R|R|Z|N
Regionales Rechenzentrum für Niedersachsen

Leibniz Universität Hannover

Excel – Automatisierung von Arbeitsschritten
Import von Daten

Arbeitsschritte für die Aufzeichnung

- Das Menüband Daten ist geöffnet.
- Klick auf das Symbol *Aus Text* in der Gruppe Externe Daten abrufen.
- In dem Dialog Textdatei importieren wird die Datei „importRTF.csv“ ausgewählt.
- Der Dialog wird mit Hilfe der Schaltfläche *Importieren* geschlossen. Der Textkonvertierungs-Assistent wird automatisch gestartet.

Einstellungen im Textkonvertierungs-Assistent

- Im ersten Schritt wird die Option Getrennt als Dateityp gewählt. Der Dateiursprung wird auf Windows (ANSI) gestellt.
- Im zweiten Schritt wird als Trennzeichen ein Semikolon ausgewählt.
- Im dritten Schritt wird ein Zahlenformat für die verschiedenen Spalten ausgewählt.
- Mit Hilfe der Schaltfläche *Fertig stellen* wird der Assistent geschlossen.
- Anschließend wird die Speicherposition für die einzufügenden Daten mit Hilfe eines Dialogs ermittelt.

Speicherposition festlegen

- Nach dem Schließen des Textkonvertierungs-Assistenten wird mit Hilfe eines Dialogs die Speicherposition der Daten auf einem vorhandenen Arbeitsblatt erfragt. Andere Möglichkeit: Für die Daten wird ein neues Arbeitsblatt angelegt.
- *OK* schließt den Dialog und die Daten werden entsprechend eingefügt.

Arbeitsschritte automatisieren

- Voraussetzung: Die Entwicklertools sind eingeblendet.
- Die Arbeitsschritte werden einmalig mit Hilfe eines Makros aufgezeichnet.
- Nach Beendigung der Aufzeichnung werden die Arbeitsschritte automatisiert über ein Symbol oder Tastenkürzel gestartet.

Entwicklertools einblenden

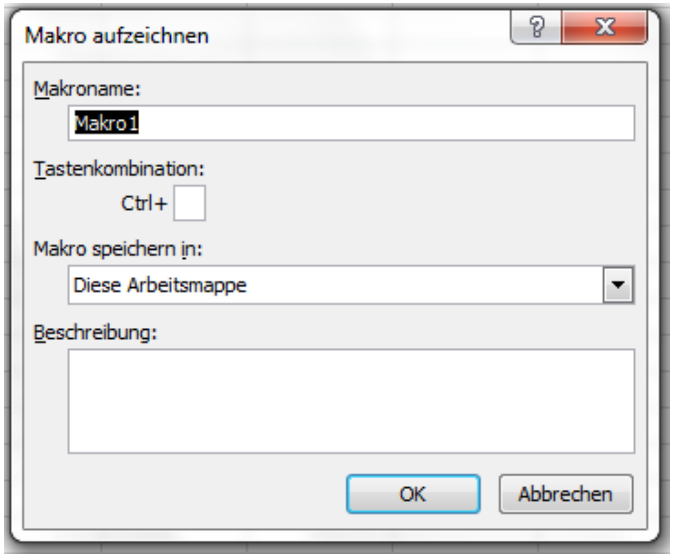
- *Datei – Optionen*.
- *Menüband anpassen*.
- In der rechten Liste Menüband anpassen werden alle Hauptregisterkarten angezeigt. Das Menüband Entwicklertools ist standardmäßig ausgeblendet (Kästchen ist leer).
- Durch einen Mausklick links von der Bezeichnung des Menübandes wird dieses aktiviert. In dem Kästchen wird ein Häkchen angezeigt.

Arbeitsschritte aufzeichnen

- Das Menüband Entwicklertools ist eingeblendet.
- Durch einen Mausklick wird die Aktion *Makro aufzchn.* in der Gruppe Code gestartet.
- Es öffnet sich der Dialog Makro aufzeichnen. In dem Dialog werden die Grundeinstellungen der Aufzeichnung festgelegt. *OK* schließt den Dialog.
- Hinweis: Die Aufzeichnung endet nicht automatisch.

Dialog „Makro aufzeichnen“

- In dem obersten Textfeld wird ein eindeutiger Name für die Aufzeichnung eingegeben.
- Zum Starten des Makros kann ein Buchstabe eingegeben werden. Hinweis: Einige Kombinationen wie <CTRL>+<C> sind belegt.
- Mit Hilfe des Kombinationsfeldes wird der Speicherort des Makros festgelegt. Standardmäßig wird ein Makro in der aktuellen Arbeitsmappe gespeichert.
- In dem unteren Textfeld kann eine Beschreibung für das Makro eingegeben werden.



Aufzeichnung beenden

- Das Menüband Entwicklertools ist eingeblendet.
- Mit einem Mausklick auf die Aktion *Aufzeichnung beenden* wird die momentan laufende Aufzeichnung beendet.
- Hinweis: Alle Arbeitsschritte zwischen dem Beginn und dem Ende der Aufzeichnung sind in der Programmiersprache VBA gespeichert.

Visual Basic for Application (VBA) ...

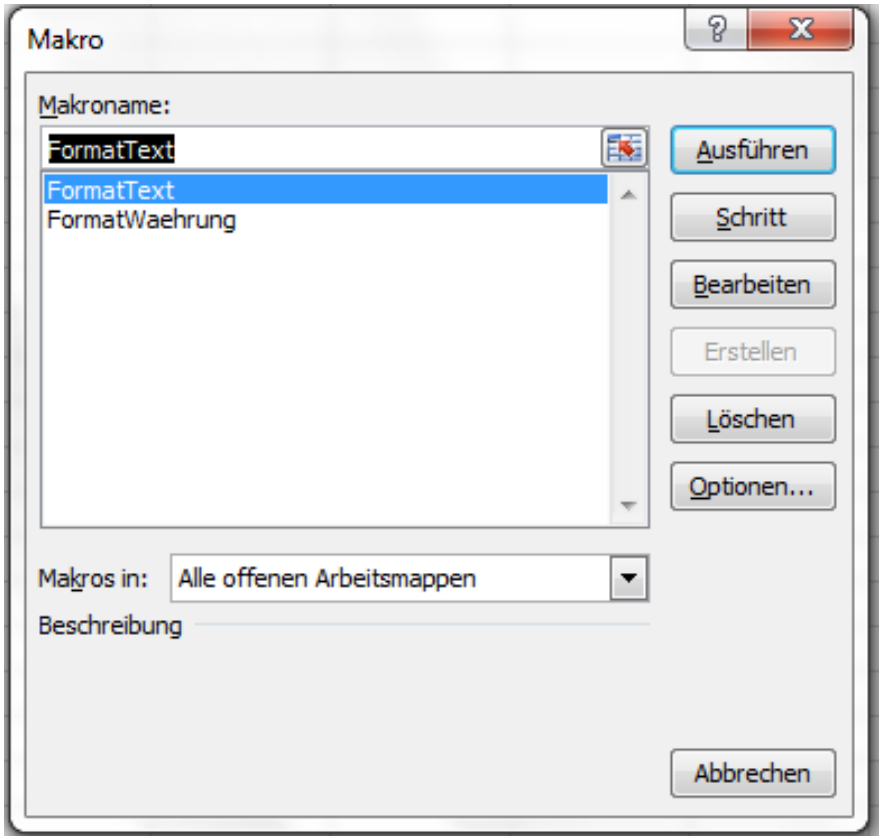
- automatisiert eine Folge von Arbeitsschritten.
- ist eine Programmiersprache, die in jeder Office-Anwendung eingebettet ist.
- erweitert die Funktionalität von Office-Anwendungen.
- passt eine Anwendung entsprechend der Wünsche des Benutzers an.

Start des Makros ...

- mit Hilfe des Symbols *Makros* im Bereich Code des Menübandes Entwicklertools.
- mit Hilfe von <CTRL> und der eingegebenen Taste.
- über ein Symbol in einem benutzerdefinierten Menüband.
- Nach dem Start werden die aufgezeichneten Arbeitsschritte abgearbeitet.
- Hinweis:
  - Durch das Makro wird die, bei der Aufzeichnung genutzte Datei immer wieder importiert.
  - Die Daten werden immer auf dem gleichen Tabellenblatt, beginnend in der gleichen Zelle abgelegt.

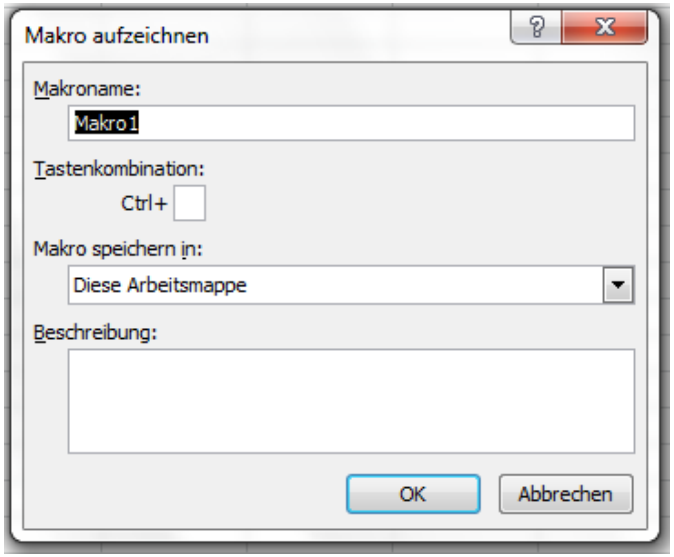
Entwicklertools - Makros

- Mit einem Mausklick wird ein Makro aus der Liste ausgewählt.
- Die Schaltfläche *Ausführen* startet das gewählte Makro.



… mit Hilfe einer Tastenkombination starten

- <CTRL>+<gewählte Taste> startet das Makro.



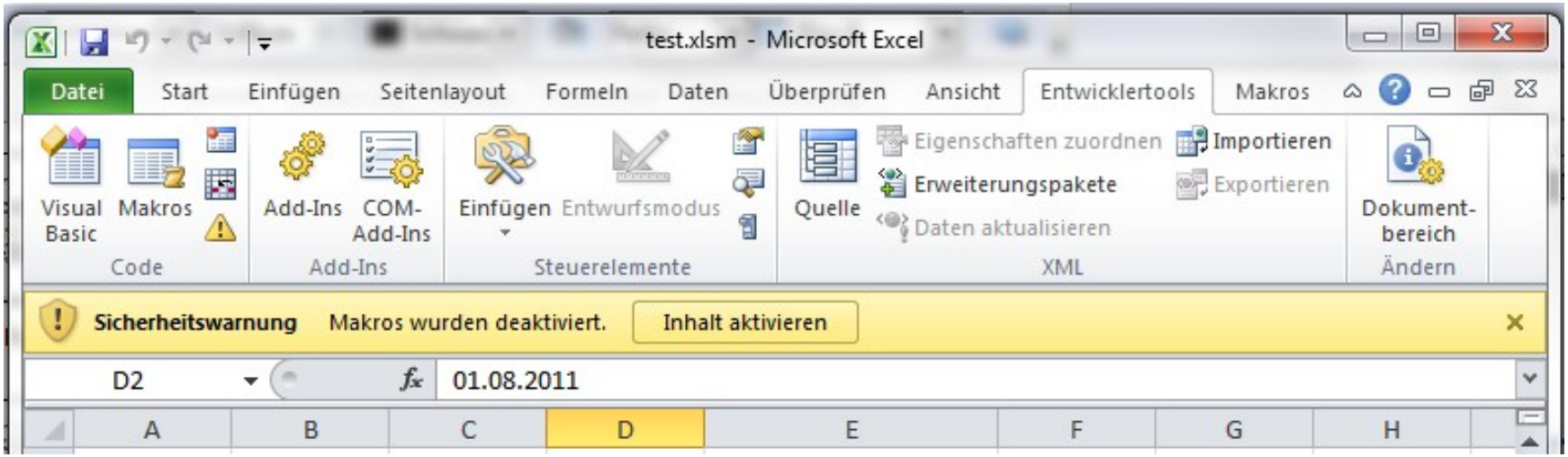
… über ein Menüband starten

- Voraussetzungen: Das Makro ist als Icon in einem Menüband sichtbar.
- Mit einem Klick auf das passende Icon wird das Makro gestartet.

