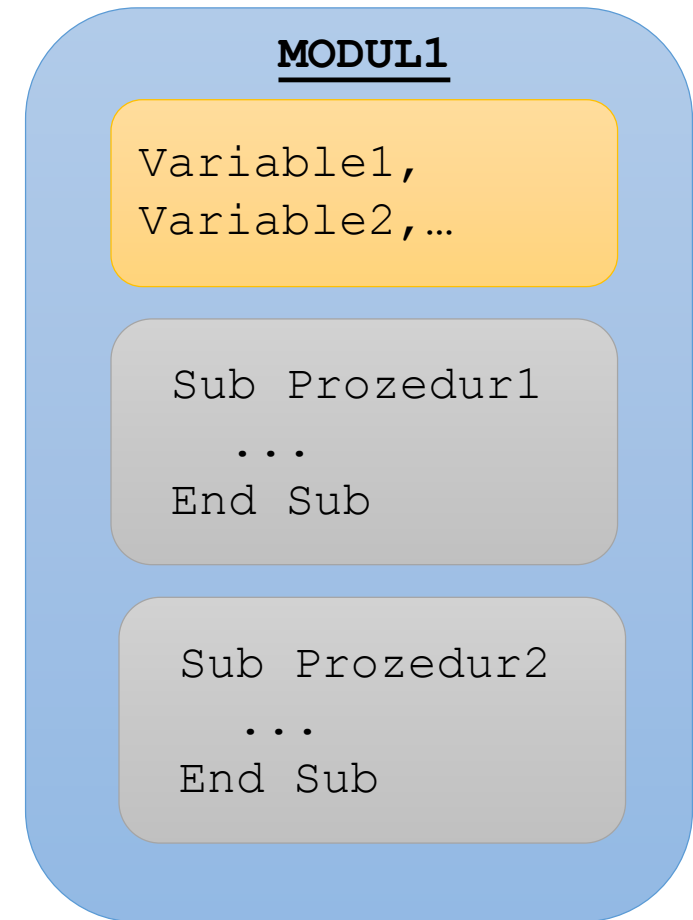
→

```
Sub Ändern(ByVal zahl As Integer)  
    '...  
    zahl = zahl * 2  
    MsgBox zahl  
End Sub
```



# Module

- Aufteilung von (Teil-) Aufgaben auf verschiedene Module
  - strukturierte Programmierung
- Bestandteile eines Moduls:
  - Deklarationsteil
    - Deklaration von globalen Variablen und Konstanten
  - Prozeduren



# Modultypen

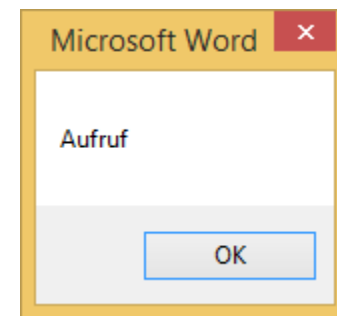
- Standardmodule:
  - Allgemeiner Quelltext für ein gesamtes Projekt
- Klassenmodule:
  - Beschreibung eines Objekts: Daten und Methoden (objektorientierte Programmierung)
- Dokumentenmodule:
  - Sind an ein Dokument gebunden
- UserForm-Module:
  - Modul (Benutzeroberfläche)
    - Ereignisprozeduren, die an Steuerelemente gebunden sind
- Formular- und Berichtsmodule:
  - Besondere Modultypen, existieren nur in Access
  - Sind an erweiterte Formulare und Berichte gebunden



# Gültigkeitsspezifizierer

Spezifizierer:	Bedeutung:
Public	Zugriff in allen Modulen/Projekten erlaubt
Private	Zugriff nur im eigenen Modul erlaubt
Static	Alle Variablen einer Prozedur werden statisch

```
Public Sub Start()  
    MsgBox "Start"  
    Aufruf  
    Call Aufruf  
End Sub  
  
Private Sub Aufruf()  
    MsgBox "Aufruf"  
End Sub
```



# Arrays

- Gruppe von Variablen
- Deklaration wie normale Variable
  - Anzahl an Elementen in Klammern
  - Indizierung beginnt bei 0
- Explizite Angabe einer Untergrenze möglich (To-Schlüsselwort)
- Deklaration mehrdimensionaler Arrays

```
Sub Array_Deklarieren()  
    Dim array1(1 To 10) As Integer      ' 1,2,3,4,5,6,7,8,9,10  
    Dim array2(10) As Integer           ' 0,1,2,3,4,5,6,7,8,9,10  
    Dim mehrdimensional(9, 9) As Integer ' 100 Möglichkeiten  
  
    array1(5) = 5  
    array2(5) = 10  
    mehrdimensional(0, 0) = array1(5)  
    mehrdimensional(0, 1) = array2(5)  
End Sub
```



# Dynamische Arrays

- Keine Festlegung der Elementanzahl
  - Klammerpaar bleibt leer
- Veränderung der Dimensions- und Elementanzahl mit der `ReDim`-Anweisung
  - Nachteil: Werte werden ebenfalls gelöscht
- Verhindern des Löschens bestehender Elemente mit der `Preserve`-Anweisung
- Freigabe von Speicher mit `Erase`





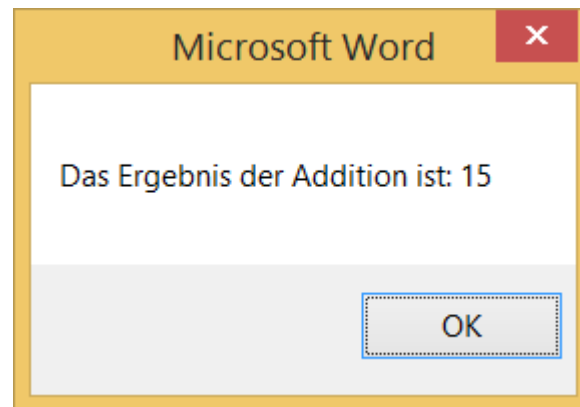
# Beispiel

```
Sub Dynamisches_Array()  
    Dim Zahlen() As Integer ' Ein dynamisches Array  
  
    ReDim Zahlen(5)          ' 0,1,2,3,4,5  
    Zahlen(0) = 10  
    Zahlen(1) = 9  
  
    ReDim Zahlen(3)          ' 0,1,2,3  
    Zahlen(2) = 8  
    Zahlen(3) = 7  
  
    ReDim Preserve Zahlen(8) ' 0,1,2,3,4,5,6,7,8  
    Zahlen(4) = 6  
    Zahlen(5) = 5  
  
    Erase Zahlen              ' Null  
End Sub
```



# Übung

- Schreiben Sie das vorherige Beispiel so um, dass der Inhalt von `zahl1`, `zahl2` in einem Array mit dem Namen „`zahlen`“ steht
- Speichern Sie das Ergebnis der Addition ebenfalls im Array „`zahlen`“ und geben Sie es danach in einem Meldungsfenster aus



# Ergebnis

```
Sub Array_Addition()  
    Dim zahlen(1 To 3) As Integer  
    zahlen(1) = 5  
    zahlen(2) = 10  
    zahlen(3) = zahlen(1) + zahlen(2)  
  
    MsgBox "Das Ergebnis der Addition ist: " & zahlen(3)  
End Sub
```



# Kontrollstrukturen

- Verzweigung mit der `If`-Anweisung
- Einschluss der Bedingung zwischen `If` und `Then`
- `If`-Block endet mit `End If`
- Alternative Verzweigung mit `Else` möglich
- Mehrfache Verzweigungen mit `ElseIf`-Ketten
- Vergleichsoperatoren für Bedingungen:

<	kleiner als
<=	kleiner oder gleich
>	größer als
>=	größer oder gleich
=	gleich
<>	ungleich
like	Mustervergleich



# Kontrollstrukturen

- Verknüpfung von Bedingungen
- Verknüpfungsoperatoren:
  - And (Und)
  - Or (Oder)
  - Xor (Exklusives Oder)
  - Not (Negierung)



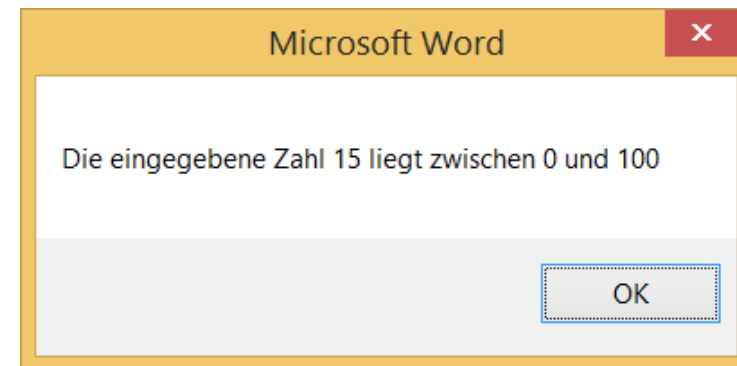
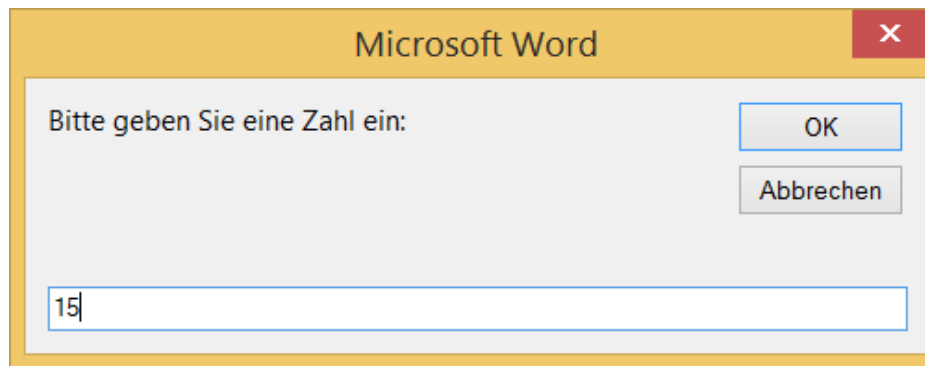
# Beispiel

```
Sub Verzweigung_Summe()  
Dim zahl1 As Integer  
Dim zahl2 As Integer  
Dim ergebnis As Integer  
  
zahl1 = 5  
zahl2 = 10  
ergebnis = zahl1 + zahl2  
  
If ergebnis < 10 Then  
    MsgBox "Das Ergebnis der Addition ist kleiner als 10"  
  
ElseIf ergebnis < 15 Then  
    MsgBox "Das Ergebnis der Addition ist kleiner als 15"  
  
ElseIf ergebnis >= 15 And ergebnis < 30 Then  
    MsgBox "Das Ergebnis der Addition liegt zwischen 15 und 30"  
  
End If  
End Sub
```



# Übung

- Lesen Sie mithilfe einer `InputBox` eine Zahl ein und überprüfen Sie, ob diese zwischen 0 und 100 liegt.
- Trifft dies zu, soll ein Meldungsfenster (`MsgBox`) dies ausgeben.
- Falls die eingegebene Zahl „50“ ist, soll das Programm ein zusätzliches Meldungsfenster öffnen



# Ergebnis

```
Sub Verzweigung_0_100()  
Dim eingabe As Integer  
  
eingabe = InputBox("Bitte geben Sie eine Zahl ein: ")  
  
If eingabe > 0 And eingabe < 100 Then  
    MsgBox "Die eingegebene Zahl " & eingabe & " liegt zwischen 0 und 100."  
    If eingabe = 50 Then  
        MsgBox "Jackpot ! Du hast meine Glückszahl erraten"  
    End If  
End If  
  
End Sub
```





# Kontrollstrukturen

- Fallauswahl mit `Select Case`
  - Angabe verschiedener Fälle mit `Case`
  - Alternativ-Fall mit `Case Else`
  - Angabe eines Bereichs mit `To`
  - Verwendung von Operatoren mit dem `Is`-Operator
    - Verknüpfungen nicht möglich



# Vergleich von Select und If

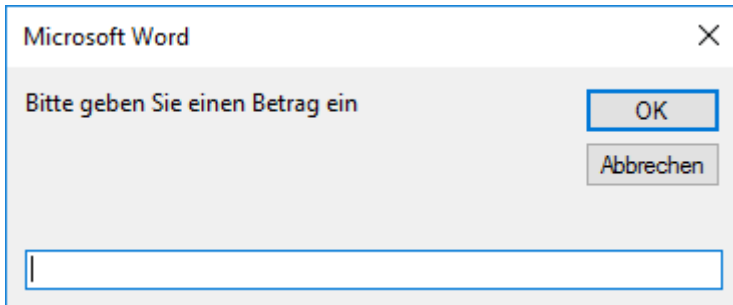
```
Sub Noten_If()  
    Dim Note As Integer  
    Dim Text As String  
  
    Note = InputBox("Bitte geben Sie Ihre Note ein: ")  
  
    If Note = 1 Then  
        Text = "Sehr Gut"  
    ElseIf Note = 2 Then  
        Text = "Gut"  
    ElseIf Note = 3 Then  
        Text = "Befriedigend"  
    ElseIf Note = 4 Then  
        Text = "Genügend"  
    ElseIf Note = 5 Then  
        Text = "Nicht Genügend"  
    Else  
        Text = "Ungültige Eingabe"  
    End If  
  
    MsgBox Text  
  
End Sub
```

```
Sub Noten_Select()  
    Dim Note As Integer  
    Dim Text As String  
  
    Note = InputBox("Bitte geben Sie Ihre Note ein: ")  
  
    Select Case Note  
        Case 1  
            Text = "Sehr Gut"  
        Case 2  
            Text = "Gut"  
        Case 3  
            Text = "Befriedigend"  
        Case 4  
            Text = "Genügend"  
        Case 5  
            Text = "Nicht Genügend"  
        Case Else  
            Text = "Ungültige Eingabe"  
    End Select  
  
    MsgBox Text  
  
End Sub
```



# Übung

- Erstellen Sie ein Makro, welches einen Betrag einliest und einen Rabatt verrechnet.



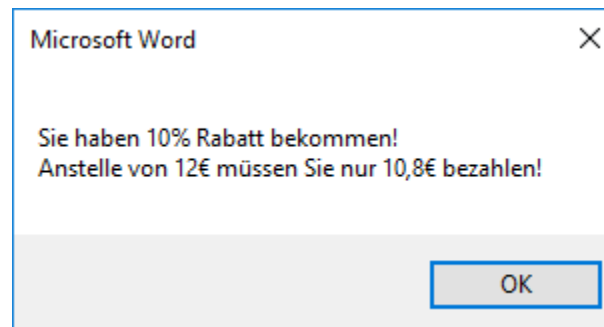
Microsoft Word

Bitte geben Sie einen Betrag ein

OK

Abbrechen

Betrag	Rabatt
weniger als 10€	5%
weniger als 25€	10%
weniger als 70€	25%
alles über 70€	33%



Microsoft Word

Sie haben 10% Rabatt bekommen!  
Anstelle von 12€ müssen Sie nur 10,8€ bezahlen!

OK



# Lösung

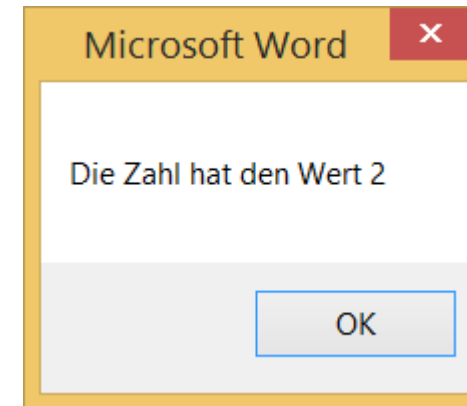
```
Sub Rabattrechner()  
    Dim betrag As Currency  
    Dim rabatt As Currency  
    Dim ergebnis As Currency  
  
    betrag = InputBox("Bitte geben Sie einen Betrag ein")  
  
    Select Case betrag  
        Case Is < 10  
            rabatt = 0.05  
        Case Is < 25  
            rabatt = 0.1  
        Case Is < 70  
            rabatt = 0.25  
        Case Else ' über 70€  
            rabatt = 0.33  
    End Select  
  
    ergebnis = betrag - betrag * rabatt  
    MsgBox "Sie haben " & rabatt * 100 & "% Rabatt bekommen!" & vbCrLf & _  
        "Anstelle von " & betrag & "€ müssen Sie nur " & ergebnis & "€ bezahlen!"  
End Sub
```



# Kontrollstrukturen

- Sprünge zu einer beliebigen Marke innerhalb einer Prozedur mit der GoTo-Anweisung
- Deklaration einer Marke durch Angabe eines beliebigen Namens gefolgt von einem Doppelpunkt

```
Sub GoTo_Beispiel()  
    Dim Zahl As Integer  
    Zahl = 2  
  
    If Zahl = 1 Then  
        GoTo Marke1  
    Else  
        GoTo Marke2  
    End If  
    Exit Sub  
  
Marke1:  
    MsgBox "Die Zahl hat den Wert 1"  
Marke2:  
    MsgBox "Die Zahl hat den Wert 2"  
End Sub
```



# Schleifen

- Zählergesteuert:
  - For-Next-Schleife
- Bedingungsgesteuert:
  - Do-Loop-Schleife

```
Sub Hochzählen1()  
    Dim Zähler As Integer  
  
    For Zähler = 1 To 20  
        MsgBox "Nr. " & Zähler  
    Next  
End Sub
```

```
Sub Hochzählen2()  
    Dim Zähler As Integer  
    Zähler = 0  
  
    Do While Zähler < 20  
        Zähler = Zähler + 1  
        MsgBox "Nr. " & Zähler  
    Loop  
End Sub
```

```
Sub Hochzählen3()  
    Dim Zähler As Integer  
    Zähler = 0  
  
    Do  
        Zähler = Zähler + 1  
        MsgBox "Nr. " & Zähler  
    Loop While Zähler < 20  
End Sub
```



# For-Next-Schleife

- Einschluss des Codes zwischen `For` und `Next`
- Steuerung mit einem Zähler
- Angabe eines Bereichs mit `To`
- Schrittweise Erhöhen: Standardmäßig 1
  - beliebige Veränderung mit `Step` möglich
- Explizites Verlassen mit `Exit For`

```
Sub Hochzählen1()  
    Dim Zähler As Integer  
  
    For Zähler = 1 To 20  
        MsgBox "Nr. " & Zähler  
    Next  
End Sub
```

```
Sub Hochzählen1()  
    Dim Zähler As Integer  
  
    For Zähler = 0 To 20 Step 2  
        MsgBox "Nr. " & Zähler  
        If Zähler = 10 Then  
            Exit For  
        End If  
    Next  
End Sub
```



# Do-Loop-Schleife

- Einschluss des Codes zwischen Do und Loop
- Angabe der Bedingung am Anfang oder Ende möglich (kopf- bzw. fußgesteuert)
- Steuerung der Abbruchbedingung mit While und Until
- Explizites Verlassen mit Exit Do

```
Sub Hochzählen2()  
    Dim Zähler As Integer  
    Zähler = 0  
  
    Do While Zähler < 20  
        Zähler = Zähler + 1  
        MsgBox "Nr. " & Zähler  
    Loop  
End Sub
```

```
Sub Hochzählen2()  
    Dim Zähler As Integer  
    Zähler = 0  
    Do While Zähler < 20  
        MsgBox "Nr. " & Zähler  
        If Zähler = 10 Then  
            Exit Do  
        End If  
        Zähler = Zähler + 2  
    Loop  
End Sub
```





# Mehrfachausführung von Code

## Vergleich: GoTo und Do-Loop - Schleife

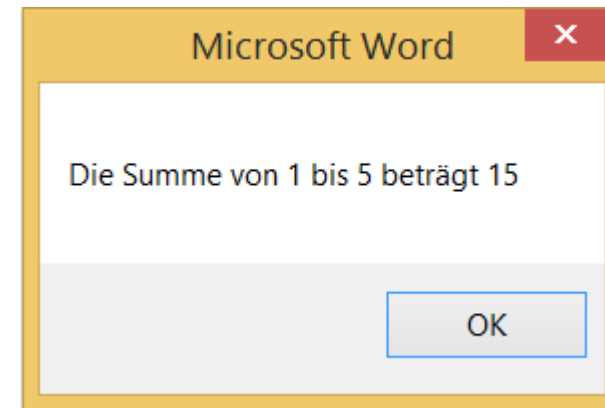
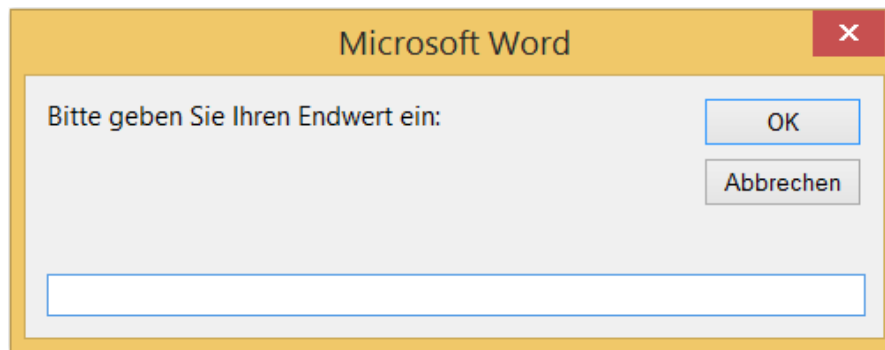
```
Sub Möglichkeit1()  
    Dim Zahl As Integer  
    Zahl = 5  
  
    Abfrage:  
    If Zahl >= 10 Then  
        GoTo Ende  
    Else  
        Zahl = Zahl + 1  
    End If  
    GoTo Abfrage  
  
Ende:  
    MsgBox "Die Zahl hat den Wert " & Zahl  
End Sub
```

```
Sub Möglichkeit2()  
    Dim Zahl As Integer  
    Zahl = 5  
  
    Do Until Zahl >= 10  
        Zahl = Zahl + 1  
    Loop  
  
    MsgBox "Die Zahl hat den Wert " & Zahl  
End Sub
```



# Übung

- Erstellen Sie ein Programm, welches mit einer `InputBox` eine Zahl einliest und alle Zahlen von 1 bis zu dem eingegebenen Endwert addiert.
- Nach der erfolgreichen Rechenoperation wird das Ergebnis in einer `MsgBox` ausgegeben:



# Ergebnis

```
Sub Summe()  
    Dim Endwert As Integer  
    Dim Summe As Integer  
  
    Summe = 0  
    Endwert = InputBox("Bitte geben Sie Ihren Endwert ein: ")  
  
    For zähler = 1 To Endwert  
        Summe = Summe + zähler  
    Next  
  
    MsgBox "Die Summe von 1 bis " & Endwert & " beträgt " & Summe  
End Sub
```



# Fehlerbehandlung

- Beim Auftreten eines Fehlers:

- Verzweigung zu einer Sprungmarke:
- Ausschalten der Fehlerbehandlung:
- Ignorieren von Fehlern mit:

`On Error GoTo Marke`

`On Error GoTo 0`

`On Error Resume Next`

- Fortsetzen der Programmausführung:

- Resume bzw. Resume Next

- Analyse eines Fehlers mit dem `Err`-Objekt

- Auslösen eines Fehlers mit `Raise`-Methode des `Err`-Objekts



# Beispiel mit Fehlerbehandlung

```
Sub Summe_mit_Fehlerbehandlung()  
    On Error GoTo Fehlermeldung ' Sprung zur Marke "Fehlermeldung"  
    Dim Endwert As Integer  
    Dim Summe As Integer  
  
    Summe = 0  
    Endwert = InputBox("Bitte geben Sie Ihren Endwert ein: ")  
    On Error GoTo 0  
  
    For zähler = 1 To Endwert  
        Summe = Summe + zähler  
    Next  
  
    MsgBox "Die Summe von 1 bis " & Endwert & " beträgt " & Summe  
    Exit Sub ' Prozedur wird beendet  
  
Fehlermeldung:  
    MsgBox "Bitte geben Sie nur ganze Zahlen ein !"  
    Resume  
End Sub
```



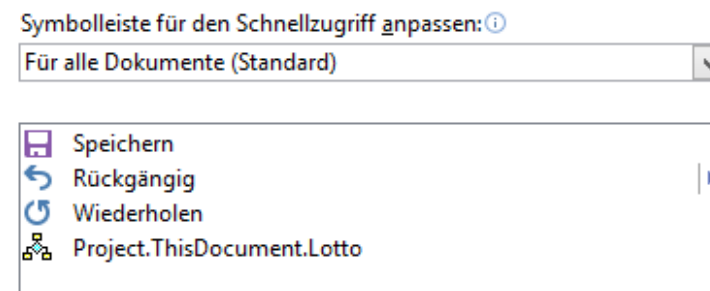
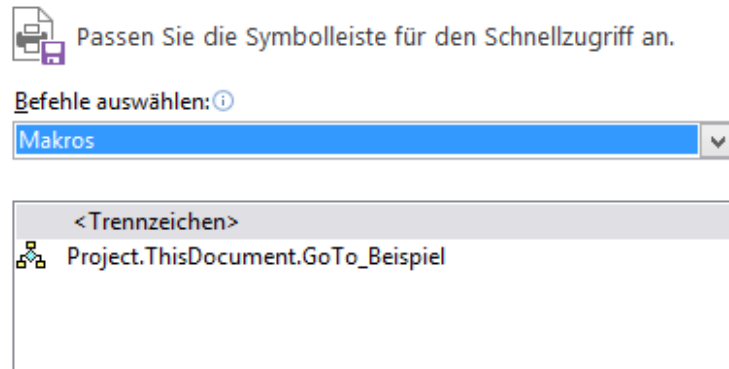
# Debugging

- Haltepunkte setzen
- Schritt-für-Schritt-Programmausführung
- Überwachung:
  - Direktfenster
  - Lokalfenster



# VBA-Prozeduren in die Menüleiste einfügen

- VBA-Prozedur = Makro
- Datei → Optionen → Symbolleiste für den Schnellzugriff
  - Optional: neue Symbolleiste für ein Dokument erstellen
- VBA-Prozeduren: Register „Befehle auswählen“: Makros
  - Gewünschte Prozedur in eine beliebige Symbolleiste einfügen



# Objektorientierte Programmierung in VBA





# Was ist ein Objekt?

- Abgeschlossene Einheit aus Daten und Prozeduren
  - Interne Implementierung eines Objekts bleibt verborgen
- Kommunikation über Methoden und Eigenschaften
- Kapselung einer bestimmten Funktionalität
  - z.B. der eines Dokuments
- Vorlage für Objekte = Klasse
- Objekt = konkretes Implementierung einer Klasse (Instanz)



# Instanzen (Objektvariablen)

1. Variable deklarieren und als Datentyp den Namen der Klasse angeben
2. Neue Instanz der Klasse (im Speicher) mit `New` anlegen:
  - Aufruf des Konstruktors
  - Verkürzte Syntax: `As New`
3. Zuweisen der Objektreferenz an eine Variable mit `Set`
4. Speicherfreigabe: Objektvariable auf `Nothing` setzen :
  - führt zum internen Aufruf des Destruktors

```
Dim Max As Person  
Set Max = New Person
```

```
' oder
```

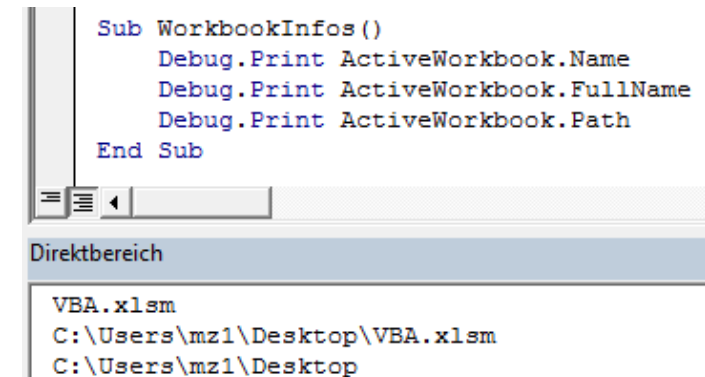
```
Dim Moritz As New Person
```

```
' Arbeitsspeicher freigeben  
Set Moritz = Nothing
```