Arbeitsmappen mit einem Makro speichern

- *Datei – Speichern unter*.
- Als Dateityp wird der Eintrag Excel-Arbeitsmappe mit Makros (*.xlsm) genutzt.
- Für die Speicherung wird ein aussagekräftiger Name eingegeben.
- Ein Ordner wird als Speicherplatz ausgewählt.

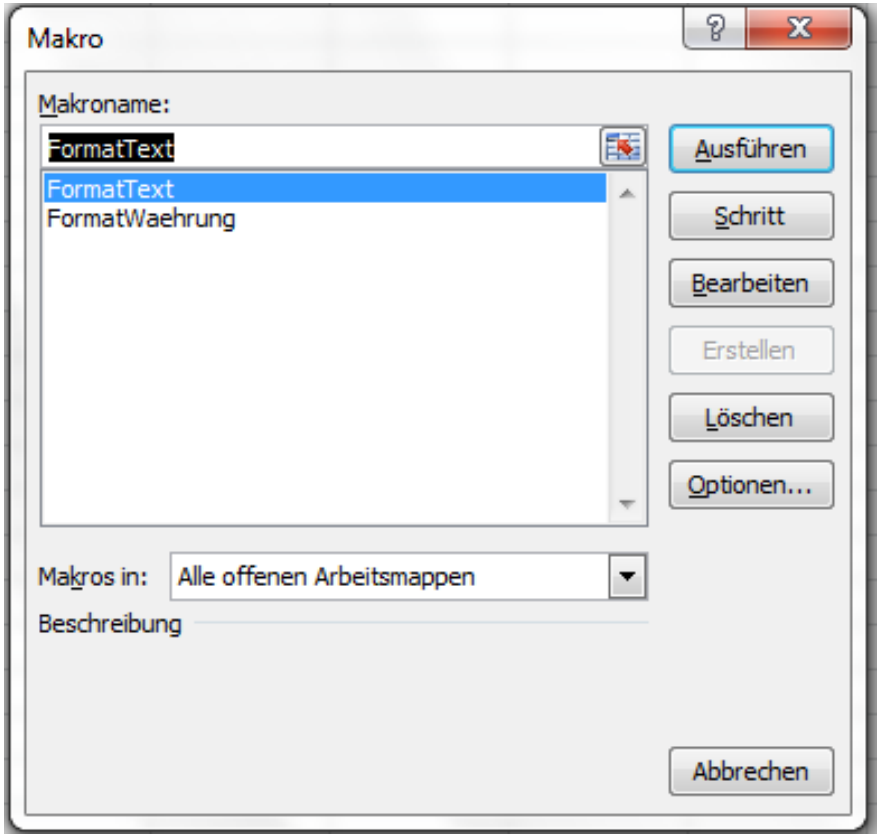
Arbeitsmappe mit einem Makro öffnen



- *Datei – Öffnen*.
- Beim erstmaligen Öffnen wird die Sicherheitswarnung angezeigt.
- Durch ein Klick auf die Schaltfläche *Inhalt aktivieren* werden die Makros aktiviert.

Makros bearbeiten

- Das Menüband Entwicklertools ist aktiv.
- In der Gruppe Code wird auf das Symbol *Makros* geklickt.
- Die Schaltfläche *Bearbeiten* öffnet den VBA-Editor. Alle Schritte des Makros werden in VBA-Code dargestellt.
- Die Schaltfläche *Optionen* öffnet ein Dialogfeld zur Eingabe einer Beschreibung und der Tastenkombination.



Aufgezeichnete Arbeitsschritte in VBA

```
Sub TextKonvertierung()
    With ActiveSheet.QueryTables.Add(Connection:= _
                "TEXT;C:\Users\Aue\Documents\meineDaten\Kursmaterial\
                ExcelAutomatisierung\makro\beispiele\importRTF.csv" _
                , Destination:=Range("$A$1"))
        .Name = "importRTF"
        .FieldNames = True
        .RowNumbers = False
        .FillAdjacentFormulas = False
        .PreserveFormatting = True
        .RefreshOnFileOpen = False
        .RefreshStyle = xlInsertDeleteCells
        .SavePassword = False
        .SaveData = True
        .AdjustColumnWidth = True
```

Aufgezeichnete Arbeitsschritte in VBA

```
        .RefreshPeriod = 0
        .TextFilePromptOnRefresh = False
        .TextFilePlatform = 1252
        .TextFileStartRow = 1
        .TextFileParseType = xlDelimited
        .TextFileTextQualifier = xlTextQualifierDoubleQuote
        .TextFileConsecutiveDelimiter = False
        .TextFileTabDelimiter = True
        .TextFileSemicolonDelimiter = True
        .TextFileCommaDelimiter = False
        .TextFileSpaceDelimiter = False
        .TextFileColumnDataTypes = Array(2, 1, 1, 1)
        .TextFileTrailingMinusNumbers = True
        .Refresh BackgroundQuery:=False
    End With
    ActiveWorkbook.Save
```

Objekte in Excel...

- sind zum Beispiel Tabellenblätter oder ein Zellbereich.
- haben Eigenschaften (Attribute). Attribute beschreiben das Aussehen und das Verhalten eines Objekts.
- haben Methoden. Methoden sind vordefinierte Funktionen, die die Eigenschaften beeinflussen.
- reagieren mit Hilfe von Ereignisprozeduren auf Benutzeraktionen. Zum Beispiel: Der Benutzer ändert den Inhalt einer Zelle. Es wird das Ereignis „Arbeitsmappe geändert“ ausgelöst. In der dazugehörigen Ereignisprozedur kann überprüft werden, ob die Änderung der Daten korrekt ist.

Objekt ActiveSheet ...

- ist eine Bezeichnung für das aktuell, aktive Arbeitsblatt in einer Excel-Anwendung.
- hat die Methode .Save, um Änderungen an dem Arbeitsblatt zu speichern.
- kann durch ThisWorkbook.Worksheets("[name]") ersetzt werden.

Objekt QueryTables

- ist ein Kind-Objekt von einem Arbeitsblatt. Kind-Objekte werden mit dem Elternobjekt durch ein Punkt verbunden.
- ist eine Auflistung von Abfragetabellen, die automatisch beim Import von Textdateien angelegt werden.
- verweist auf Daten in einem Arbeitsblatt, die aus einer externen Quelle bezogen werden.
- entspricht der Aktion *Verbindungen* auf dem Menüband Daten.

… hinzufügen

```
ActiveSheet.QueryTables.Add(Connection:= "TEXT;[dateipfad]",
                            Destination:=Range("$A$1")
```

- Die Methode .Add fügt der Auflistung eine neue Abfragetabelle hinzu.
- Die Methode benötigt folgende Eingabeparameter:
  - Eine Beschreibung der Verbindung (Connection). Welche Dateien (Text) werden von welchen Speicherort ([dateipfad]) importiert?
  - An welcher Position in welcher Arbeitsmappe werden die importierten Daten eingefügt (Destination)?

… löschen

```
ActiveSheet.QueryTables(1).Delete
```

- Die Methode .Delete löscht ein Element aus der Auflistung
- Mit Hilfe eines Index wird ein Element aus der Auflistung QueryTables eindeutig identifiziert. Dieses Element wird gelöscht. Als Index kann eine Ganzzahl oder der Name des Elements genutzt werden.

Einstellungen für ein Objekt zusammenfassen

```
With ActiveSheet.QueryTables.Add()


End With
```

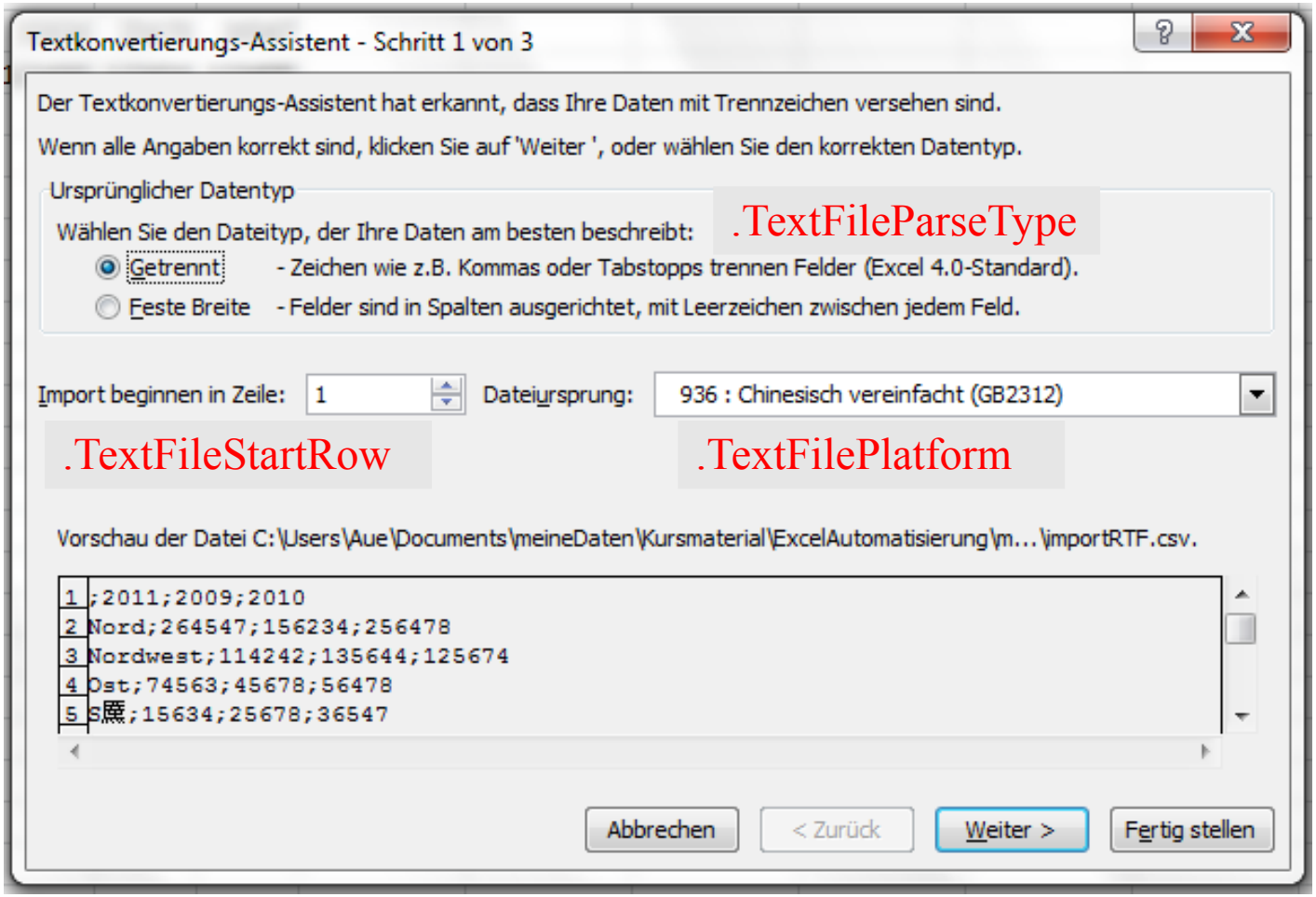
- Eine Zusammenfassung von Anweisungen für ein bestimmtes Objekt beginnt mit dem Schlüsselwort With.
- Dem Schlüsselwort folgt die Bezeichnung eines Objekts. In diesem Beispiel beziehen sich die nachfolgenden Anweisungen auf die neu erstellte Abfragetabelle.
- Die Zusammenfassung für das angegebene Objekt endet mit den Schlüsselwörtern End With.

Anweisungen in der With-Anweisung ...

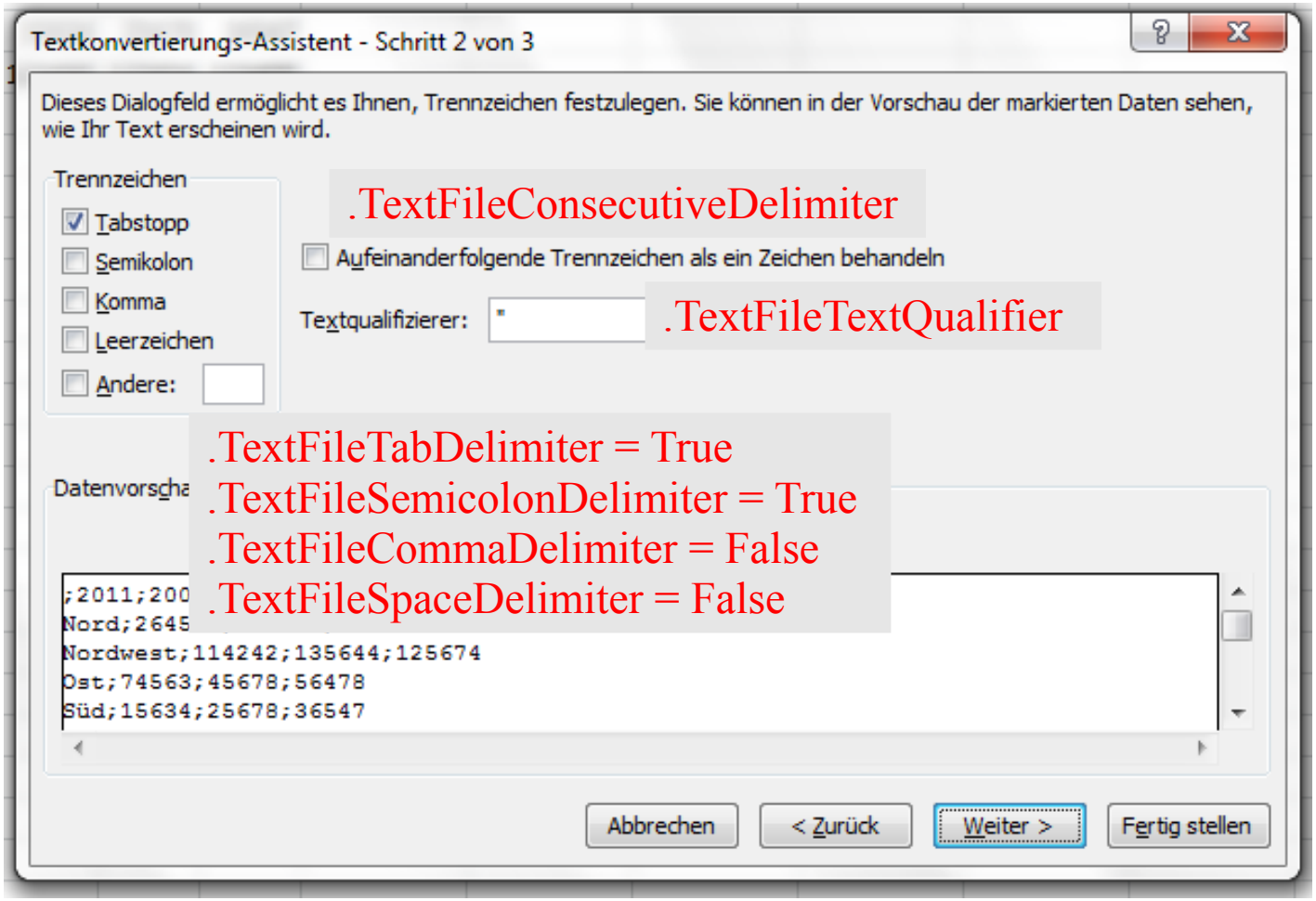
```
With ActiveSheet.QueryTables.Add()
    .Name = "importRTF_5"
    .FieldNames = True
End With
```

- die mit einem Punkt beginnen, beziehen sich auf das angegebene Objekt.
- Dem Punkt folgen
  - … Bezeichnungen von Attributen, die sich auf das angegebene Objekt beziehen.
  - Methoden, die Attribute des angegebenen Objekts verändern.

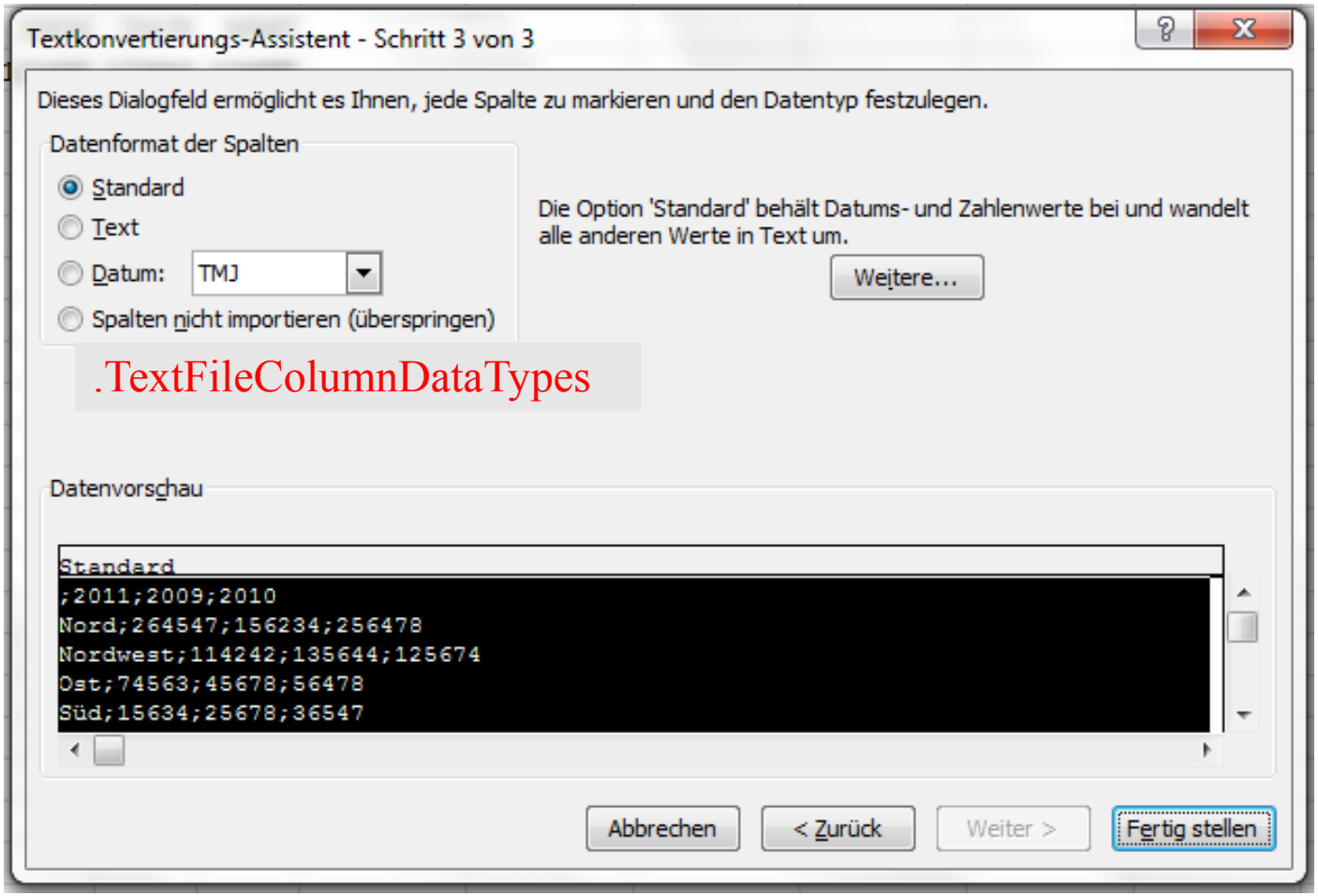
Textkonvertierungs-Assistent; Schritt 1



Textkonvertierungs-Assistent; Schritt 2



Textkonvertierungs-Assistent; Schritt 3



Daten einfügen

```
With ActiveSheet.QueryTables.Add()
    .Refresh BackgroundQuery:=False
End With
```

- Die Methode .Refresh aktualisiert den angegebenen Datenbereich.
- Falls diese Methode nicht vorhanden ist, werden die Daten aus der externen Quelle nicht im Arbeitsblatt angezeigt.
- In diesem Beispiel werden die externen Daten nicht im Hintergrund nicht automatisch aktualisiert.

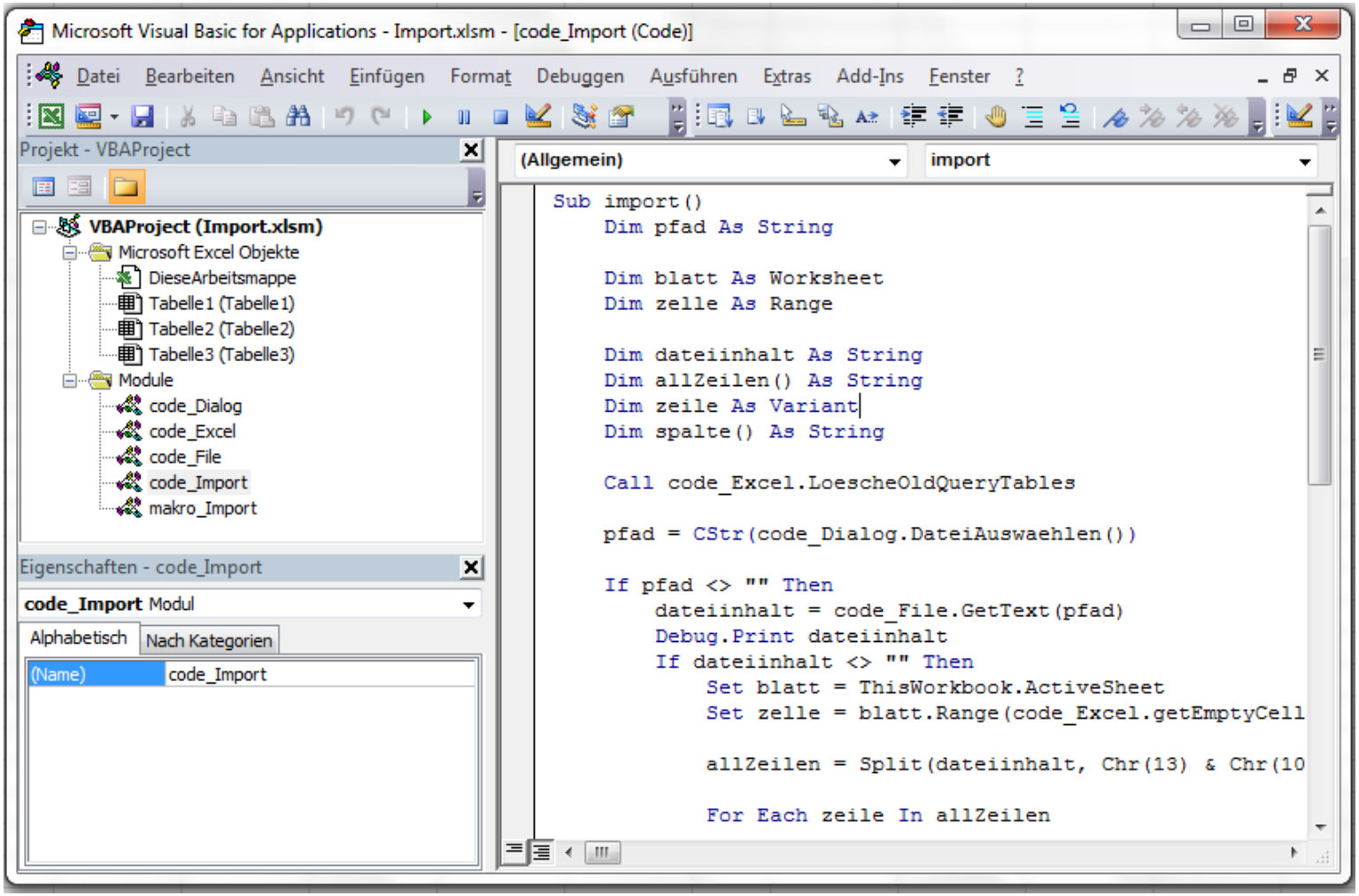
„Import“ selber schreiben

- Es werden alle vorhandenen Abfragetabellen gelöscht.
- Der Name der zu importierenden Datei wird ermittelt.
- Die gewählte Datei wird gelesen.
- Der Inhalt der Datei wird bearbeitet und ab der ersten leeren Zelle im aktiven Arbeitsblatt eingefügt.