# Felder

- interne Daten eines Objekts
- Deklaration wie globale Variablen
  - Deklaration für interne Verwendung: `Private`
  - Zugriff von außen: `Public`
- Zugriff erfolgt nicht wahlfrei, sondern kann genau gesteuert werden
  - `Property Get` für den Lesezugriff
  - `Property Let` für den Schreibzugriff



```
Sub WorkbookInfos()  
    Debug.Print ActiveWorkbook.Name  
    Debug.Print ActiveWorkbook.FullName  
    Debug.Print ActiveWorkbook.Path  
End Sub
```

Direktbereich

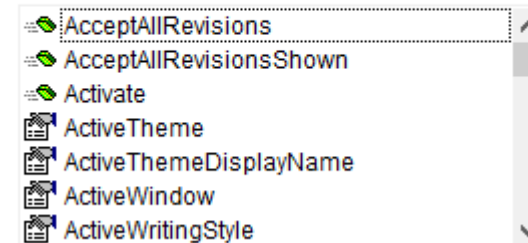
VBA.xlsm  
C:\Users\mz1\Desktop\VBA.xlsm  
C:\Users\mz1\Desktop

# Methoden

- Prozedur eines Objekts = Methode
  - Nur Sub- und Function-Prozeduren
- interne/öffentliche (Private/Public) Methoden
- Zugriff auf (öffentliche) Methoden über eine Instanz und den .-Operator

```
Sub Beispiel()
```

```
Dim doc As Word.Document  
doc.
```



# Beispiel

## Klasse Person

```
Dim vn As String
Dim nn As String
Dim k As Currency

Private Sub Class_Initialize()
    vn = ""
    nn = ""
    k = 0
End Sub

Private Sub Class_Terminate()

End Sub

Public Property Let Vorname(ByVal neuerVorname As String)
    vn = neuerVorname
End Property

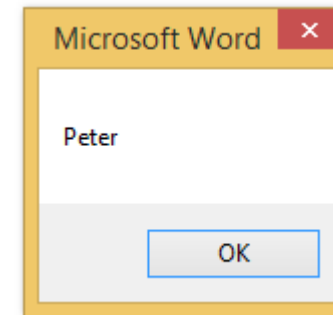
Public Property Get Vorname() As String
    Vorname = vn
End Property
```

## Prozedur Personaldatenbank

```
Sub Personaldatenbank()
    Dim mitarbeiter As Person

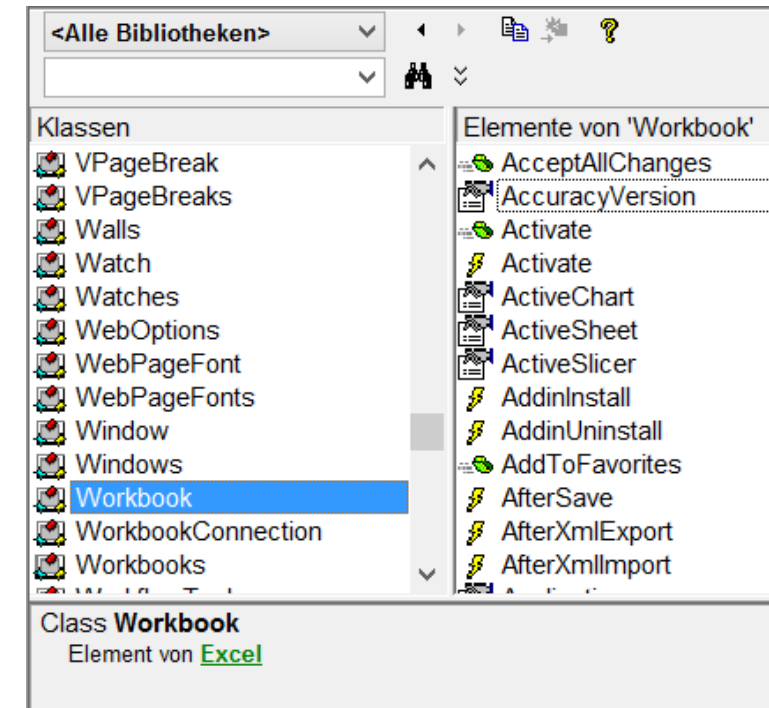
    Set mitarbeiter = New Person
    mitarbeiter.Vorname = "Peter"
    MsgBox mitarbeiter.Vorname

End Sub
```



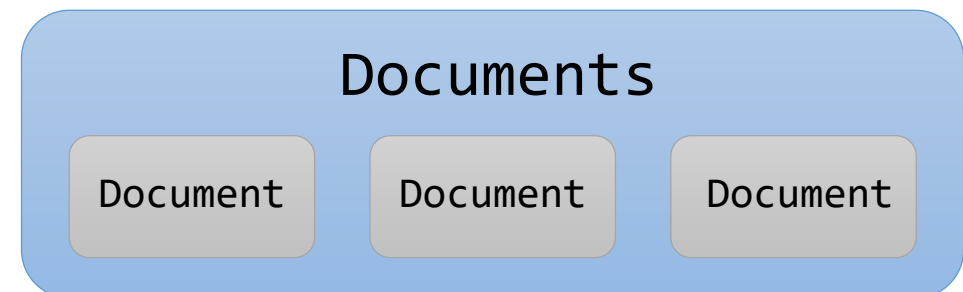
# Objektkatalog

- Anzeige aller verfügbaren Module des aktuellen Projekts
- Anzeige von Prozeduren und Variablen
- Anzeige von Methoden, Eigenschaften und Ereignissen in Klassenmodulen
- Genaue Beschreibung der einzelnen Objekte



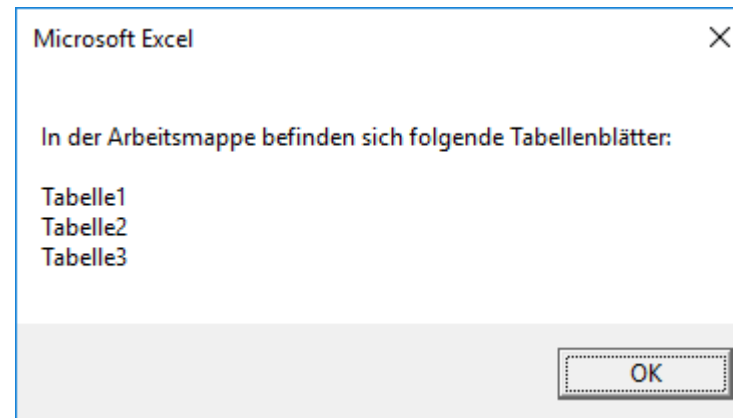
# Auflistungen

- Vielfach verwendet in VBA
- Zugriff erfolgt über Indizes oder die `Item`-Methode
- Anzahl der Elemente in der `Count`-Eigenschaft
- Durchlaufen aller Elemente eines bestimmten Typs in einer Auflistung mithilfe einer For-Each-Schleife
- Indizierung beginnt immer bei 1



# Übung

- Erstellen Sie ein Programm, welches die Namen aller Excel-Tabellenblätter ausgibt. Nutzen Sie hierfür die `Count` Eigenschaft der `Liste Worksheets`.





# Ergebnis

```
Sub ExcelCounter()  
    Dim index As Integer  
    Dim Ausgabe As String  
  
    Ausgabe = "In der Arbeitsmappe befinden sich folgende Tabellenblätter:" & vbCrLf & vbCrLf  
    For index = 1 To Worksheets.Count  
        Ausgabe = Ausgabe & Worksheets(index).Name & vbCrLf  
    Next  
  
    MsgBox Ausgabe  
End Sub
```



# Gemeinsam genutzte VBA - Elemente



# Wichtige VBA – Objekte und Methoden

- MsgBox
- InputBox
- Application
  - Repräsentiert die Anwendung
- Window
  - Objekt zum Steuern des Dokumentenfensters
- FileDialog
  - Objekt zum Anzeigen eines Dialogfensters
- Dir
  - VBA-Funktion zum Suchen von Dateien und Ordner
- Assistant
  - Objekt des Office-Assistenten



# Application-Objekt

- Stellt die gesamte Office-Anwendung inklusive der Anwendungsfenster dar
- Größe und Position der Anwendungsfenster ist änderbar
  - Standardeinheit: Pixel
  - Width: Breite
  - Height: Höhe
  - Left: Abstand vom linken Rand
  - Top: Abstand vom oberen Rand
  - WindowState: Anzeigemodus
    - Normal, Minimiert, Maximiert
  - Name: Name der Anwendung



# Window-Objekt

- Stellt ein Dokumentenfenster in einer Anwendung dar
- Ermöglicht die Anpassung des Dokumentenfensters
- Wichtige Eigenschaften/Methoden:
  - `Activate`: Aktiviert das Dokumentenfenster
  - `Caption`: Beschriftung
  - `Close`: Schließt das Dokumentenfenster
  - `Width/Height/Left/Top`: Größe und Position
  - `WindowState`: Aktueller Status des Fensters
    - Normal, Maximiert, Minimiert



# Window-Objekt

- Weitere, wichtige Eigenschaften/Methoden:
  - `Visible`: Gibt an, ob ein Dokumentenfenster sichtbar ist
- Liste aller Dokumentenfenster in der `Windows`-Auflistung
  - `Word: Documents`
  - `Excel: Workbooks`
  - `PowerPoint: Presentations`
- Aktives Fenster kann über `ActiveWindow` angesprochen werden



# Übung

- Passen Sie das aktuelle Anwendungsfenster mit folgenden Eigenschaften an:
  - Setzen Sie `WindowState` auf `wdWindowStateNormal`
  - 75 Pixel Abstand vom oberen und linken Bildschirmrand
  - Breite und Höhe des Fensters : je 500 Pixel



# Lösung

```
Sub Kopie()  
    ActiveWindow.WindowState = wdWindowStateNormal  
    ActiveWindow.Width = 500  
    ActiveWindow.Height = 500  
    ActiveWindow.Top = 75  
    ActiveWindow.Left = 75  
End Sub
```





# FileDialog-Objekt

- Stellt einen Dialog dar, mit dem Dateien/Verzeichnisse gesucht und ausgewählt werden können
- Typen:
  - Datei öffnen
  - Datei speichern
  - Datei auswählen
  - Verzeichnis auswählen



# FileDialog-Objekt

- Wichtige Methoden/Eigenschaften:
  - `Filters`: Liste von Dateifiltern
  - `FilterIndex`: Ausgewählter Filter
  - `AllowMultiSelect`: Mehrfachauswahl
  - `InitialFileName`: Startverzeichnis
  - `SelectedItems`: Liste der ausgewählten Dateien
  - `Title`: Titel des Dialogs
  - `Show`: Zeigt den Dialog an



# Beispiel

- Beispiel: Dialogfenster zur Auswahl von Word-Dokumenten

```
Sub Dialogbeispiel()  
    Dim fd As FileDialog  
    Dim zähler As Integer  
    Set fd = Application.FileDialog(msoFileDialogFilePicker)  
  
    fd.Filters.Add "Word-Dokumente", "*.doc; *.docx; *.docm; *.rtf"  
    fd.AllowMultiSelect = True  
    fd.Show  
  
    For zähler = 1 To fd.SelectedItems.Count  
        Application.Selection.TypeText fd.SelectedItems(zähler) & vbCrLf  
    Next  
End Sub
```

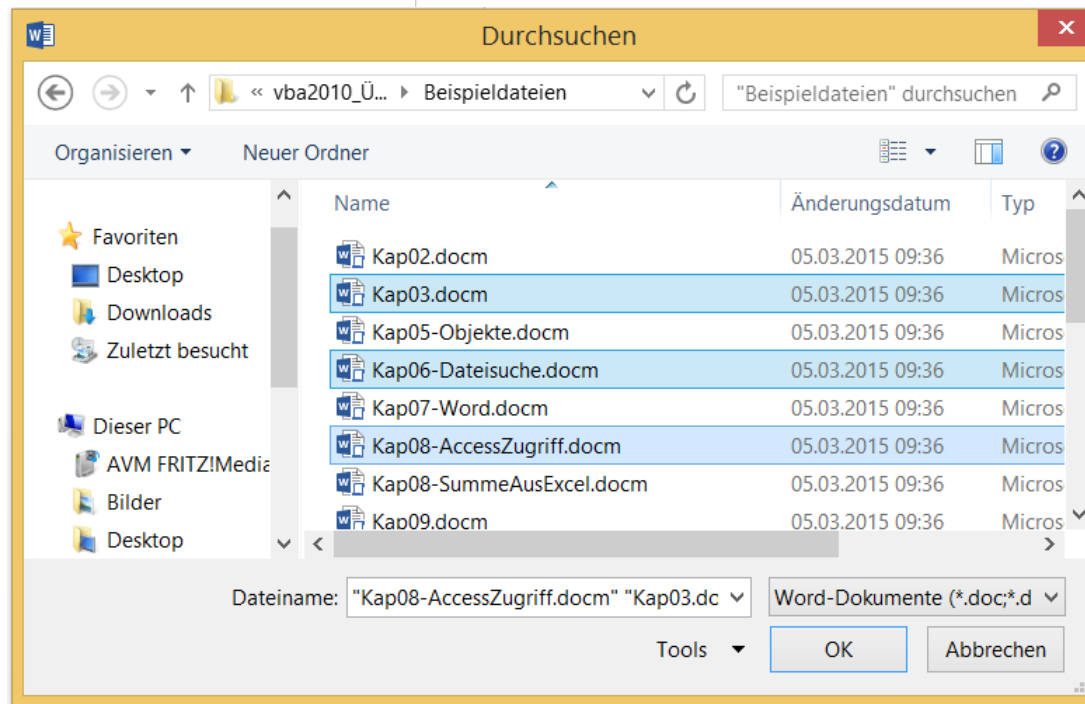


# Beispiel

C:\Users\mz1\Downloads\vba2010\_Übungsdateien\Beispieldateien\Kap03.docm

C:\Users\mz1\Downloads\vba2010\_Übungsdateien\Beispieldateien\Kap06-Dateisuche.docm