VBA-Editor öffnen

- Das Menüband Entwicklertools ist eingeblendet.
- Mit einem Klick auf *Visual Basic* in der Gruppe Code wird der VBA-Editor geöffnet.

Beispiel


Microsoft Visual Basic for Applications - Import.xlsm - [code_Import (Code)]
Datei Bearbeiten Ansicht Einfügen Format Debuggen Ausführen Extras Add-Ins Fenster ?
Projekt - VBAProject
VBAProject (Import.xlsm)
Microsoft Excel Objekte
DieseArbeitsmappe
Tabelle1 (Tabelle1)
Tabelle2 (Tabelle2)
Tabelle3 (Tabelle3)
Module
code_Dialog
code_Excel
code_File
code_Import
makro_Import
Eigenschaften - code_Import
code_Import Modul
Alphabetisch
Nach Kategorien
(Name)
code_Import
(Allgemein)
import
Sub import()
Dim pfad As String
Dim blatt As Worksheet
Dim zelle As Range
Dim dateiinhalt As String
Dim allZeilen() As String
Dim zeile As Variant
Dim spalte() As String
Call code_Excel.LoescheOldQueryTables
pfad = CStr(code_Dialog.DateiAuswaehlen())
If pfad <> "" Then
dateiinhalt = code_File.GetText(pfad)
Debug.Print dateiinhalt
If dateiinhalt <> "" Then
Set blatt = ThisWorkbook.ActiveSheet
Set zelle = blatt.Range(code_Excel.getEmptyCell
allZeilen = Split(dateiinhalt, Chr(13) & Chr(10
For Each zeile In allZeilen

VBA-Editor ...

- ist eine integrierte Entwicklungsumgebung (IDE) für die Programmiersprache V(isual)B(asic for)A(pplication).
- ist in jeder Office-Anwendung vorhanden.
- ist eine eigenständige Anwendung, die in der Taskleiste als Symbol eingeblendet wird.
- bietet die Möglichkeit VBA-Code zu lesen und zu bearbeiten.

Aufbau

- Am oberen Rand befindet sich die Titelleiste, die Informationen zu dem Projekt anzeigt.
- Darunter wird die Menüleiste und die Symbolleiste angezeigt.
- Am linken Rand wird standardmäßig der Projekt-Explorer sowie das Eigenschaftenfenster eingeblendet.
- Am rechten Rand ist das Codefenster angedockt.

Titelleiste ...

- befindet sich am oberen Rand der Anwendung.
- hat in der linken Ecke ein Icon, welches die Anwendung symbolisiert. Mit einem Klick auf das Icon wird das dazugehörige Systemmenü geöffnet.
- zeigt den Namen der Excel-Datei sowie des aktiven Moduls an.
- hat am rechten Rand Schaltflächen zum Minimieren, Verkleinern / Maximieren oder Schließen der Anwendung.

Menüleiste ...

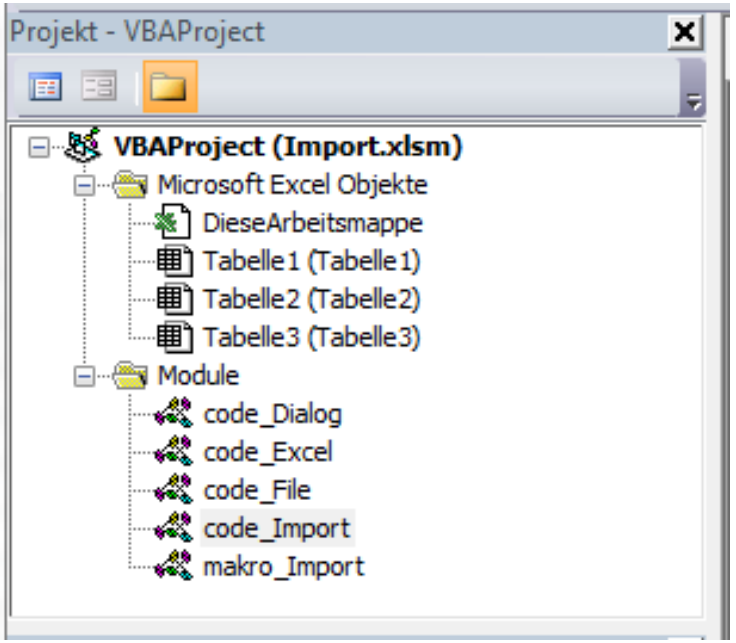
- befindet sich unterhalb der Titelleiste.
- sammelt alle Befehle der Anwendung.
- bietet aufklappbare Menüs zu verschiedenen Themen an. In diesen Menüs werden Befehle und Funktionen der Anwendung gesammelt.
- zeigt auf der obersten Ebene Kategorien an. In Abhängigkeit dieser Kategorien werden die Befehle zusammengefasst. Der Name der Kategorie gibt einen ersten Hinweis auf die Benutzung der darin enthaltenen Befehle.

Symbolleisten ...

- sammeln häufig genutzte Aktionen zu einem Thema. Die Aktionen werden durch Icons dargestellt.
- beginnen mit der senkrechten gestrichelten Linie am linken Rand. Sobald die Maustaste über diese Linie liegt, kann die Symbolleiste mit Hilfe von Drag (Maustaste gedrückt) & Drop (Maustaste loslassen) verschoben werden.
- werden über das Menü *Ansicht* – *Symbolleiste* ein- oder ausgeblendet.
- haben einen Pfeil nach unten am rechten Rand. Mit einem Klick auf den Pfeil wird ein Menü zum Ein- und Ausblenden von Icons in der Symbolleiste angezeigt.

Projekt-Explorer ...

- ist die Schaltzentrale für die Programmierung einer Excel-Anwendung.
- verwaltet die, zu dem Projekt gehörende Arbeitsmappe sowie die darin enthaltenen Arbeitsblätter.
- zeigt aufgezeichnete Makros in Modulen an.
- ist frei platzierbar.
- kann über das Menü *Ansicht* ein- oder ausgeblendet werden.



Aufbau des Projekt-Explorers

- In der Titelleiste wird die Schließen-Schaltfläche angezeigt.
- Darunter befindet sich die Symbolleiste mit den Icons
  - „Zeige zum gewählten Element den Code an“.
  - „Zeige das passende Objekt zum Code an“.
  - „Sortierung mit Hilfe von Ordnern“.
- Unterhalb der Symbolleiste werden die Module alphabetisch oder in Abhängigkeit ihres Typs sortiert angezeigt.

Module ...

- kapseln Code zu einem Thema.
- sind Container für Code. Ein Container ist ein Sammelbecken für Code zu einem bestimmten Thema.
- fassen Programmiercode und Deklarationen zu einem Thema zusammen.
- werden automatisch durch die Aufzeichnung eines Makros angelegt.
- können vom Entwickler oder der Anwendung angelegt werden.

Modul-Typen ...

- werden mit Hilfe von Ordnern im Projekt-Explorer dargestellt.
- (Microsoft Excel Objekte) enthält Module, die Code für die Arbeitsmappe oder die darin enthaltenen Arbeitsblätter sammeln.
- (Module) ist ein Container für aufgezeichnete Makros oder vom Entwickler selbst geschriebene Prozeduren.

Neues Modul einfügen

- *Einfügen – Modul*.
- Im Eigenschaftenfenster wird der Name des Moduls eingegeben.
- *Datei - … speichern* speichert die Excel-Anwendung und die darin enthaltenen Module.

Start-Prozedur in das neue Modul einfügen

```
Sub startImport()
        ...

End Sub
```

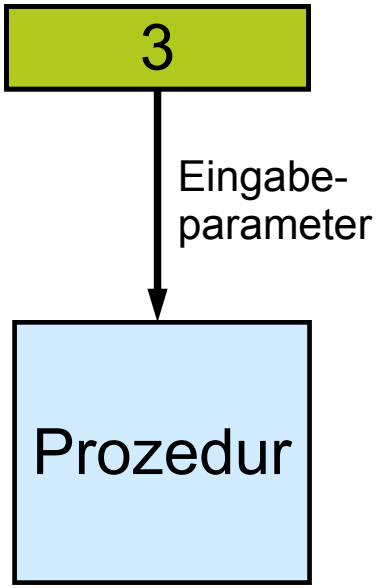
- Mit Hilfe der Tastatur wird der Code in das neue Modul eingegeben.
- In der Start-Prozedur werden die einzelnen Arbeitsschritte zusammengefasst.
- Arbeitsschritte, die in späteren Programmen benötigt werden, werden in andere Prozeduren ausgelagert.

Prozeduren ...

- beginnen in VBA mit Sub und enden mit End Sub. Das Ende der Prozedur wird häufig automatisch durch den VBA-Editor eingefügt.
- können mit Hilfe Ihres Namens aufgerufen werden.
- geben keinen Wert an den Aufrufer zurück.
- können mit Hilfe von Exit Sub vorzeitig beendet werden.

Arbeitsweise

- Der Prozedur können Eingabeparameter (Argumente) übergeben werden.
- Diese Eingabeparameter werden in der Prozedur verarbeitet.
- Die Verarbeitung selber ist dem Aufrufer aber nicht bekannt. Der Nutzer kennt nur die Schnittstelle (den Prozedurkopf) nach außen.



Prozedurkopf

```
Sub startImport()
Sub procName(arg01 As Typ, arg02 As Typ, ...)
```

- Der Prozedurkopf beginnt mit dem Schlüsselwort `Sub`.
- Dem Schlüsselwort folgt der Name der Prozedur. Der Name ist frei wählbar.
- In runden Klammern steht die Argumentliste. Die Argumentliste beschreibt die Anzahl und Art der Eingabeparameter. Die Argumente werden in der Liste durch Kommata getrennt.

Argumentliste ...

Sub startImport**()**

Sub procName**(arg01 As Typ, arg02 As Typ, ...)**

- beginnt und endet mit den runden Klammern.
- steht direkt hinter dem Prozedur-Namen.
- kann aus beliebig vielen Argumenten bestehen. Die Argumente werden durch Kommata getrennt.
- kann leer sein. Der Prozedur werden keine Argumente übergeben.

Argumente ...

Sub procName(**arg01 As Typ, arg02 As Typ, ...**)

- symbolisieren Platzhalter für einen Wert, die der Funktion übergeben werden..
- haben einen eindeutigen Namen.
- sind in der Prozedur gekapselt.
- bekommen mit Hilfe von As einen Datentyp zugewiesen. Es kann jeder beliebiger Datentyp genutzt werden. Unter dem Menüpunkt *Visual Basic-Sprachverzeichnis – Datentypen* in der Hilfe im VBA-Editor werden alle vorhandenen Standardtypen aufgelistet.

Aufruf von Prozeduren

```
Sub startImport()
        Call code_Excel.LoescheOldQueryTables

End Sub
```

```
Sub LoescheOldQueryTables()
  Dim count As Integer

  For count = ActiveSheet.QueryTables.count To 1 Step -1
    ActiveSheet.QueryTables(count).Delete
  Next
End Sub
```

Prozedur aufrufen

Call code_Excel.LoescheOldQueryTables

- Mit Hilfe des Schlüsselwortes Call wird ein Prozedur-Aufruf gestartet.
- Die Prozedur wird mit ihren Namen aufgerufen (LoescheOldQueryTables).
- Der Speicherort der Prozedur muss angegeben werden (code_Excel). Das Modul, in dem die Prozedur definiert ist und die aufzurufende Prozedur werden mit einem Punkt verbunden.
- Falls der Prozedur keine Parameter übergeben werden, sind die runden Klammern leer.