C:\Users\mz1\Downloads\vba2010\_Übungsdateien\Beispieldateien\Kap08-AccessZugriff.docm



# Dir - Funktion

- Liefert Dateinamen oder Ordnernamen, der mit den Angaben in den Parametern übereinstimmt
- Platzhalter \* und ?
  - „C:\Users\User\Dokumente\VBA\\*.doc“
- Beispiel:

```
Sub Dateisuche()  
    Dim Dateiname As String  
    Const pfad As String = "C:\Users\mz1\Desktop\VBA - Kurs\Beispiele Teil 1\*.doc?"  
  
    Dateiname = Dir(pfad)      ' Liefert eine Auflistung zurück  
  
    Do Until Dateiname = ""    ' Wenn Dir "" zurückliefert, gibt es keine weitere Dateien  
        MsgBox Dateiname      ' Der gefundene Pfad wird ausgegeben  
        Dateiname = Dir       ' Speichert den nächsten Pfad in der Variable  
End Sub
```



# Programmieren von Office-Anwendungen



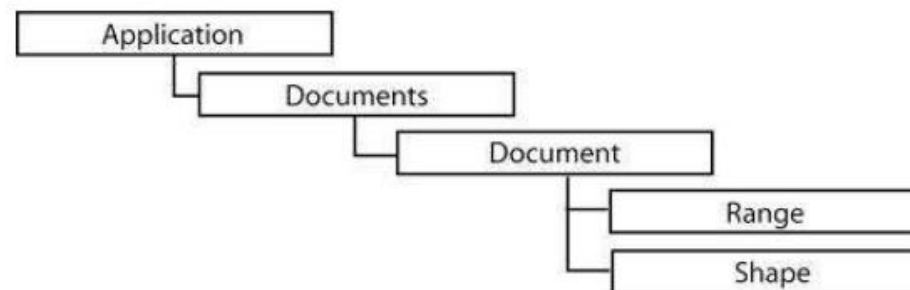
# Anwendungen

- Word
- Excel
- Access



# Das Word-Objektmodell

- Oberstes Objekt: `Application`, stellt die Anwendung selbst dar
- Enthält `Documents`-Liste für die Dokumente der Anwendung
- `Document`-Objekt steht für einzelnes Dokument
- `Range`-Objekt stellt einen begrenzten Bereich eines Dokuments dar





# Mit Dokumenten arbeiten

- Das aktuelle Dokument ist über die `ActiveDocument`-Eigenschaft verfügbar
- Öffnen von vorhandenen Dokumenten
  - `Open`-Methode der `Documents`-Liste
- Hinzufügen von Dokumenten
  - `Add`-Methode
- Schließen eines Dokuments
  - `Close`-Methode eines `Document`-Objekts



# Beispiel

```
Sub Paragraphen_Auslesen()  
    Dim doc As Document  
    Dim inhalt As String  
    Dim zähler As Integer  
  
    Set doc = Documents.Open(ActiveDocument.Path & "\VBA.docx")  
  
    For zähler = 1 To doc.Paragraphs.Count  
        inhalt = inhalt & doc.Paragraphs(zähler).Range.Text  
    Next  
  
    MsgBox inhalt  
End Sub
```



# Weitere Möglichkeiten bei Dokumenten

- **Wichtige Methoden/Eigenschaften (Document):**
  - `FullName`: Name und Pfad des Dokuments
  - `Path`: Pfad des Dokuments
  - `PrintOut`: Druckt ein Dokument aus
  - `Protect`: Schützt ein Dokument mit einem Passwort
  - `Save`: **Speichert** das Dokument
  - `SaveAs`: **Speichert** das Dokument unter einem neuen Namen
  - `Saved`: **Gibt an**, ob das Dokument seit der letzten Änderung gespeichert wurde



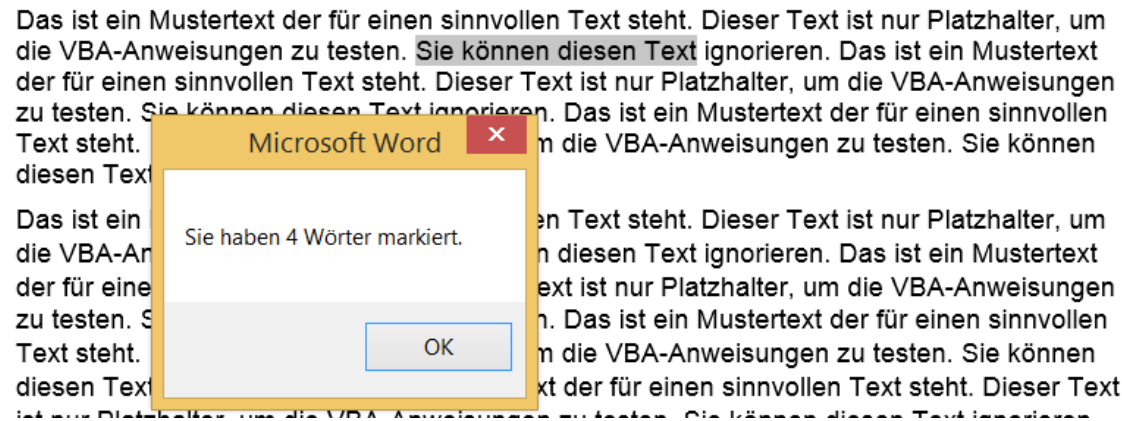
# Zugriff auf den Inhalt eines Dokuments

- Zugriff auf markierten Teil eines Dokuments mit Selection-Objekt:
  - Characters/Sentences/Paragraphs/Words: Zugriff auf Zeichen/Sätze/Absätze/Wörter der Markierung
  - Start/End: Liefert Start-/Endposition der Markierung
  - Text: Text der Markierung
  - Move: Markierung verschieben
  - WholeStory: Markiert das gesamte Dokument
  - Collapse: Hebt die aktuelle Markierung auf
  - TypeText: Text einfügen



# Übung

- Markieren Sie in einem geöffneten Word-Dokument einen Absatz und zählen Sie die markierten Wörter



# Lösung

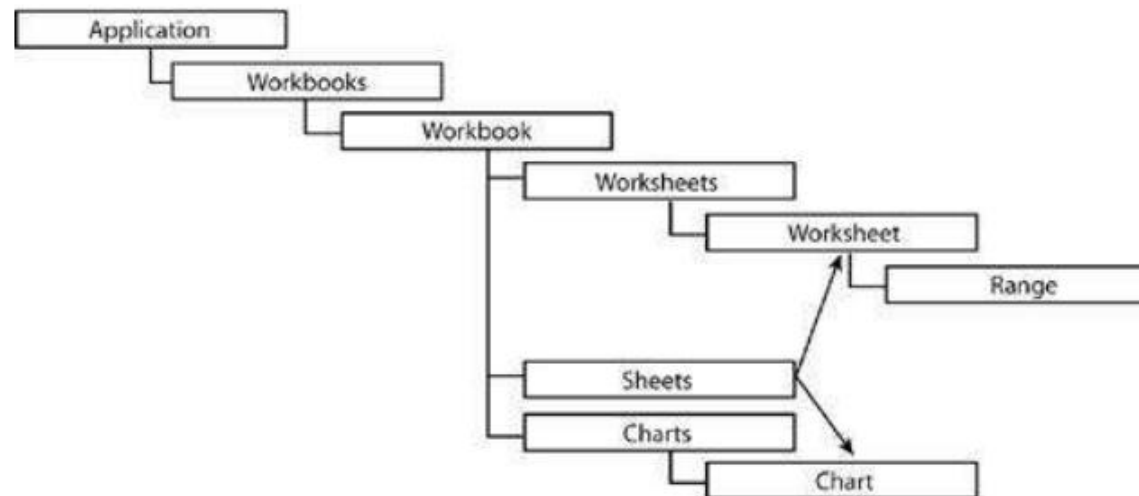
```
Sub Markierte_Wörter()  
    Dim wörter As Integer  
  
    wörter = Selection.Words.Count  
    MsgBox "Sie haben " & wörter & " Wörter markiert."  
End Sub
```

```
Sub Markierte_Wörter_2()  
    Dim wörter As Integer  
  
    wörter = Selection.Range.ComputeStatistics(wdStatisticWords)  
    MsgBox "Sie haben " & wörter & " Wörter markiert."  
End Sub
```



# Das Excel-Objektmodell

- Oberstes Objekt: `Application`, stellt die Anwendung selbst dar
- Enthält `Workbooks`-Liste für die einzelnen Arbeitsmappen der Anwendung
- Enthält `Worksheets`-Liste für die einzelnen Arbeitsblätter



# Mit Arbeitsmappen arbeiten

- Die aktive Arbeitsmappe ist über die `ActiveWorkbook`-Eigenschaft verfügbar
- Öffnen von vorhandenen Arbeitsmappen
  - `Open`-Methode der `Workbooks`-Liste
- Hinzufügen mit `Add`-Methode
  - Verwendung einer Vorlage möglich
- Schließen einer Arbeitsmappe
  - `Close`-Methode eines `Workbook`-Objekts





# Arbeiten mit Tabellenblättern

- Der Zugriff auf Tabellenblätter erfolgt über die `Worksheets`-Auflistung einer Arbeitsmappe (`Workbook`)
  - Zugriff auf Diagramme
  - Zugriff auf Tabellenbereiche
  - Zugriff auf Zellen
- Tabellenblätter hinzufügen
  - `Add`-Methode
- Tabellenblätter verschieben
  - `Move`-Methode



# Arbeiten mit Tabellenblättern

- Wichtige Methoden/Eigenschaften eines Tabellenblattes (Worksheet):
  - `Activate`: Aktiviert ein Tabellenblatt
  - `Delete`: Löscht ein Tabellenblatt
  - `Name`: Name des Tabellenblattes
  - `Range`: Ermöglicht Zugriff auf den Inhalt eines Tabellenblattes



# Zugriff auf Zellen und Zellbereiche

- Zugriff erfolgt über das Range-Objekt:
  - Columns/Rows: Zugriff auf Spalten/Zeilen
  - Select: Selektiert einen Zellbereich
  - Activate: Cursor auf eine Zelle setzen
  - Interior: Innenbereich einer Zelle
  - Formula: Die Formel einer Zelle
  - Value: Der Wert der Zelle
  - Text: Der Wert der Zelle inklusive der Formatierung
- Zugriff auf selektierte Zelle
  - ActiveCell-Eigenschaft
- Zugriff auf einzelne Zellen
  - Cells-Eigenschaft eines Worksheet-Objekts



# Beispiel

- Auslesen und Schreiben von Daten in einem Excel Worksheet

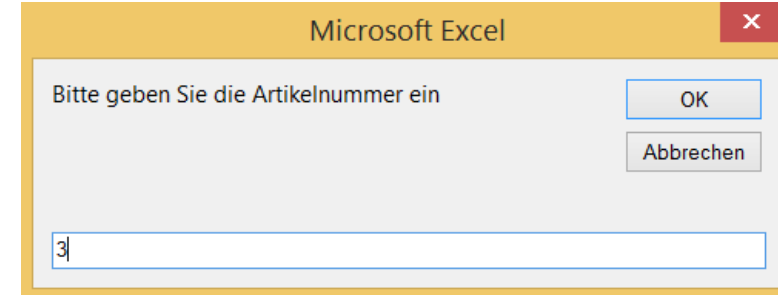
```
Sub Zellenzugriff()  
    Dim zeile As Variant  
    Dim zähler As Integer  
  
    For zähler = 1 To 10  
        Worksheets(1).Cells(zähler, 1).Value = zähler  
        Worksheets(1).Range("B" & zähler).Value = zähler  
        ' Beide Zeilen führen die gleiche Operation aus  
    Next  
  
    zeile = InputBox("Bitte geben Sie eine Zeilennummer ein")  
    If IsNumeric(zeile) = True Then  
        MsgBox Worksheets(1).Cells(zeile, 1).Value  
    Else  
        MsgBox "Bitte geben Sie eine Zahl ein"  
    End If  
End Sub
```



# Übung

- Erstellen Sie ein Programm, welches mit einer `InputBox` nach einer Artikelnummer fragt und danach die gewünschten Artikelinformationen in einer `MsgBox` ausgibt

	A	B	C
1	1	Bleistifte (5 Stk)	0,50 €
2	2	Buntstifte (10 Stk)	2,50 €
3	3	DIN A4 Papier (500 Blatt)	12,20 €
4	4	Klarsichtfolien (50 Stk)	3,49 €
5	5	Klebeband (1 Stk)	0,99 €
6			
7			



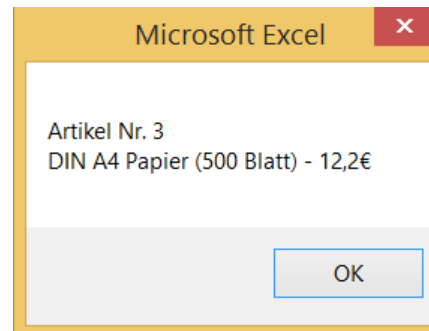
Microsoft Excel

Bitte geben Sie die Artikelnummer ein

OK

Abbrechen

3



Microsoft Excel

Artikel Nr. 3  
DIN A4 Papier (500 Blatt) - 12,2€

OK



# Lösung

```
Sub Artikelsuche()  
    Dim eingabe As Variant  
    Dim artikelname As String  
    Dim artikelpreis As String  
  
    eingabe = InputBox("Bitte geben Sie die Artikelnummer ein")  
    If IsNumeric(eingabe) Then  
        artikelname = Worksheets(1).Range("B" & eingabe).Text  
        artikelpreis = Worksheets(1).Range("C" & eingabe).Text  
  
        MsgBox "Artikel Nr. " & eingabe & vbCrLf & _  
            artikelname & " - " & artikelpreis  
    Else  
        MsgBox "Bitte geben Sie eine Artikelnummer ein"  
    End If  
End Sub
```