Abfrage-Tabellen löschen

```
Sub LoescheOldQueryTables()
  Dim count As Integer

  For count = ActiveSheet.QueryTables.count To 1 Step -1
    ActiveSheet.QueryTables(count).Delete
  Next

End Sub
```

Variablen ...

- sind Platzhalter für Zahlen und Zeichen im Speicher.
- können einen beliebigen Standard-Datentyp annehmen.
- haben einen eindeutigen Namen. Der Name sollte selbsterklärend sein.
- bekommen mit Hilfe des Gleichheitszeichen einen Wert zugewiesen.

... definieren

Dim count As Integer

Dim variablenname As Datentyp

- Jede Variable hat einen eindeutigen Namen (count), der mit einem Buchstaben beginnt. Die Bezeichnung count sollte genauso wie name nicht für Variablennamen genutzt. Beide Bezeichnung beschreiben Attribute / Methoden von Objekten.
- Jede Variable ist von einem bestimmten Typ. Der Typ wird mit Hilfe des Schlüsselwortes As zugewiesen. In diesem Beispiel kann die Variable Ganzzahlen (Integer) speichern.
- Mit Hilfe des Schlüsselwortes Dim wird der Nutzer der Variablen festgelegt. In diesem Beispiel kann die Variable nur in der Prozedur lokal genutzt werden.

Datentypen ...

- legen das Format des zu speichernden Wertes fest.
- legen Regeln für die Verwendung der Variablen fest.
- beschreiben die maximale Größe des Platzhalters.
- legen den Speicherbedarf fest.
- werden in der Hilfe unter *Visual Basic-Sprachverzeichnis - Datentypen* aufgelistet.

Datentypen für Ganzzahlen

	Speicherbedarf in Bytes	Datenbereich
As Byte	1	0 - 255
As Integer	2	-32.768 - +32.767
As Long	4	-2.147.483.648 - +2.147.483.647
As Boolean	2	0 (falsch, false) <> 0 (wahr, true)

Zählschleifen ...

```
Dim count As Integer

For count = ActiveSheet.QueryTables.count To 1 Step -1

Next
```

- durchlaufen ein Intervall. Die Größe des Intervalls wird durch ein Startwert und ein Endwert definiert..
- haben eine definierte Anzahl von Durchläufen.
- Das Schlüsselwert Next berechnet den Zähler neu und startet einen neuen Schleifendurchlauf.

Start- und Endwert für die Schleife

```
For count = ActiveSheet.QueryTables.Count To 1
For Startwert To Endwert
```

- Der Kopf der Schleife beginnt mit dem Schlüsselwort For.
- Dem Schlüsselwort folgt eine Anweisung, die einem Zähler einen Anfangswert zuweist. In diesem Beispiel wird als Startwert die Anzahl der vorhandenen Abfragetabellen genutzt.
- Der Start sowie das Ende des Intervalls werden im Kopf definiert. In diesem Beispiel läuft die Schleife rückwärts von der Anzahl der vorhandenen Abfragetabellen bis zu (To) eins.

Schrittweite festlegen

For Startwert To Endwert Step -1

For Startwert To Endwert Step Schrittweite

- Dem Schlüsselwort Step folgt ein Wert, der die Schrittweite angibt.
- Falls keine Schrittweite angegeben ist, wird der Zähler bei jeden Durchlauf um eins hochgezählt.
- Intervall vorwärts durchlaufen: Positive Schrittweite.
- Inter rückwärts durchlaufen: Negative Schrittweite.
- Die Schrittweite und der Zähler sind vom gleichen Typ.

Beispiele

- `For count = 1 To 10 Step 2` durchläuft die Zahlen 1 bis 10 von von der kleinsten Zahl zur größten Zahl. Bei jedem Schleifendurchlauf wird zu dem Zähler 2 addiert.
- `For count = 10 To 1 Step -2` durchläuft die Zahlen 10 bis 1 von von der größten Zahl zur kleinsten Zahl. Bei jedem Schleifendurchlauf wird der Zähler um 2 subtrahiert.
- `For count = 1 To 10 Step 0.5` erzeugt eine Endlosschleife. Die Schrittweite entspricht nicht dem Datentyp des Zählers. Zu einer Ganzzahl kann keine Dezimalzahl hinzuaddiert werden.

Datei für den Import auswählen und zurückgeben

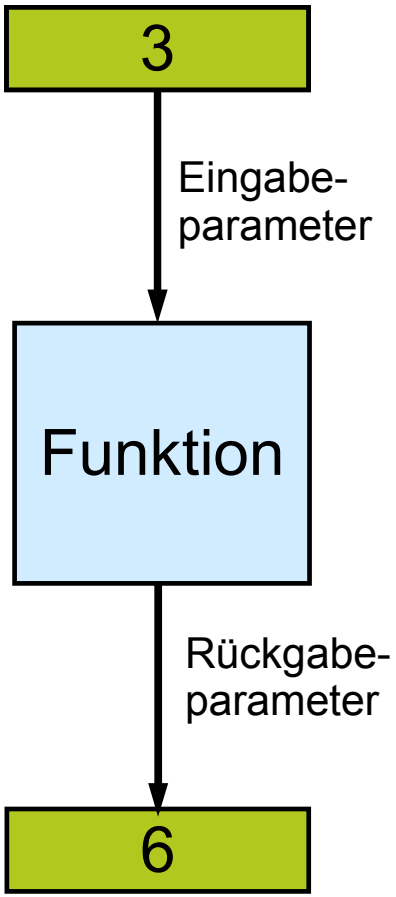
```
Function DateiAuswaehlen() As String
    Dim dateiname As Variant
    Dim filter As String

    filter = "CSV-Dateien (*.csv),*.csv, "
    filter = filter & "Micrsoft Excel-Dateien(*.xls), *.xlsx,"
    dateiname = Application.GetOpenFilename(FileFilter:=filter, FilterIndex:=1, _
                                Title:="...", ButtonText:="Auswählen", MultiSelect:=False)

   If dateiname <> False Then
       DateiAuswaehlen = dateiname
   Else
       DateiAuswaehlen = ""
   End If
End Function
```

Funktionen ...

- beginnen in VBA mit Function und enden mit End Function. Das Schlüsselwort End wird häufig durch den VBA-Editor automatisch eingefügt.
- können mit Hilfe Ihres Namens aufgerufen werden.
- geben exakt einen Wert an den Aufrufer zurück.
- können mit Hilfe von Exit Function vorzeitig beendet werden.

Arbeitsweise

- Der Funktion können Eingabeparameter (Argumente) übergeben werden.
- Diese Eingabeparameter werden in der Funktion verarbeitet. Die Verarbeitung selber ist dem Aufrufer aber nicht bekannt. Der Nutzer kennt nur die Schnittstelle (den Funktionskopf) nach außen.
- Die Funktion gibt einen Wert zurück. Wie dieser Wert berechnet wurde, weiß der Nutzer nicht. Der Nutzer kennt nur die Art des Rückgabewertes.



Funktionskopf

```
Function DateiAuswaehlen() As String
Function funcName(arg01 As Typ, arg02 As Typ, ...)
```

- Der Funktionskopf beginnt mit dem Schlüsselwort `Function`.
- Dem Schlüsselwort folgt der Name der Funktion. Der Name ist frei wählbar.
- In runden Klammern steht die Argumentliste. Die Argumentliste beschreibt die Anzahl und Art der Eingabeparameter. Die Argumente werden in der Liste durch Kommata getrennt.
- Der Argumentliste folgt der Datentyp der Funktion.

Argumentliste ...

Function DateiAuswaehlen**()** As String

Function funcName**(arg01 As Typ, arg02 As Typ, ...)**

- beginnt und endet mit den runden Klammern.
- steht direkt hinter dem Funktionsnamen.
- kann aus beliebig vielen Argumenten bestehen. Die Argumente werden durch Kommata getrennt.
- kann leer sein. Der Funktion werden keine Argumente übergeben.

Argumente ...

Function funcName(**arg01 As Typ, arg02 As Typ**, ...)

- symbolisieren Platzhalter für Werte, die der Funktion übergeben werden.
- haben einen eindeutigen Namen.
- sind in der Funktion gekapselt.
- bekommen mit Hilfe von As einen Datentyp zugewiesen. Es kann jeder beliebiger Datentyp genutzt werden. Unter dem Menüpunkt *Visual Basic-Sprachverzeichnis – Datentypen* in der Hilfe im VBA-Editor werden alle vorhandenen Standardtypen aufgelistet.

Datentyp der Funktion...

Function DateiAuswaehlen() *As String*

Function funcName(arg01 As Typ, arg02 As Typ, ...)

- wird am Ende des Funktionskopfes angegeben. Der Argumentliste folgt das Schlüsselwort As plus einer Typ-Angabe.
- kann von jeden beliebigen Datentyp sein. Unter dem Menüpunkt *Visual Basic-Sprachverzeichnis – Datentypen* in der Hilfe im VBA-Editor werden alle vorhandenen Standardtypen aufgelistet.
- ist optional. Wenn keine Angabe zum Datentyp vorhanden ist, wird der Datentyp anhand des Rückgabewertes ermittelt.

Rückgabewert an die Funktion übergeben

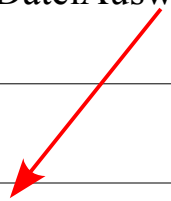
DateiAuswaehlen = dateiname

FuncName = Ausdruck

- Dem Funktionsnamen kann an einer beliebigen Stelle im Code mit Hilfe des Gleichheitszeichen ein Wert zugewiesen werden.
- Diesen Wert kann der Aufrufer weiterverarbeiten.
- Die Zuweisung eines Rückgabewertes an eine Funktion beendet nicht die Funktion. Die Funktion wird durch das Schlüsselwort End beendet.
- Der Rückgabewert und der Typ der Funktion sollten gleich sein.

Aufruf der Funktion aus der Start-Prozedur

```
Sub startImport()
    Dim pfad As String

   Call code_Excel.LoescheOldQueryTables

    pfad = CStr(code_Dialog.DateiAuswaehlen())
End Sub
```



```
Function DateiAuswaehlen() As String
   Dim dateiname As Variant
   Dim filter As String

End Function
```

Funktion aufrufen

```
pfad = code_Dialog.DateiAuswaehlen()
dateiinhalt = code_File.GetText(pfad)
```

- Mit Hilfe des Namens (DateiAuswaehlen, GetText) wird die Funktion aufgerufen.
- Dem Funktionsnamen folgen die runden Klammern für die Argumentliste. Falls keine Parameter übergeben werden, sind die runden Klammern leer. Andernfalls werden die Parameter wie im Kopf der Funktion aufgeführt, angegeben.
- Die Funktionen können in jeder beliebigen Funktion oder Prozedur aufgerufen werden.

Speicherort einer Funktion angeben

```
pfad = code_Dialog.DateiAuswaehlen()
dateiinhalt = code_File.GetText(pfad)
```

- Eine Funktion oder Prozedur befindet sich in einem Modul.
- Der Modulname wird mit dem Funktionsnamen durch ein Punkt verbunden.