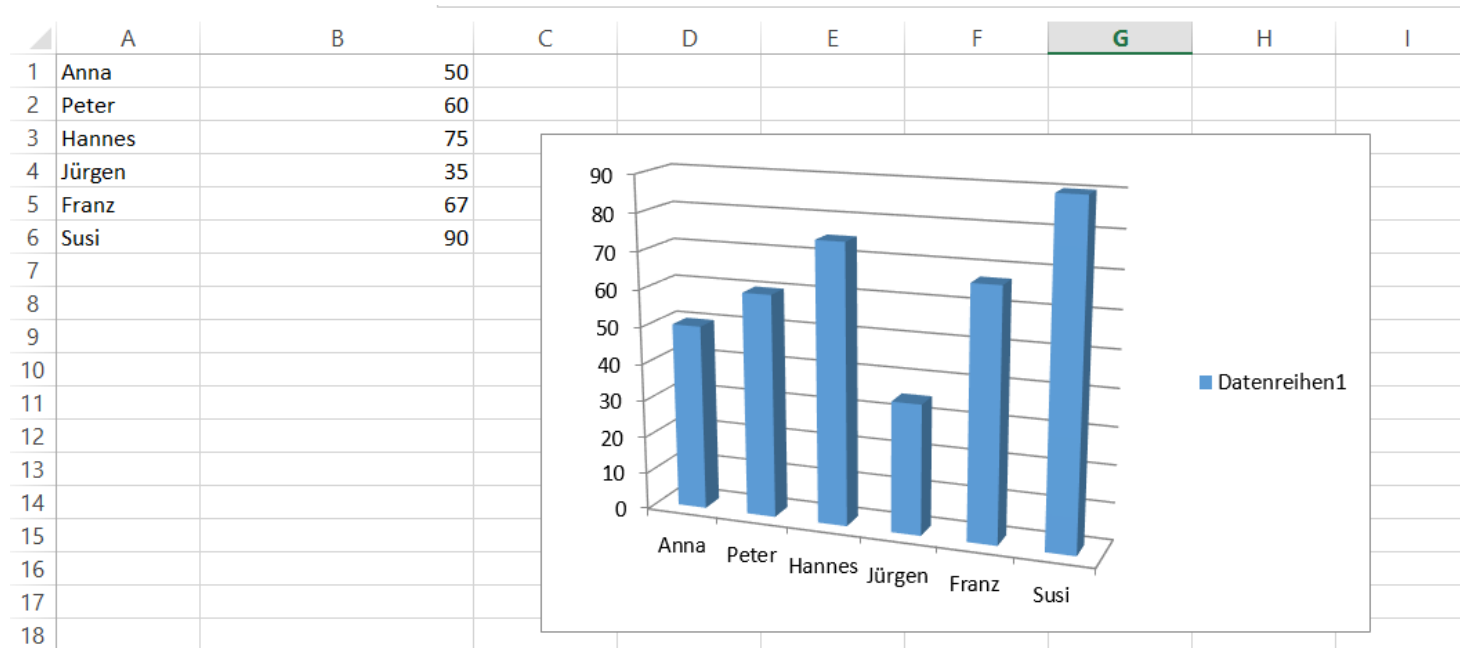
# Arbeiten mit Diagrammen

- Diagramme sind über die `Charts`-Auflistung aufrufbar
  - `Add`: Hinzufügen
  - `Delete`: Löschen
  - `Move`: Verschieben
- Wichtige Eigenschaften/Methoden eines Diagramms (`Chart`):
  - `SetSourceData`: Festlegen der Datenquelle
  - `ChartType`: Diagrammtyp
  - `Location`: Die Position des Diagramms



# Beispiel

```
Sub Diagramm()  
    Charts.Add ' Erstellt ein leeres Diagramm in Excel  
    ActiveChart.SetSourceData Worksheets(1).Range("A1:B6")  
  
    ActiveChart.ChartType = xl3DColumnClustered ' Diagrammtyp  
    ActiveChart.Location xlLocationAsObject, "Tabelle1"  
End Sub
```



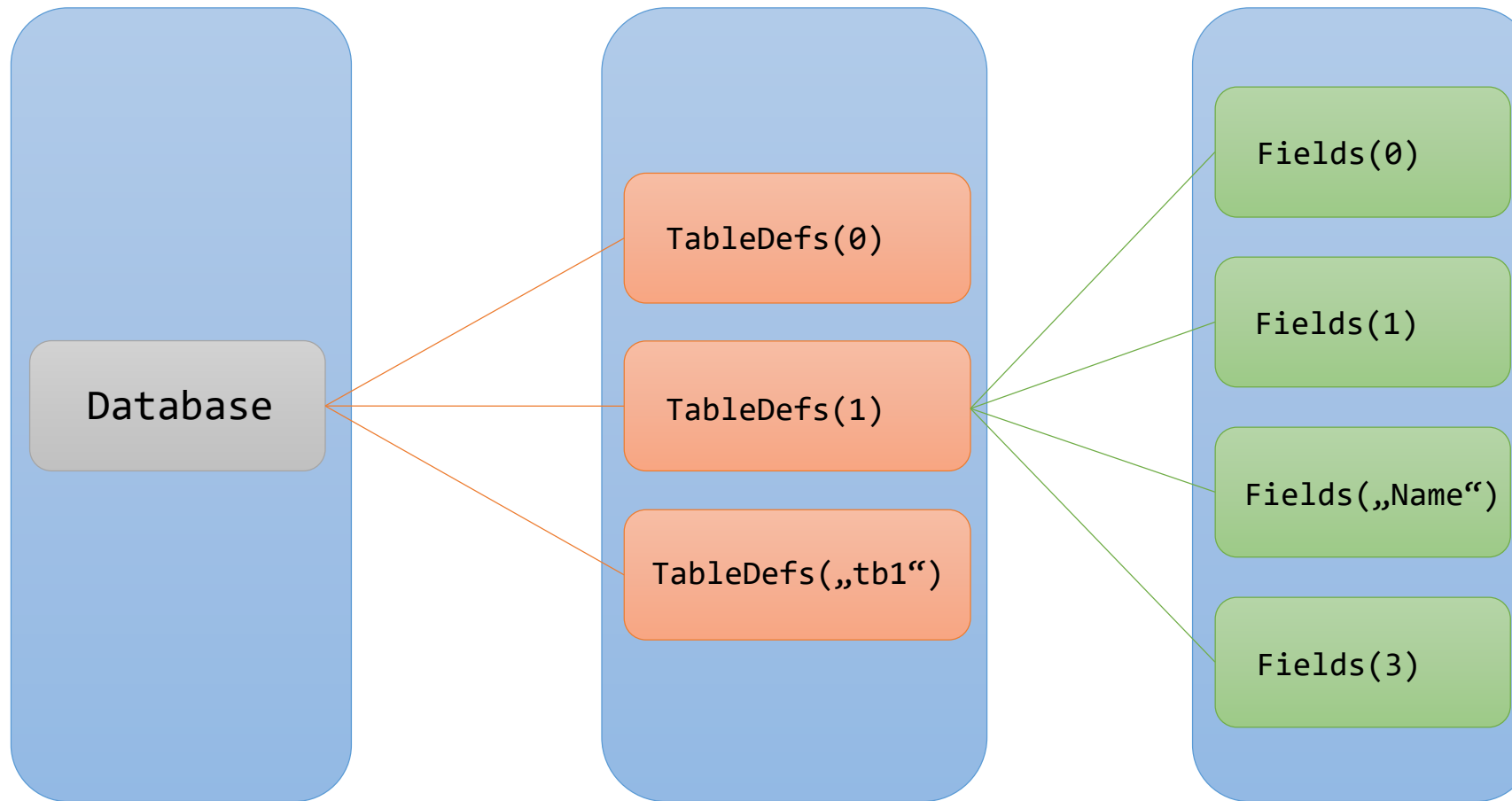


# Das Access-Objektmodell

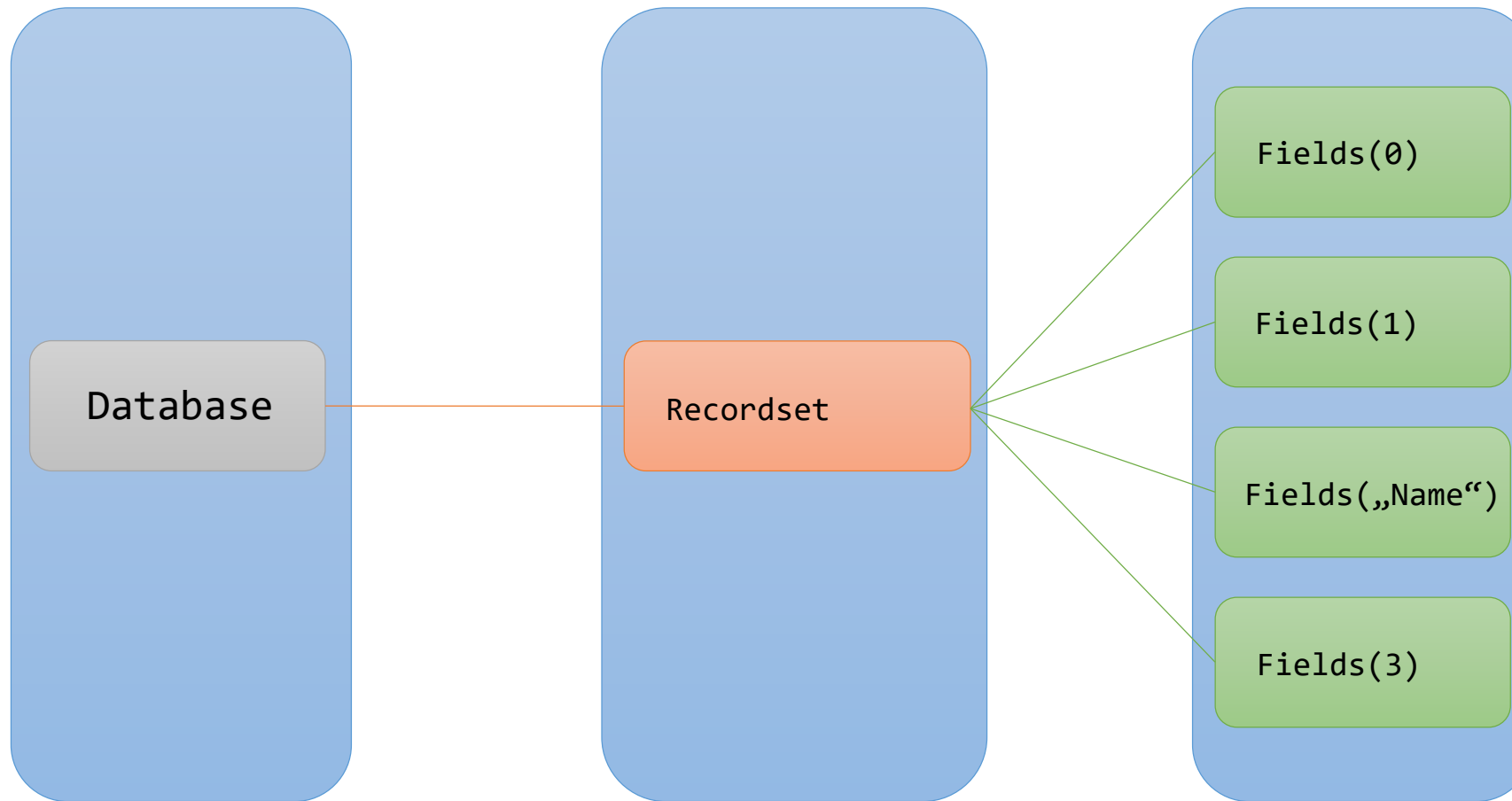
- Andere Funktionsweise als bei anderen Office-Anwendungen
  - Microsoft Office 14.0 Access Database Engine Object
    - Im Programm: DAO (Data Access Objects)
- Oberstes Objekt: Database
  - Beinhaltet Tabellen und Abfragen, keine Werte !
- Zugriff auf den Inhalt einer Tabelle: Recordset



# Objektmodell für die Struktur



# Objektmodell für den Inhalt



# Mit Datenbanken und Tabellen arbeiten

- Methoden des DBEngine-Objekts:
  - OpenDatabase: Öffnet eine existierende DB
  - CreateDatabase: Erstellt eine neue DB
- Methoden und Eigenschaften des Database-Objekts:
  - Name: Name der Datenbank
  - Close: Schließt die Datenbank
  - OpenRecordset: Erstellt ein neues Recordset-Objekt



# Datensätze auslesen

- Zugriff erfolgt über internen Datensatzzeiger (Cursor)
- `Fields`-Auflistung für einzelne Felder (Indizierung)
- Methoden zur Bewegung des Cursors:
  - `MoveFirst`: Positionierung auf ersten Datensatz
  - `MoveNext`: Positionierung auf nächsten Datensatz
  - `MovePrevious`: Positionierung auf vorherigen Datensatz
  - `MoveLast`: Positionierung auf letzten Datensatz
  - `Move`: Verschiebung um gewisse Anzahl an Datensätzen
- Weitere wichtige Eigenschaften:
  - `EOF`: Gibt an, ob sich Cursor hinter letztem Datensatz befindet
  - `BOF`: Gibt an, ob sich Cursor vor erstem Datensatz befindet



# Beispiel

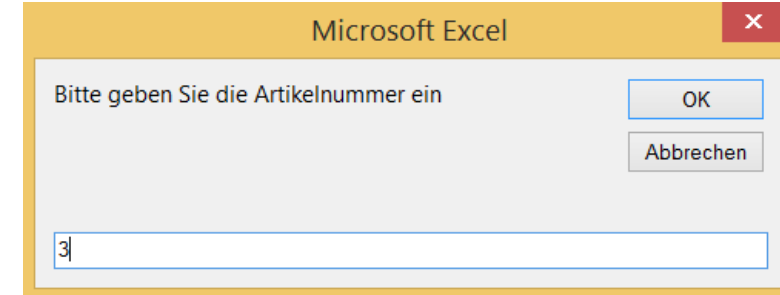
```
Sub EinfacherZugriff()  
    Dim rs As Recordset  
  
    Set rs = CurrentDb.OpenRecordset("Person")  
    rs.MoveFirst ' Springt zum ersten Datensatz  
    Debug.Print rs.Fields("Vorname").Value  
    rs.MoveLast  ' Springt zum letzten Datensatz  
    Debug.Print rs.Fields("Vorname").Value  
End Sub
```