Zuweisung des Rückgabewertes

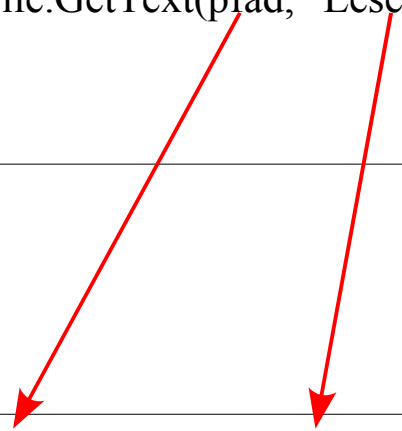
```
dateiinhalt = code_File.GetText(pfad)
pfad = CStr(code_Dialog.DateiAuswaehlen())
```

- Der Rückgabewert der Funktion wird einer Variablen mit Hilfe eines Gleichheitszeichen zugewiesen. Der Rückgabewert und die Variable, in der dieser gespeichert wird, haben den gleichen Datentyp. Andernfalls muss der Wert mit Hilfe von Konvertierungsfunktionen umgewandelt werden. Die Funktion CStr() wandelt einen beliebigen Datentypen in einen String um. Eine Liste der Funktionen finden Sie unter http://www.techonthenet.com/excel/formulas/
- Der Rückgabewert kann aber auch direkt in einer Berechnung oder Bedingung weiterverarbeitet werden.

Zuordnung der Argumente beim Aufruf

Start ...

```
Private Sub import()
    dateiinhalt = code_File.GetText(pfad, "Lesen")

End Sub
```



Modul code_File

```
Function GetText(filename As String, modus As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

End Function
```

Erläuterung

- Die Anzahl der Parameter im Aufruf entspricht der Anzahl der Elemente in der Argumentliste.
- Die Parameter werden den Argumenten von links nach rechts zugeordnet.
- Die Parameter werden den Argumenten in Abhängigkeit der Position zugeordnet. D. h. der erste Parameter wird dem ersten Argument zugeordnet und so weiter.
- Um Fehler zu vermeiden sollte der Parameter und das Argument den gleichen Datentyp besitzen.

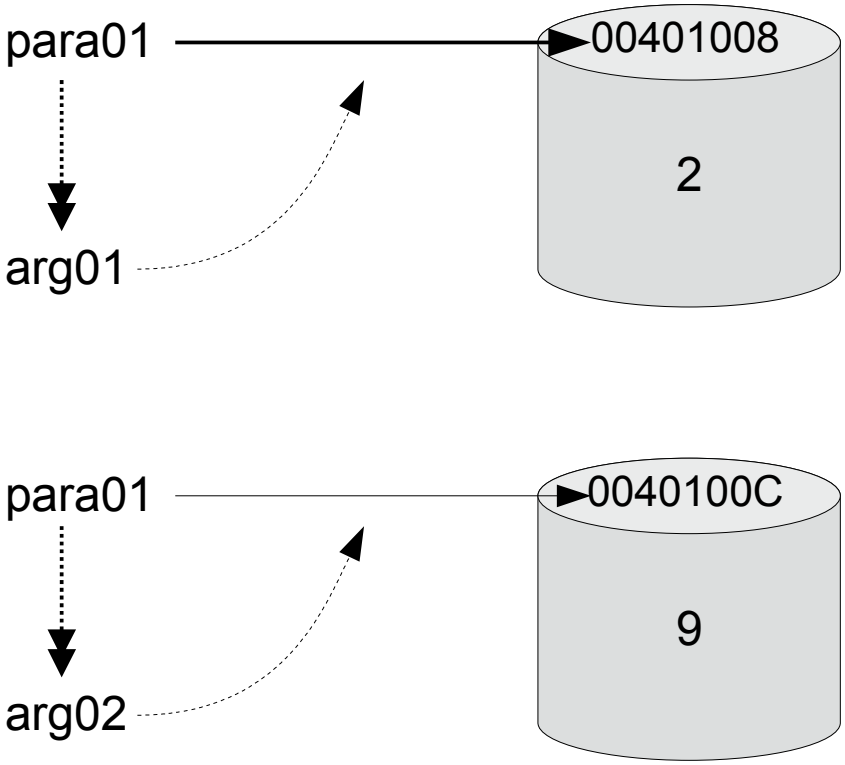
Gemeinsame Nutzung von Werten / Variablen

Function GetText(**ByRef** filename As String, **ByRef** modus As String)

Function GetText(filename As String, modus As String)

- ist die Standardübergabe von Argumenten.
- Der zu übergebene Parameter verweist auf einen Speicherplatz, an dem ein bestimmter Wert abgelegt ist.
- Das Argument bekommt nicht den Wert übergeben, sondern die Position des Speicherplatzes. Das Argument verweist auf den gleichen Speicherplatz und damit Wert wie der Parameter.

Grafische Darstellung

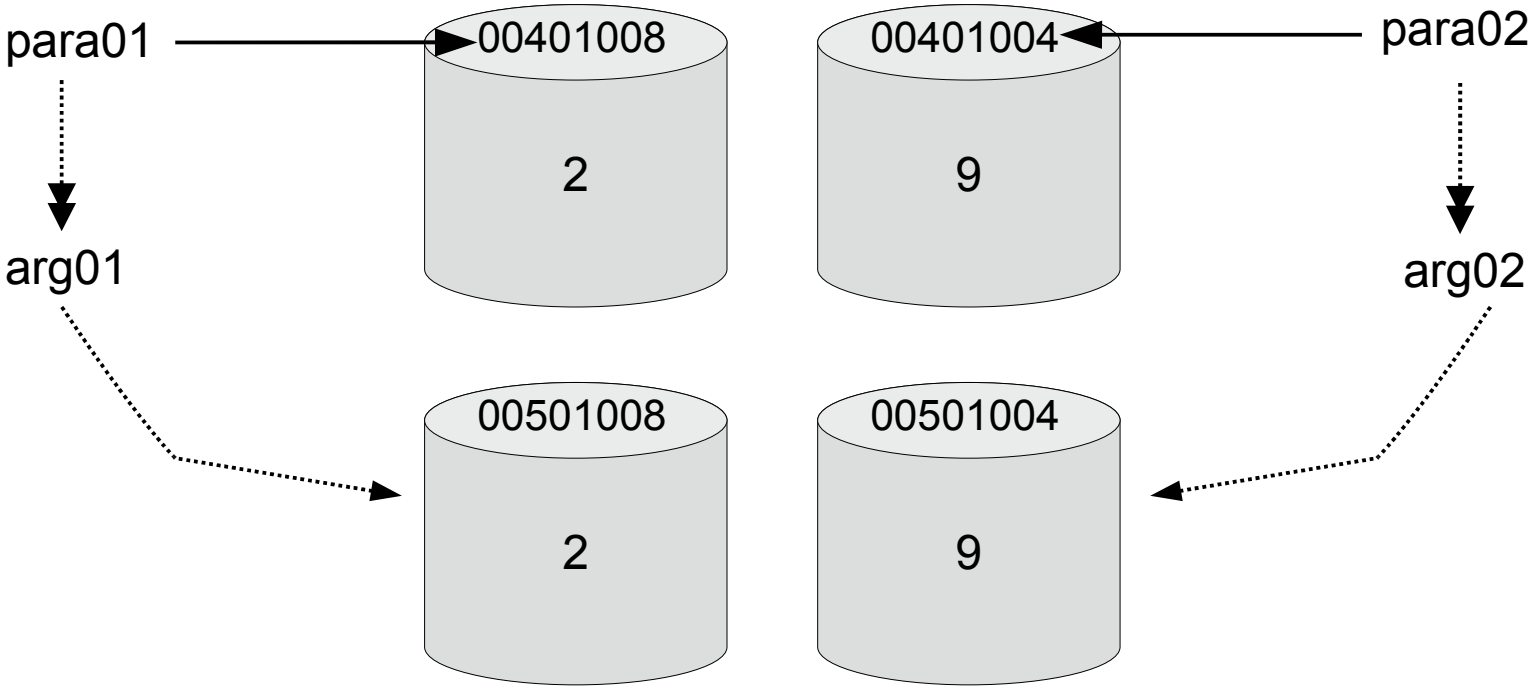


Schreibschutz für die Parameter

Function GetText(**ByVal** filename As String, **ByVal** modus As String)

- Der zu übergebene Parameter verweist auf einen Speicherplatz, an dem ein bestimmter Wert abgelegt ist.
- Das Argument bekommt eine Kopie des Wertes übergeben. Der Platzhalter „Parameter“ wird kopiert.
- Vorteil: Die Parameter können nicht durch die aufgerufene Prozedur oder Funktion verändert werden.

Grafische Darstellung



Dialog „Öffnen“ nutzen

```
Function DateiAuswaehlen() As String
    Dim dateiname As Variant
    Dim filter As String

    filter = "CSV-Dateien (*.csv),*.csv, "
    filter = filter & "Micrsoft Excel-Dateien(*.xls), *.xlsx,"

    dateiname = Application.GetOpenFilename(FileFilter:=filter, _
                                            FilterIndex:=1, _
                                           Title:="...", _
                                            ButtonText:="Auswählen", _
                                             MultiSelect:=False)

End Function
```

Erläuterung

- Das Objekt Application ist ein Platzhalter für die Excel-Anwendung.
- Dieses Objekt hat mehrere Methoden zum Erstellen von Dialogen wie „Speichern“ oder „Öffnen“. In diesem Beispiel wird der Dialog „Öffnen“ (GetOpenFilename) genutzt.
- Der Methode GetOpenFilename …
  - werden verschiedene Argumente übergeben.
  - gibt einen Wert an den Aufrufer zurück. Der Rückgabewert spiegelt die gedrückte Schaltfläche in dem Dialog wieder.

Argumente

- FileFilter legt die Auswahl der Dateitypen fest.
- FilterIndex legt den favorisierten Dateityp fest. Welcher Dateityp wird nach dem Öffnen angezeigt?
- Title beschreibt die Aufgabe des Dialogs. Der Text wird in der Titelleiste angezeigt.
- MultiSelect erlaubt eine Mehrfachauswahl von Dateien.

Rückgabewert

- Wenn der Benutzer die OK-Schaltlfäche gedrückt hat, wird der gewählte Dateiname zurückgegeben. Der Name kann in einer Variablen vom Datentyp String (Text) gespeichert werden.
- Wenn der Benutzer die Abbrechen-Schaltfläche drückt, wird der Wert False zurückgeliefert. Dieser Wert kann in einer Variablen vom Datentyp Boolean (Wahr; Falsch) gespeichert werden.
- Durch die unterschiedlichen Typen wird der Rückgabewert in einer Variablen vom Datentyp Variant gespeichert. Dieser Typ passt sich dem Datentyp des Rückgabewertes an.

… abfragen

```
Function DateiAuswaehlen() As String
    Dim dateiname As Variant
    Dim filter As String



   If dateiname <> False Then
        DateiAuswaehlen = dateiname
   Else
        DateiAuswaehlen = ""
   End If

End Function
```

… durch eine bedingte Anweisung

```
Wenn Bedingung = Wahr Dann

  Wenn Bedingung = Wahr Dann
    ' Anweisungen
  Andernfalls Wenn Bedingung = Wahr Dann
    ' Anweisungen
  Andernfalls
    ' Anweisungen
  Ende

Ende
```

Bedingungen ...

- sind Ausdrücke, die wahr oder falsch sind.
- vergleichen Werte.
- nutzen Vergleichsoperatoren.
- überprüfen das boolsche Ergebnis von Funktionen, die mit „Is“ beginnen.
- können verknüpft werden.
- werden häufig in runde Klammern gesetzt. Die runden Klammern dienen der besseren Lesbarkeit.

Vergleichsoperatoren

Operator	Erläuterung
=	ist gleich
<>	ungleich
<	kleiner als
<=	kleiner gleich
>	größer
>=	größer gleich

… für Zahlen nutzen

Operator	Erläuterung	Beispiel
=	ist gleich	3 = 4 ≈ *Falsch*
<>	ungleich	3 <> 4 ≈ *Wahr*
<	kleiner als	3 < 4 ≈ *Wahr*
<=	kleiner gleich	3 <= 4 ≈ *Wahr*
>	größer	3 > 4 ≈ *Falsch*
>=	größer gleich	3 >= 4 ≈ *Falsch*

- Dezimalzahlen vom Typ Double oder Single sollten nicht auf Gleichheit geprüft werden. Näherungswerte müssen nicht exakt den angegebenen Wert entsprechen.
- Ganzzahlen werden als Byte, Integer oder Long definiert.