# Übung

- Erstellen Sie ein Programm, welches mit einer `InputDialog` nach einer Artikelnummer fragt und danach die gewünschten Artikelinformationen in einer `MessageBox` ausgibt

ID	Artikel	Preis
1	Bleistifte (5 Stk)	€ 0,50
2	Buntstifte (10 Stk)	€ 2,50
3	DIN A4 Papier (500 Blatt)	€ 12,20
4	Klarsichtfolien (50 Stk)	€ 3,49
5	Klebeband (1 Stk)	€ 0,99



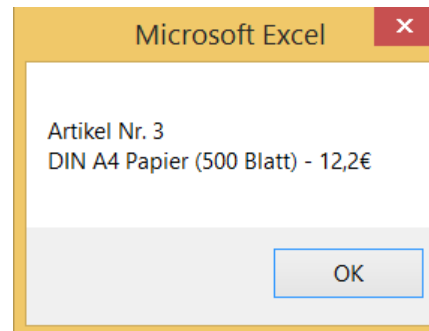
Microsoft Excel

Bitte geben Sie die Artikelnummer ein

OK

Abbrechen

3



Microsoft Excel

Artikel Nr. 3  
DIN A4 Papier (500 Blatt) - 12,2€

OK



# Lösung

```
Sub ArtikelsucheAccess()  
    Dim eingabe As Variant  
    Dim gefunden As Boolean  
    Dim rs As Recordset  
  
    gefunden = False  
    eingabe = InputBox("Bitte geben Sie die Artikelnummer ein")  
    If IsNumeric(eingabe) Then  
        Set rs = CurrentDb.OpenRecordset("Lager")  
        rs.MoveFirst  
        Do Until rs.EOF = True  
            If rs.Fields("ID").Value = CInt(eingabe) Then  
                MsgBox "Artikel Nr. " & eingabe & vbCrLf & _  
                    rs.Fields("Artikel").Value & " - " & rs.Fields("Preis").Value & "€"  
                gefunden = True  
                Exit Do  
            End If  
            rs.MoveNext  
        Loop  
    Else  
        MsgBox "Bitte geben Sie eine gültige Artikelnummer ein"  
    End If  
  
    If Not gefunden Then  
        MsgBox "Ihr Artikel wurde leider nicht gefunden"  
    End If  
End Sub
```





# Datensätze filtern

- Optimierung:  
Eine Abfrage öffnen, um nicht alle Datensätze mit einer If-Struktur abzuprüfen

```
Dim rs As Recordset
Set rs = CurrentDb.OpenRecordset("SELECT * FROM Person WHERE Vorname = 'Tom'", dbOpenDynaset)

Do Until rs.EOF
    Debug.Print rs.Fields("Vorname").Value
    rs.MoveNext
Loop
```



# Kommunikation zwischen Office-Anwendungen



# Integrierte Office-Automatisierung

- Typen der Integration:
  - Funktionen anderer Office-Anwendungen nutzen
  - Zugriff auf Daten außerhalb einer Office-Anwendung
- Möglichkeiten für Zugriff auf Daten anderer Office-Anwendungen:
  - Zwischenablage
  - COM-Schnittstelle
  - ODBC
  - ADO/DAO
  - DDE
- Client/Server-Prinzip



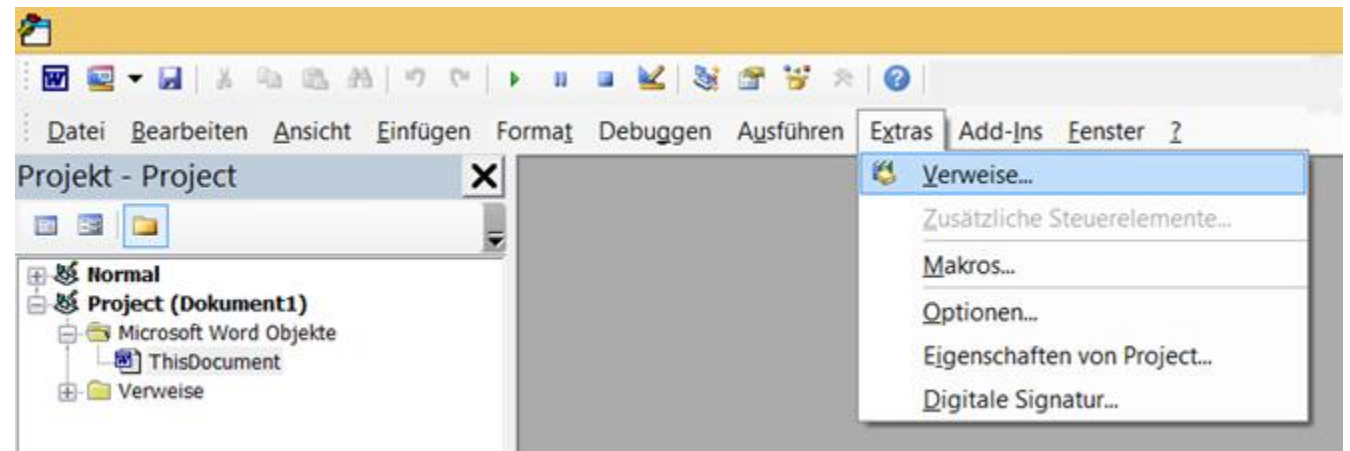
# Technische Grundlagen

- COM
  - Basisprotokoll zur Ansteuerung anderer Anwendungen
- OLE-Automatisierung
  - Spezieller Rahmen für die Einbettung verschiedener Office-Anwendungen
- OCX/ActiveX
  - Komponente, die sich in andere Anwendungen einbinden lässt
- DCOM
  - Rechnerübergreifendes COM
- COM+
  - Erweiterung von COM (Administration)
- Verwendung von COM, ActiveX und OLE wird allgemein als „Automation“ bezeichnet



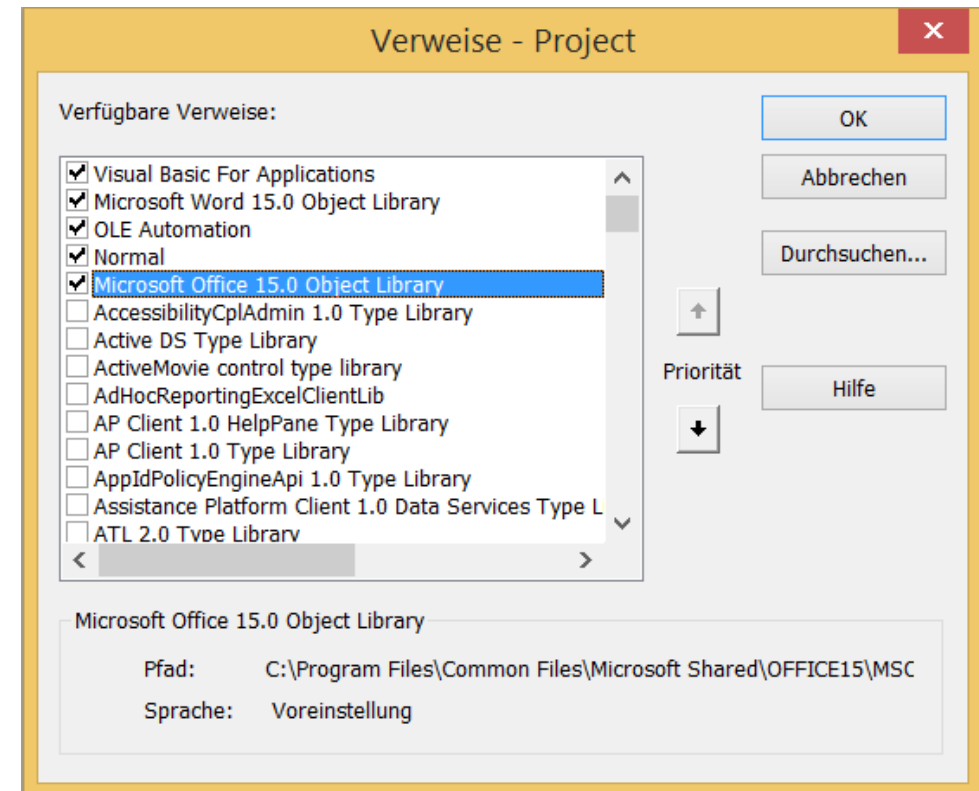
# Verweise hinzufügen

- Über Verweise wird die Kommunikation mit über COM ansprechbare Objektbibliotheken möglich
- Visual Basic Editor öffnen
- Im Menü auf Extras -> Verweise klicken



# Verweise hinzufügen

- Anzeige aller Objektbibliotheken
- Auswahl gewünschter Bibliotheken
- Hinzufügen einzelner Verweise



# Objektvariablen

- Auch als Instanz bezeichnet
- Frühe Bindung durch Angabe des Datentyps:
  - Vorteile: Schneller und Benutzung der Eingabehilfen möglich
  - Nachteile: Unflexibel, da nur Objekte eines bestimmten Typs
- Späte Bindung durch Weglassen des Datentyps oder Verwendung des Datentyps `Object`:
  - Vorteile: Flexibel, jede Art von Objekt möglich
  - Nachteile: Langsam und keine Intellisense-Unterstützung

```
Dim appXL As Excel.Application ' frühes Binden
```

```
Dim appXL As Object
```

```
'...
```

```
Set appXL = Excel.Application ' spätes Binden
```



# Objektvariablen

Anwendung	Objektklasse	Wiedergegebenes Objekt
Access	Access.Application	Access-Anwendung
Access	Access.Form	Access-Formular
Excel	Excel.Application	Excel-Anwendung
Excel	Excel.Sheet	Excel-Arbeitsmappe
Word	Word.Application	Word-Anwendung
PowerPoint	PowerPoint.Application	PowerPoint-Anwendung





# Objektvariablen

- Erstellen einer Instanz: `CreateObject`
  - Gibt im Fehlerfall einen Fehlercode zurück
- `GetObject` liefert einen Verweis auf bereits gestartete Anwendung
  - Nur 1 Instanz im Hintergrund -> minimaler Arbeitsspeicherverbrauch
- Instanziierung auch über `New` möglich
  - Keine Rückgabe eines Fehlercodes im Fehlerfall



# Objekte schließen und freigeben

- Close-Methode
  - Schließen von Dokumenten innerhalb einer Anwendung
- Quit-Methode
  - Schließen einer Anwendung
- Nothing
  - Freigabe des verwendeten Speichers

```
Dokumentvariable.Close      ' Schließt ein Dokument  
Appvariable.Quit            ' Schließt eine Anwendung  
Set Appvariable = Nothing   ' gibt den belegten Speicherplatz frei
```



# Übung

- Schreiben Sie ein Programm, welches die Rechnungssumme aus einer Excel-Arbeitsmappe ausliest und in ein Word-Dokument hineinschreibt
- Optional:  
Um Arbeitsspeicher zu sparen und eine schnellere Ausführung der Prozedur zu erreichen, soll zuerst geprüft werden, ob Excel bereits gestartet wurde

	A	B
1	23,40 €	
2	120,00 €	
3	45,30 €	
4	22,90 €	
5	60,50 €	
6	99,99 €	
7	34,20 €	
8		
9	Summe	406,29 €

€ 406,38 €|



# Lösung

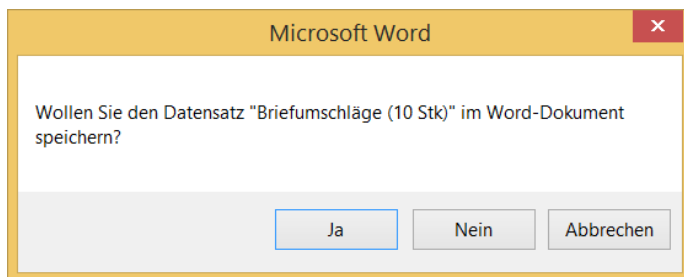
```
Sub Summe_aus_Excel()  
    On Error GoTo Fehler  
    Dim appXL As Excel.Application  
    Set appXL = GetObject(Class:="Excel.Application")  
    On Error GoTo 0  
  
    appXL.Workbooks.Open ActiveDocument.Path & "\Werte.xlsx"  
    Selection.InsertAfter appXL.ActiveSheet.Range("B9").Text  
  
    appXL.Quit  
    Set appXL = Nothing  
    Exit Sub  
  
Fehler:  
    If Err = 429 Then  
        Set appXL = CreateObject("Excel.Application")  
    Else  
        Err.Raise Err.Number ' originaler Fehler  
    End If  
  
    Resume Next  
End Sub
```



# Übung

- Programmieren Sie ein Makro, welches eine Tabelle aus einer Excel-Datei ausliest und die Daten in einem Word-Dokument speichert
- Das Programm fragt bei jedem Datensatz den Benutzer, ob er diesen in das Word-Dokument übertragen will

```
Dim antwort As VbMsgBoxResult  
antwort = MsgBox("Frage", vbYesNoCancel)  
If antwort = vbYes Then  
    ...  
ElseIf antwort = vbNo Then  
    ....  
End If
```



1	Bleistifte	0,20 €
2	Briefumschläge (10 Stk)	2,56 €
3	DIN A4 Papier (500 Blatt)	12,19 €
4	Post-It Notes 656	5,00 €
5	Post-It Notes 657	10,40 €
6	Buntstifte	0,50 €
7	Büroklammern (100 St.)	2,50 €
8	Locher	8,95 €
9	Bildtrommeln	127,90 €
10	CD-R	1,20 €
11	Laser Labels 6590 (25 Bla	18,89 €
12	Toner	85,90 €
13	Drucker-Tinte	22,00 €



# Lösung

```
Sub Produkte_Auslesen()  
    Dim appXL As Excel.Application  
    Set appXL = New Excel.Application ' Alternative  
    Dim zähler As Integer  
    Dim ausgabe As VbMsgBoxResult  
  
    zähler = 1  
    Set appXL = CreateObject("Excel.Application")  
    appXL.Workbooks.Open ActiveDocument.Path & "\Produkte.xlsx"  
  
    Do Until appXL.Range("A" & zähler).Value = ""  
        ausgabe = MsgBox("Wollen Sie den Datensatz " & appXL.Range("B" & zähler).Value _  
                        & " im Word-Dokument speichern ?", vbYesNoCancel)  
        If ausgabe = vbYes Then  
            ActiveDocument.Range.InsertAfter appXL.Range("B" & zähler).Value & " - " & appXL.Range("C" & zähler).Text & vbCrLf  
        ElseIf ausgabe = vbCancel Then  
            Exit Do  
        End If  
        zähler = zähler + 1  
        Application.ScreenRefresh ' Optional  
    Loop  
    appXL.Quit  
    Set appXL = Nothing  
End Sub
```



# Grafische Benutzeroberflächen



# Dialoge

- Vordefinierte Dialoge
  - z.B. `InputBox`, `MsgBox`, ...
- Integrierte Dialoge
  - z.B. Speichern, Datei Öffnen, ...
- UserForm-Dialoge
  - „VBA-Formulare“
- Formulare in Access
  - Teil einer Datenbank, werden direkt von Access verwaltet
- Dokumente als Formulare
  - z.B. in Word-Dokumenten enthaltene Steuerelemente





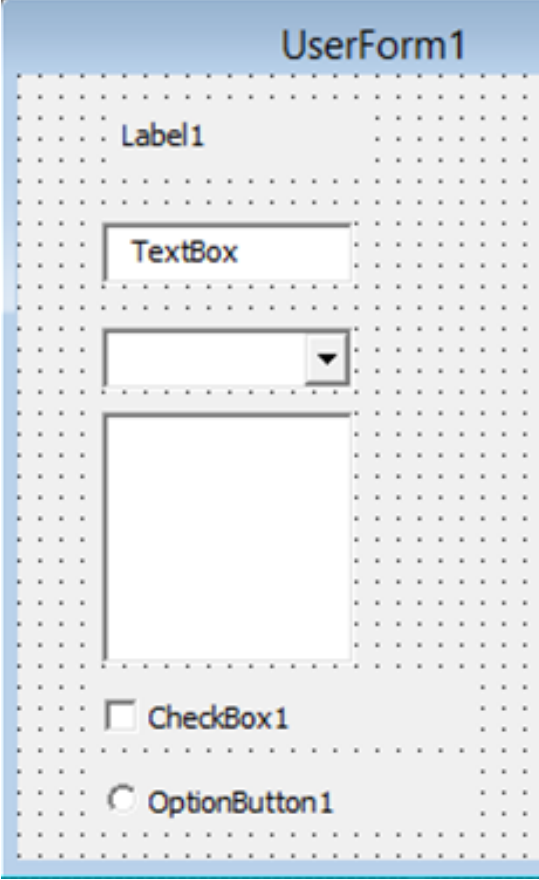
# Erstellen von UserForms

- Projektextplorer
  - Einfügen -> UserForm
- Toolbox
  - Hinzufügen von Steuerelementen
  - Steuerelemente mit gedrückter Maustaste auf das Formular ziehen
- Löschen von Steuerelementen
  - „Entf“
- Eigenschaftsfenster
  - Steuerung des Verhaltens des Elements



# Steuerelemente

- Bezeichnungsfeld (Label)
  - Beschriftung
- Textfeld (TextBox)
  - Feld zur Eingabe von Daten
- Kombinationsfeld (ComboBox)
  - Kombination aus Listen- und Textfeld
- Listenfeld (ListBox)
  - Mehrzeiliges Feld zur Anzeige und Auswahl einzelner Einträge
- Kontrollfeld (CheckBox)
  - Feld zur Aktivierung/Deaktivierung einer Option
- Optionsfeld (OptionButton)
  - Siehe Kontrollfeld.
  - Nur eines pro Gruppe aktivierbar

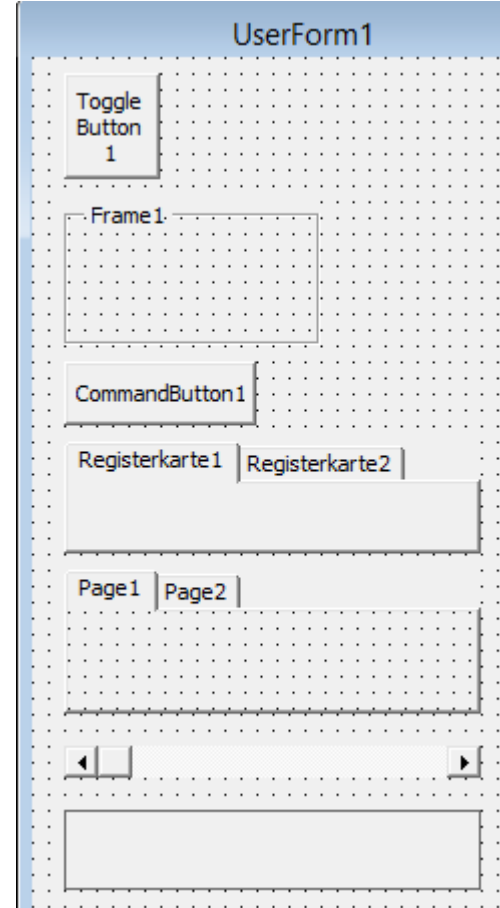


The image shows a screenshot of a Windows UserForm titled "UserForm1". The form is set against a dotted grid background. It contains the following controls from top to bottom: a label "Label1", a text box "TextBox", a combobox (a text box with a dropdown arrow), a list box (a large empty rectangular area), a checkbox labeled "CheckBox1", and a radio button labeled "OptionButton1".



# Steuerelemente

- Umschaltfeld (ToggleButton)
  - Schaltfläche zum Umschalten eines Zustandes
- Rahmen (Frame)
  - Gruppierung von Steuerelementen
- Befehlsschaltfläche (CommandButton)
  - Schaltfläche zum Auslösen einer Aktion
- Register (TabStrip)
  - Registergruppe für zusammenhängende Bereiche
- Multiseiten (MultiPages)
  - Wie Register, aber mit mehreren Seiten für unterschiedliche Bereiche
- Bildlaufleiste (ScrollBar)
  - Scrollen
- Anzeigefeld (Image):
  - Anzeige einer Grafik



# Programmieren mit UserForms

- Reagieren auf Ereignisse mit Ereignisprozeduren
  - Einfache Erzeugung mithilfe des Codeeditors
- Anzeige eines UserForms mit der `Show`-Methode
  - Ausblenden mit `Hide`
  - Vollständig schließen mit `Unload`
- Gemeinsame Eigenschaften:
  - `Name`: Name des Steuerelements (eindeutigen Identifikation)
  - `Left/Top`: Position des Steuerelements
  - `Width/Height`: Größe des Steuerelements
  - `Enabled`: Gibt an, ob das Steuerelement aktiviert ist



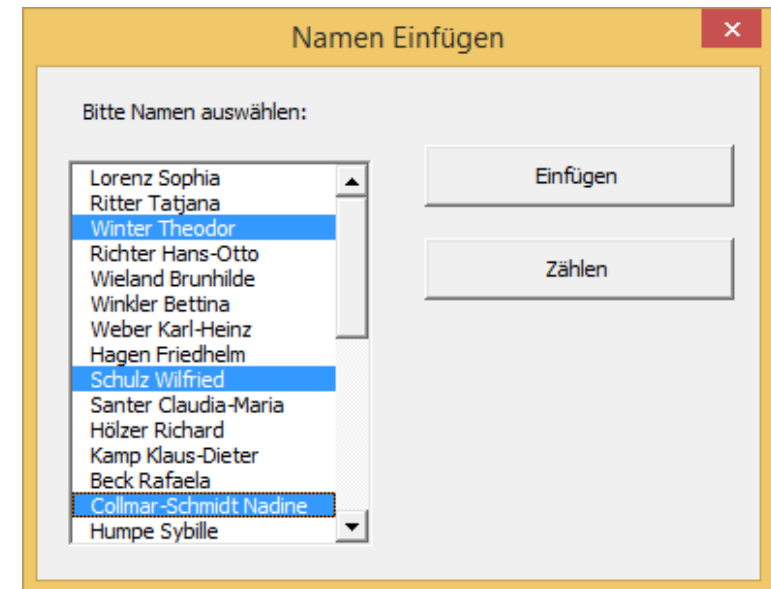
# Steuerelemente programmieren

- Eingabefelder/Kontrollfelder:
  - `Value`: Wert
  - `Change`: Wertänderung
- Schaltflächen:
  - `Caption`: Aufschrift
  - `Click`: Ereignis, das bei einem Mausklick ausgelöst wird
- Listen:
  - `AddItem/RemoveItem`: Eintrag hinzufügen/löschen
  - `ListCount`: Anzahl an Einträgen in Liste
  - `ListIndex`: Ausgewählter Eintrag als Index
  - `Selected`: Ausgewählter Eintrag



# Übung

- Erstellen Sie ein User-Formular, welches die Vor- und Nachnamen einzelner Personen aus einer Excel-Liste ausliest. Die Namen werden in einer `ListBox` angezeigt.
- Wenn man mehrere Namen in der `ListBox` anklickt und daraufhin den Button „Einfügen“ klickt, sollen die ausgewählten Namen in das Word-Dokument übertragen werden
- Mit einem Klick auf den Button „Zählen“ gibt das Programm die Anzahl der Namen in der Liste aus



The screenshot shows a VBA UserForm titled "Namen Einfügen" with a yellow title bar and a red close button. Inside the form, there is a label "Bitte Namen auswählen:" above a `ListBox` containing a list of names. The names are: Lorenz Sophia, Ritter Tatjana, Winter Theodor, Richter Hans-Otto, Wieland Brunhilde, Winkler Bettina, Weber Karl-Heinz, Hagen Friedhelm, Schulz Wilfried, Santer Claudia-Maria, Hölzer Richard, Kamp Klaus-Dieter, Beck Rafaela, Collmar-Schmidt Nadine, and Humpe Sybille. The names "Winter Theodor", "Schulz Wilfried", and "Collmar-Schmidt Nadine" are currently selected. To the right of the `ListBox` are two buttons: "Einfügen" and "Zählen".

Name	Status
Lorenz Sophia	Not Selected
Ritter Tatjana	Not Selected
Winter Theodor	Selected
Richter Hans-Otto	Not Selected
Wieland Brunhilde	Not Selected
Winkler Bettina	Not Selected
Weber Karl-Heinz	Not Selected
Hagen Friedhelm	Not Selected
Schulz Wilfried	Selected
Santer Claudia-Maria	Not Selected
Hölzer Richard	Not Selected
Kamp Klaus-Dieter	Not Selected
Beck Rafaela	Not Selected
Collmar-Schmidt Nadine	Selected
Humpe Sybille	Not Selected

# Lösung

```
Private Sub UserForm_Activate()  
    Dim appAC As New Access.Application ' Alternative  
    Dim rs As Variant ' Recordset oder Recordset2  
  
    With appAC.DBEngine.OpenDatabase(ActiveDocument.Path & "\\Gehalt.accdb")  
        Set rs = .OpenRecordset("Mitarbeiter")  
        rs.MoveFirst  
  
        Do Until rs.EOF  
            ListBoxNamen.AddItem rs.Fields("Name").Value & " " & rs.Fields("Vorname").Value  
            rs.MoveNext  
        Loop  
    End With  
    appAC.Quit  
    Set appAC = Nothing  
End Sub  
  
Private Sub CommandButtonEinfügen_Click()  
    Dim zähler As Integer  
  
    For zähler = 0 To ListBoxNamen.ListCount - 1  
        If ListBoxNamen.Selected(zähler) Then  
            ActiveDocument.Range.InsertAfter ListBoxNamen.List(zähler) & vbCrLf  
            Application.ScreenRefresh ' Optional  
        End If  
    Next  
End Sub  
  
Private Sub CommandButtonZählen_Click()  
    MsgBox ListBoxNamen.ListCount & " Namen geladen.", vbOKOnly + vbInformation  
End Sub
```



# Datenbankzugriffe





# Grundlagen

- Zwei Objektmodelle für den Datenzugriff:
  - DAO (Data Access Objects)
  - ADO (ActiveX Data Objects)
- ADO - Verweis auf:
  - Microsoft ActiveX Data Objects 6.0 Library
  - Microsoft ActiveX Data Objects Recordset 6.0 Library
- Objekte in ADO:
  - `Connection`: Verbindung mit Datenquelle
  - `Command`: Befehl zur Ausführung in Datenquelle
  - `Recordset`: Stellt mehrere Datensätze (Tabelle)
  - `Parameter`: Parameter für Befehl
  - `Field`: Repräsentiert ein Datenfeld



# Ablauf eines Datenzugriffs

- Verbindung herstellen
  - `Connection`-Objekt
- Befehle an die Datenquelle senden
  - `Command`-Objekt
- Zurückgegebene Datensätze zwischenspeichern
  - `ResultSet`-Objekt
- Datenquelle aktualisieren
  - `ResultSet`-Objekt
- Ggf. Auswertung von Fehlern
  - `Error`-Objekt
- Verbindung schließen
  - `Connection`-Objekt



# Verbindung herstellen

- Neues `Connection`-Objekt erstellen
- `Provider`-Eigenschaft festlegen
- Verbindung mit der `Open`-Methode öffnen
  - Datenbankpfad übergeben
  - Alternative: Angabe eines `ConnectionString`
- Verbindung mit der `Close`-Methode schließen
- `Connection`-Objekt auf `Nothing` setzen