… für Datumswerte nutzen

Operator	Erläuterung	Beispiel
=	ist gleich	#2/12/2011# = #6/11/2011# ≈ *Falsch*
<>	ungleich	#2/12/2011# <> #6/11/2011# ≈ *Wahr*
<	kleiner als	#2/12/2011# < #6/11/2011# ≈ *Wahr*
<=	kleiner gleich	#2/12/2011# <= #6/11/2011# ≈ *Wahr*
>	größer	#2/12/2011# > #6/11/2011# ≈ *Falsch*
>=	größer gleich	#2/12/2011# >= #6/11/2011# ≈ *Falsch*

- Datumswerte beginnen und enden in VBA mit einem Hash-Zeichen.
- Datumswerte werden als konstanter Wert in der Form #monat/tag/jahr# angegeben.
- Variablen für Datums- und Zeitwerten sind vom Datentyp Date.

… für Texte nutzen

Operator	Erläuterung	Beispiel
=	ist gleich	*"abc" = "ABC" ≈ Falsch*
<>	ungleich	*"abc" <> "ABC" ≈ Wahr*
<	kleiner als	*"abc" < "ABC" ≈ Falsch*
<=	kleiner gleich	*"abc" <= "ABC" ≈ Falsch*
>	größer	*"abc" > "ABC" ≈ Wahr*
>=	größer gleich	*"abc" >= "ABC" ≈ Wahr*

- Texte beginnen und enden immer mit den Anführungszeichen.
- Variablen für Texte sind vom Datentyp String.
- Es werden die ASCII-Codierungen der Buchstaben verglichen. (A = 65; a = 97).

Inhalt der Datei lesen

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

   filenumber = FreeFile

   Open filename For Input As #filenumber
   dateiinhalt = Input$(LOF(filenumber), #filenumber)
   Close

   GetText = dateiinhalt
End Function
```

Dateinummer festlegen

```
Function GetText(filename As String) As String
  Dim filenumber As Integer

  filenumber = FreeFile
End Function
```

- Die vordefinierte Funktion FreeFile gibt eine freie Dateinummer zurück.
- Die freie Dateinummer wird in einer Variablen vom Typ Ganzzahl gespeichert.

Textdatei öffnen

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

   filenumber = FreeFile
   Open filename … As #filenumber
End Function
```

- Die vordefinierte Funktion Open öffnet eine Datei. In diesem Beispiel wird der Dateiname als Argument der Funktion übergeben.
- Die Datei wird in einem Puffer zwischengespeichert.
- Mit Hilfe der Dateinummer ist die geöffnete Datei eindeutig identifizierbar.

Beispiele für das Arbeiten mit Dateien im Web

- http://www.vbarchiv.net/tipps/tipp_301-textdateien-komfortabel-einlesen-und-schreiben.html

Textdatei in einem bestimmten Modus öffnen

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

   filenumber = FreeFile
   Open filename For Input As #filenumber
End Function
```

- Der Modus Input öffnet die Datei zum Lesen.
- Der Modus Output öffnet die Datei zum Schreiben.

Inhalt lesen

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

   filenumber = FreeFile
   Open filename For Input As #filenumber
   dateiinhalt = Input$(LOF(filenumber), #filenumber)
End Function
```

- Mit Hilfe der Funktion Input$ wird der Inhalt der Datei gelesen.
- Das erste Argument legt die Länge des zu lesenden Abschnitts fest.
- Das zweite Argument legt die zu lesende Datei fest.

Länge des Abschnitts

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

   filenumber = FreeFile
   Open filename For Input As #filenumber
   dateiinhalt = Input$(LOF(filenumber), #filenumber)
End Function
```

- Die Funktion LOF gibt die Größe einer Datei in Bytes zurück.

Datei schließen

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

   filenumber = FreeFile
   Open filename For Input As #filenumber
   dateiinhalt = Input$(LOF(filenumber), #filenumber)
   Close
End Function
```

Text in VBA...

- enthält Buchstaben, Zahlen, Satzzeichen etc.
- ist vom Datentyp String.
- muss in Anführungsstriche gesetzt werden.
- wird mit einem anderen Text mit Hilfe des kaufmännischen Unds verbunden.

Text-Zeichen in VBA...

- werden mit Hilfe von Zahlen im Bereich von 0 bis 255 codiert.
- Die ersten 128 Zeichen entsprechen dem ASCII-Zeichensatz. Dieser Zeichensatz enthält alle Zeichen der US-amerikanischen Tastatur.
- Die Zeichen von 128 bis 255 sind Sonderzeichen wie zum Beispiel ö, ä etc.
- Die ASCII-Codierung eines Zeichens kann mit Hilfe der Funktion ASC(zeichen) ermittelt werden.
- Ein ASCII-Code kann mit Hilfe der Funktion CHR(code) in ein Zeichen umgewandelt werden.

Ausschnitt aus der ASCII-Zeichentabelle

0 =		18 = ↕		36 = $		54 = 6		72 = H		90 = Z		108 = l	
1 =	☺	19 = ‼		37 = %		55 = 7		73 = I		91 = [		109 = m	
2 =	☻	20 = ¶		38 = &		56 = 8		74 = J		92 = \		110 = n	
3 =	♥	21 = §		39 = ’		57 = 9		75 = K		93 =]		111 = o	
4 =	♦	22 = ▬		40 = (		58 = :		76 = L		94 = ^		112 = p	
5 =	♣	23 = ↨		41 =)		59 = ;		77 = M		95 = _		113 = q	
6 =	♠	24 = ↑		42 = *		60 = <		78 = N		96 = ‘		114 = r	
7 =	•	25 = ↓		43 = +		61 = =		79 = O		97 = a		115 = s	
8 =	◘	26 = →		44 = ,		62 = >		80 = P		98 = b		116 = t	
9 =	◇	27 = ←		45 = -		63 = ?		81 = Q		99 = c		117 = u	
10 =	◙	28 = ∟		46 = .		64 = @		82 = R		100 = d		118 = v	
11 =	♂	29 = ↔		47 = /		65 = A		83 = S		101 = e		119 = w	
12 =	♀	30 = ▲		48 = 0		66 = B		84 = T		102 = f		120 = x	
13 =	♪	31 = ▼		49 = 1		67 = C		85 = U		103 = g		121 = y	
14 =	♫	32 =		50 = 2		68 = D		86 = V		104 = h		122 = z	
15 =	☼	33 = !		51 = 3		69 = E		87 = W		105 = i		123 = {	
16 =	►	34 = "		52 = 4		70 = F		88 = X		106 = j		124 = ¦	
17 =	◄	35 = #		53 = 5		71 = G		89 = Y		107 = k		125 = }	

Text-Informationen

```
Dim text As String
text = "30159 Hannover"
```

Funktion	Erläuterung	Rückgabewert
Len(text)	Anzahl Zeichen	14 (Ganzzahl)
InStr(text, " ")	Suche nach einem Teilstring in einem Text und Rückgabe der Anfangsposition	6 (Ganzzahl)

Teilstring aus einem Text

```
Dim text As String
text = "30159 Hannover"
```

Funktion	Erläuterung	Rückgabewert
Left(text, 5)	Teilstring mit x Zeichen, beginnend links	30159 (String)
Right(text, 8)	Teilstring mit x Zeichen, beginnend rechts	Hannover (String)
Mid(text, 4, 2)	Teilstring mit x Zeichen, beginnend an der angegebenen Position	59 (Text)
Split(text, " ")	Teilung des Textes an einem bestimmten Zeichen	„30159“, „Hannover“ (Liste von Elementen)

VBA-Konstanten für Zeilenende etc.

```
Dim text As String
text = "30159 Hannover" & vbCrLF
```

VBA-Konstante	ASCII-Zeichen	Beschreibung
vbCrLf	Chr(13) & Chr(10)	Wagenrücklauf und Zeilenvorschub
vbNewLine	Chr(13) & Chr(10)	Plattformspezifischer Zeilenumbruch
vbCr	Chr(13)	Wagenrücklauf
vbLf	Chr(10)	Zeilenvorschub
vbTab	Chr(9)	Tabulatorzeichen
vbBack	Chr(8)	Rückschrittzeichen

Fehler abfangen

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

   On Error GoTo whichErr

   filenumber = FreeFile
   Open filename For Input As #filenumber
   dateiinhalt = Input$(LOF(filenumber), #filenumber)
   Close
End Function
```

Wenn ein Fehler auftritt ...

On Error GoTo whichErr

- Falls ein Fehler auftritt, gehe zu der angegebenen Marke.
- Der Name der Marke ist frei wählbar
- Die Anweisung wird häufig vor allen Anweisungen gesetzt, um Fehler abzufangen und an einer Stelle im Programm zu behandeln.

Fehler behandeln

```
Function GetText(filename As String) As String
   On Error GoTo whichErr
   filenumber = FreeFile
whichErr:
   Select Case Err.Number
        Case 0: GetText = dateiinhalt
        Case 62:
            Seek #filenumber, 1
            dateiinhalt = Input$(LOF(filenumber), #filenumber)
            GetText = dateiinhalt
        Case Else:
            MsgBox "Error: " & Err.Description, vbOKOnly + vbInformation
   End Select
```

Marke im Code setzen

whichErr:

- Der Name der Textmarke ist frei wählbar.
- Dem Name folgt der Doppelpunkt.
- Die Textmarke entspricht einem Lesezeichen.
- Eine Textmarke beendet nicht die Ausführung des Codes!

Objekt „Fehler“ (Err)

- Das Objekt Err enthält Informationen zu dem aktuellen Fehler.
- Das Attribut .Number gibt die Fehlernummer zurück. Unter folgenden Punkten werden auffangbare Nummern aufgelistet:
  - http://support.microsoft.com/kb/142138/DE/
  - Suche nach dem Stichwort „Auffangbare Fehler“ in der Hilfe.
- Das Attribut .Description gibt eine Standardbeschreibung des Fehlers zurück.
- Die Methode .Clear setzt das Objekt zurück.

Fallunterscheidung

```
Select Case variable
        Case wert:
              Anweisung
        Case wert:
            Anweisung
        Case Else:
            Anweisung
End Select
```

- In Abhängigkeit des Wertes einer Variablen wird zu verschiedenen Anweisungen verzweigt.

Kein Fehler

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

   On Error GoTo whichErr
   filenumber = FreeFile
   Open filename For Input As #filenumber
   dateiinhalt = Input$(LOF(filenumber), #filenumber)

whichErr:
   Select Case Err.Number
        Case 0:
            GetText = dateiinhalt          ' Der Dateiinhalt wird zurückgegeben
   End Select
```

Fehler: Lesen nach dem Dateiende

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String
   ...
whichErr:
   Select Case Err.Number
        Case 62:
            Seek #filenumber, 1              ' Der Dateizeiger wird gesetzt
            dateiinhalt = Input$(LOF(filenumber), #filenumber)
            GetText = dateiinhalt
   End Select
```

Alle anderen Fehler

```
Function GetText(filename As String) As String
   Dim filenumber As Integer
   Dim dateiinhalt As String

   On Error GoTo whichErr
   filenumber = FreeFile
   Open filename For Input As #filenumber
   dateiinhalt = Input$(LOF(filenumber), #filenumber)

whichErr:
   Select Case Err.Number
        Case Else:
            MsgBox "Error: " & Err.Description, vbOKOnly + vbInformation
  End Select
```

Ausgabe am Bildschirm

MsgBox "Error: " & Err.Description, vbOKOnly + vbInformation

MsgBox prompt [, buttons + information] [, title] [, helpfile, context])

- Der Funktion können folgende Argumente übergeben werden:
  - Der anzuzeigende Text prompt. Der Text informiert den Benutzer über Fehler etc.
  - Welche Schaltflächen (buttons; vbOKOnly) werden im Meldungsfenster angezeigt? Welche Icons werden in dem Fenster als Information für den Benutzer angezeigt (vbInformation)? Alle Elemente werden mit dem Pluszeichen verbunden.
  - Ein Text (title) für die Titelleiste des Fensters.

Leere Zeilen ermitteln