# Tabellen öffnen

- Neues `Recordset`-Objekt erstellen
- Über die `Open`-Methode die gewünschte Tabelle öffnen
  - SQL-Kommando, Tabellename oder Abfragen als erstes Argument
  - Bestehende Verbindung übergeben
  - Cursortyp und Sperrtyp angeben



# Datensätze auslesen

- Zugriff erfolgt über internen Datensatzzeiger (Cursor)
- `Fields`-Auflistung für einzelne Felder (Indizierung)
- For-Each-Schleife zum Durchlaufen der Felder
- Methoden zur Bewegung des Cursors:
  - `MoveFirst`: Positionierung auf ersten Datensatz
  - `MoveNext`: Positionierung auf nächsten Datensatz
  - `MovePrevious`: Positionierung auf vorherigen Datensatz
  - `MoveLast`: Positionierung auf letzten Datensatz
  - `Move`: Verschiebung um gewisse Anzahl an Datensätzen
- Weitere Eigenschaften:
  - `EOF`: Gibt an, ob sich Cursor hinter letztem Datensatz befindet
  - `BOF`: Gibt an, ob sich Cursor vor erstem Datensatz befindet



# Datensätze verarbeiten

- Suche nach Datensätzen
  - Find-Methode
  - Mustersuche, SQL-ähnliche Syntax
- Bearbeiten eines Datensatzes
  - Wertzuweisung an ein bestehendes Feld
- Hinzufügen eines neuen Datensatzes
  - AddNew-Methode
- Löschen eines Datensatzes
  - Delete-Methode
- Datenquelle aktualisieren
  - Update-Methode
- Änderungen an Datensätzen nicht in der Datenquelle speichern
  - CancelUpdate-Methode



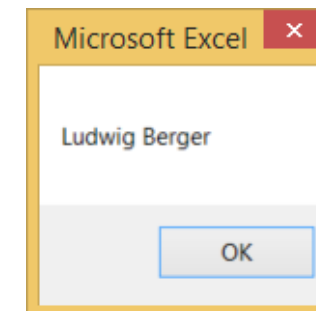
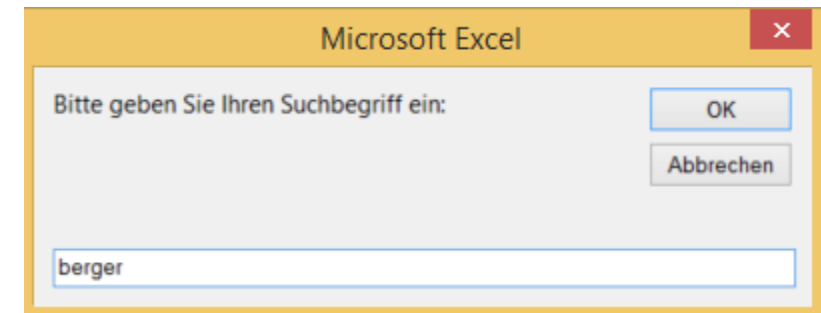
# Beispiel

```
Sub ADO_Datensätze_Auslesen()  
    Dim ADOConnection As ADODB.Connection  
    Dim ADOTabelle As ADODB.Recordset  
    Dim Index As Integer  
  
    Index = 0  
    Set ADOConnection = New ADODB.Connection  
    ADOConnection.Provider = "Microsoft.ACE.OLEDB.12.0"  
    ADOConnection.Open ActiveWorkbook.Path & "\Gehalt.accdb"  
    Set ADOTabelle = New ADODB.Recordset  
    ADOTabelle.Open "Mitarbeiter", ADOConnection  
  
    Do Until ADOTabelle.EOF = True  
        Index = Index + 1  
        Range("A" & Index).Value = ADOTabelle!VorName & " " & ADOTabelle!Name  
        ADOTabelle.MoveNext  
    Loop  
  
    ADOConnection.Close  
    Set ADOConnection = Nothing  
End Sub
```



# Beispiel

```
Sub ADO_Datensätze_Durchsuchen()  
    Dim ADOConnection As ADODB.Connection  
    Dim ADOTabelle As ADODB.Recordset  
    Dim Index As Integer  
    Dim Eingabe As String  
  
    Eingabe = InputBox("Bitte geben Sie Ihren Suchbegriff ein:")  
    Index = 0  
    Set ADOConnection = New ADODB.Connection  
    ADOConnection.Provider = "Microsoft.ACE.OLEDB.12.0"  
    ADOConnection.Open ActiveWorkbook.Path & "\Gehalt.accdb"  
    Set ADOTabelle = New ADODB.Recordset  
    ADOTabelle.Open "Mitarbeiter", ADOConnection, adOpenDynamic  
  
    With ADOTabelle  
        .Find "Name Like '*' & Eingabe & '*'"  
        If .EOF Then  
            MsgBox "Es wurde kein passender Datensatz gefunden!", vbInformation  
        Else  
            MsgBox !VorName & " " & !Name  
        End If  
    End With  
  
    ADOConnection.Close  
    Set ADOConnection = Nothing  
End Sub
```





# Integrierte Lösungen mit Word



# Automation mit Word

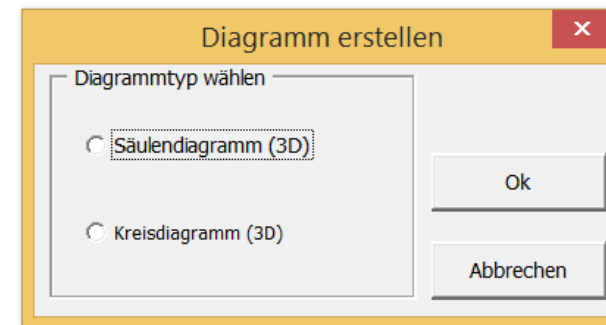
- Umfangreiche Möglichkeiten zur Textverarbeitung
- Import von Daten aus Excel/Access ist oft notwendig
- Word als Client von Excel:
  - Tabellen auslesen und Formeln berechnen
  - Diagrammfunktion verwenden
- Word als Client von Access:
  - Ausgabe/Eingabe von Werten in Datenbanken
- Word als Client von Outlook:
  - Adressbücher auslesen
  - Terminfunktion von Outlook in Word verwenden



# Übung

- Erstellen Sie ein Programm, welches Werte aus den Spalten einer Word-Tabelle liest und diese Daten in einer versteckten Excel-Instanz zu einem Diagramm verarbeitet.
- Das Diagramm wird am Ende in der Zwischenablage gespeichert und kann danach an einer beliebigen Stelle im Dokument eingefügt werden.

Lager	Warenbestand
Köln	450
Dortmund	612
Hamburg	492
München	630
Berlin	470
Leipzig	210
Mainz	380



# Diagrammfunktion aus Excel nutzen

- Tabellenfunktionen von Word ist oft nicht ausreichend
  - fehlende Diagrammfunktionen
  - keine Tabellenkalkulation
- Excel-Objektbibliothek einbinden
- Ggf. über UserForm Einstellungen für das Diagramm vorgeben
- Objekt der Excel-Anwendung erstellen
- Word-Tabelle kopieren
  - Copy-Methode
- Workbook erstellen und kopierte Tabelle in ein Worksheet einfügen
  - Paste-Methode



# Diagrammfunktion aus Excel nutzen

- Diagramm hinzufügen
  - Charts-Auflistung
- Typ, Datenquelle und Position festlegen
  - ActiveChart-Eigenschaft
- Diagramm kopieren
  - Copy-Methode
- Diagramm in Word an einer beliebiger Stelle einfügen
  - Paste-Methode
- Einfacher Datenaustausch über Zwischenablage:
  - Copy: Kopieren
  - Paste: Einfügen
  - Cut: Ausschneiden



# Lösung

```
Private Sub CommandButtonOk_Click()  
    Dim appXL As Excel.Application  
    Dim diagrammtyp As Integer  
  
    Set appXL = CreateObject("Excel.Application")  
  
    Selection.Tables(1).Select  
    Selection.Copy  
  
    If UserFormDiagramm.OptionButtonSäulendiagramm = True Then  
        diagrammtyp = xl3DColumn  
    ElseIf UserFormDiagramm.OptionButtonKreisdiagramm = True Then  
        diagrammtyp = xl3DPie  
    End If  
  
    With appXL  
        .Workbooks.Add  
        .ActiveSheet.Paste  
        .Charts.Add  
        With ActiveChart  
            .ChartType = diagrammtyp  
            .SetSourceData appXL.Sheets("Tabelle1").UsedRange  
            .Location xlLocationAsNewSheet  
            .ChartArea.Copy  
        End With  
        .ActiveWorkbook.Close False ' Schließen ohne Speichern  
        .Quit  
    End With  
  
    Set appXL = Nothing  
    MsgBox "Das Diagramm wurde in die Zwischenablage kopiert.", vbInformation  
End Sub  
  
Private Sub CommandButtonAbbrechen_Click()  
    UserFormDiagramm.Hide  
End Sub
```



# Integrierte Lösungen mit Excel



# Automation mit Excel

- Umfangreiche Möglichkeiten zur Analyse und Berechnung von Daten in Tabellenform
- Excel als Client von Word:
  - Spezielle Textverarbeitungsfunktionen benutzen
  - Flexiblere Druckfunktion verwenden
- Excel als Client von Outlook:
  - Verwenden der E-Mailfunktion
  - Adressbücher auslesen





# Übung

- Erstellen Sie ein Programm, welches die Vor- und Nachnamen von Personen aus einer Excel-Liste in ein Formular einliest.
- Der gewählte Name der Person und die entsprechende Anrede soll nach der Auswahl in das Word-Dokument eingefügt werden.

	A	B	C
1	Tom	Ate	Herr
2	Anna	Nass	Frau
3	Martha	Pfahl	Frau
4	Stein	Bach	Herr
5	Fels	Fluss	Herr
6	Salz	Burger	Herr
7	De	Fault	Frau

Briefanrede

Tom Ate  
Anna Nass  
Martha Pfahl  
Stein Bach  
Fels Fluss  
Salz Burger  
De Fault

Load Excel Sheet

Anrede:  
Guten Morgen, |

Bestätigen

# Lösung

```
Dim Person As String ' Globaler Zwischenspeicher

Private Sub ListBoxPersonen_Click()
    Dim i As Integer
    For i = 0 To ListBoxPersonen.ListCount - 1
        If ListBoxPersonen.selected(i) Then
            Person = ListBoxPersonen.List(i)
        End If
    Next
End Sub

Private Sub CommandButtonLoadExcel_Click()
    Dim Excel As Excel.Application
    Dim zähler As Integer
    zähler = 1

    Set Excel = CreateObject("Excel.Application")
    Excel.Workbooks.Open (ActiveDocument.Path & "\Briefanrede - Liste.xlsx")

    Do While CStr(Excel.ActiveSheet.Cells(zähler, 1)) <> ""
        ListBoxPersonen.AddItem Excel.ActiveSheet.Cells(zähler, 1) & " " _
        & Excel.ActiveSheet.Cells(zähler, 2), ListBoxPersonen.ListCount -
        zähler = zähler + 1
    Loop
    Excel.Quit
    Set Excel = Nothing
End Sub

Private Sub CommandButtonBestätigen_Click()
    Dim Briefanrede As String

    Briefanrede = TextBoxAnrede.Text & Person & vbCrLf
    ActiveDocument.Range.InsertAfter Briefanrede
End Sub
```



# Integrierte Lösungen mit Access



# Automation mit Access

- Erstellung von relationaler Datenbanken mit Benutzeroberflächen
- Access als Client von Excel:
  - Tabellen auslesen und importieren
- Access als Client von Word:
  - Erstellen von Serienbriefen mithilfe von Datenbank-Input
- Zugriffsmöglichkeiten:
  - COM
  - ADO



# Übung

- Erstellen Sie eine Funktion, die in einer Access-Datenbank nach dem Nachnamen einer Person sucht und dessen Daten in die entsprechenden Textmarken eines Word-Dokumentes einträgt.
- Einfügen in eine Textmarke:

```
ActiveDocument.GoTo (What:=wdGoToBookmark, Name:="Textmarkenname").Text = "Mein Text"
```



# Adressen aus Access übernehmen

- Import erfolgt über ADO
- Verbindung mit der Access-Datenbank herstellen
  - `Connection`-Objekt
- Tabelle öffnen
  - `Open`-Methode eines `Recordset`-Objekts
- Nach einem Namen suchen
  - SQL-Abfrage
  - `Find`-Methode
- Datensätze lesen und Adressen in Textmarken einfügen
- Verbindung schließen



```

Private Sub CommandButton1_Click()
    Dim ADOConnection As ADODB.Connection
    Dim ADOTabelle As ADODB.Recordset
    Dim Name As String

    Set ADOConnection = New ADODB.Connection

    Name = InputBox("Nach welchem Nachnamen soll gesucht werden?", "Adresse suchen", "Wenger") ' Vorschlag

    If Name <> "" Then

        ADOConnection.Provider = "Microsoft.ACE.OLEDB.12.0"
        ADOConnection.Open ActiveDocument.Path & "\Adressen.accdb"
        Set ADOTabelle = New ADODB.Recordset

        With ADOTabelle

            .Open "Adressen", ADOConnection, adOpenDynamic, adLockOptimistic
            .Find "Nachname='" & Name & "'" ' Find-Befehl: "Nachname='Wenger'"

            If Not .EOF Then ' Wenn nichts gefunden wird -> EOF
                ActiveDocument.GoTo(wdGoToBookmark, Name:="Adresse").Select

                Selection.InsertAfter .Fields("Vorname") & " " & .Fields("Nachname") & vbCrLf
                Selection.InsertAfter .Fields("Adresse") & vbCrLf
                Selection.InsertAfter .Fields("Land") & " - " & .Fields("PLZ") & " " & .Fields("Ort")

                ActiveDocument.GoTo(wdGoToBookmark, Name:="Anrede").Text = .Fields("Briefanrede") & ", "
            Else
                MsgBox "Die Person befindet sich nicht in der Datenbank"
            End If

            .Close
        End With

        ADOConnection.Close
        Set ADOConnection = Nothing
    End If
End Sub

```