```
Function getEmptyCell() As String
    Dim blatt As Worksheet
    Dim zelle As Range
    Dim spalte As Long
    Dim zeile As Long
    Set blatt = ThisWorkbook.ActiveSheet
    spalte = 1
    zeile = 1

    Do
        Set zelle = blatt.Cells(zeile, spalte)
        spalte = spalte + 1
    Loop While zelle.Value <> ""

    getEmptyCell = zelle.Address
End Function
```

Auflistungen von Objekten ...

- enden immer mit einem „s“
- enthalten Elemente vom gleichen Objekttyp.
- Beispiele:
  - Cells enthält alle Zellen in einem bestimmten Bereich.
  - Worksheets listet alle Tabellenblätter in einer Arbeitsmappe auf.

… vollständig durchlaufen

```
Dim zelle As Range


For  Each zelle In arbeitsblatt.Rows(zeilennr).Cells
    txtZeile = txtZeile & zelle.Value
Next
```

- For [Zelle] In [arbeitsblatt.Zeile(1).Zelle]
- Mit Hilfe von Next wird der nächste Schleifendurchlauflauf gestartet.
- Die Liste wird von der ersten Zeile bis zur letzten Zeile für die angegebene Zeile durchlaufen.

Objektvariablen ...

- verweisen auf ein bestimmtes Objekt in Excel.
- enthalten eine Referenz auf die Excel-Anwendung selbst oder auf Elemente in einer Arbeitsmappe.
- werden für Objekte definiert, die häufig in einem Programm genutzt werden.
- In dem vorherigen Beispiel ist die Variable blatt vom Datentyp Worksheet (Arbeitsblatt).

… deklarieren

Dim blatt As Worksheet

Zugriff objektvariablename As Objekttyp

- Jede Objektvariable hat einen eindeutigen Namen. In diesem Beispiel blatt.
- Objektvariablen sind von einem bestimmten Typ „Objekt“ (Worksheet; Arbeitsblatt).
- Auf diese Variable kann nur zwischen Sub und End Sub zugegriffen werden. Es ist eine private Variable der Ereignisprozedur.

Verweis auf ein Objekt speichern

```
Set blatt = ThisWorkbook.ActiveSheet
Set objektvariablename = Objektverweis
```

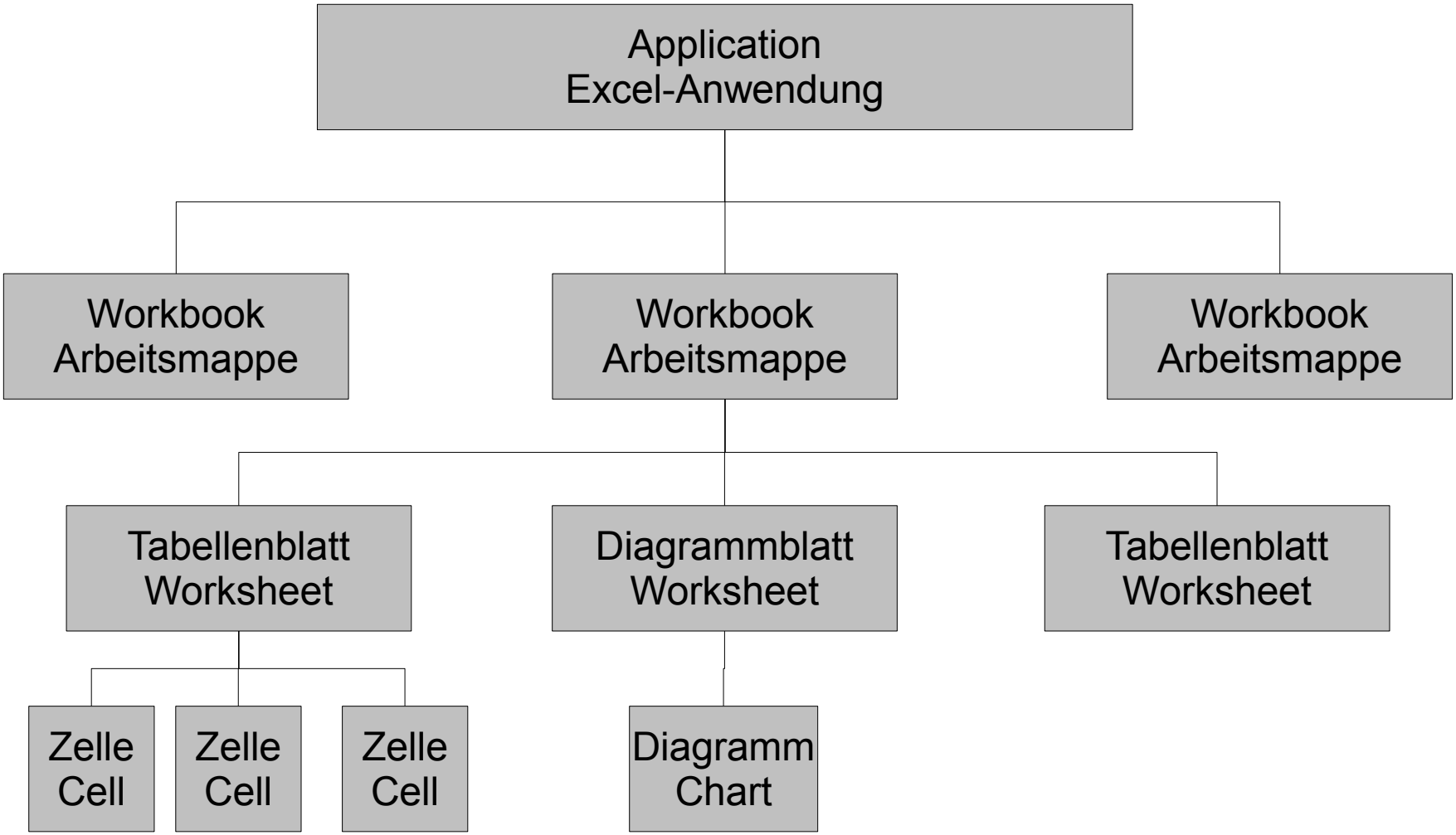
- Die Zuweisung muss mit dem Schlüsselwort `Set` eingeleitet werden.
- Mit Hilfe des Gleichheitszeichens wird der Variablen ein Verweis auf ein bestimmtes Objekt zugewiesen.
- In diesem Beispiel verweist die Variable `blatt` auf das aktive Arbeitsblatt in der zum Code gehörenden Arbeitsmappe.

Hierarchie von Objekten

```
Set blatt = ThisWorkbook.ActiveSheet
```

- Eine Arbeitsmappe besteht aus mindestens ein Arbeitsblatt.
- Diese Hierarchie spiegelt sich auch in dem Verweis wieder.
- Das übergeordnete Objekt wird mit dem untergeordneten Objekt durch einen Punkt verbunden.

Hierarchien im Excel-Objektmodell



Blätter einer Arbeitsmappe ...

- werden in der Auflistung Sheets gesammelt. Über diese Auflistung kann einer Arbeitsmappe ein neues Blatt hinzugefügt werden.
- können Arbeitsblätter (Worksheet) sein. Arbeitsblätter bestehen aus Zeilen und Spalten.
- können Diagrammblätter (Chart) sein. Das Diagramm wird als DIN A4-Blatt angezeigt. Diagrammblätter werden für Präsentationen oder Ausdrucke genutzt.

Arbeitsblätter in VBA

```
Dim arbeitsblatt As Worksheet

' Das aktive Blatt
Set arbeitsblatt = ThisWorkbook.ActiveSheet

' Verweis auf das Arbeitsblatt TB1
Set arbeitsblatt = ThisWorkbook.Worksheets("TB1")
```

Cells ...

- enthält alle Zellen eines Arbeitsblattes.
- beschreibt mit Hilfe des Zeilen- und Spaltenindex eindeutig eine Zelle.

... in VBA

```
Function getEmptyCell() As String
    Dim blatt As Worksheet
    Dim zelle As Range
    Dim spalte As Long
    Dim zeile As Long
    dim pos as String

    Set blatt = ThisWorkbook.ActiveSheet
    spalte = 1
    zeile = 1
    Set zelle = blatt.Cells(zeile, spalte)

End Function
```

Eigenschaften von Zellen

- .Value speichert den Inhalt einer Zelle. Der Wert kann mit Hilfe von VBA gelesen (wert = zelle.Value) oder verändert (zelle.Value = wert) werden.
- .Address gibt die Position der Zelle in dem Arbeitsblatt zurück. Der Wert kann nur Hilfe von VBA gelesen (pos = zelle.Address). Zum Beispiel die Positionsangabe "A1" bezeichnet eine Zelle in der ersten Zeile und ersten Spalte.

Range ...

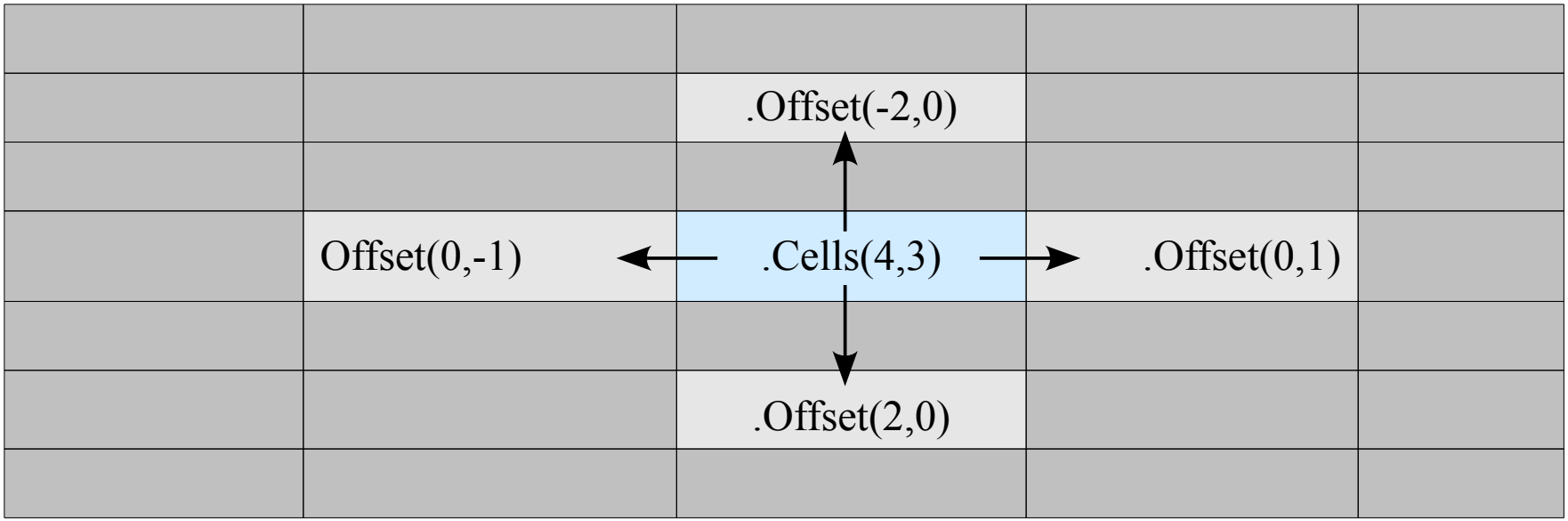
Range("A1")				
	Range("B3:C4")			

- beschreibt eine Zelle, Zeile, Spalte oder Zellbereich.
- nimmt auf einen Zellbereich Bezug.
- kann durch die Angabe von Zellen und Spalten für eine Zelle beschrieben werden.

Zellbezug in Abhängigkeit einer anderen Zelle

- Die Methode .Offset legt einen Zellbereich in Abhängigkeit einer anderen Zelle fest.
- Der Methode werden folgende Argumente übergeben:
  - Relative Verschiebung der Zeile.
  - Relative Verschiebung der Spalte.

Beispiel



Eine Verschiebung um ...

- Null. Die Zeilen- oder Spaltennummer wird übernommen.
- einen positiven Wert. Die Zeilennummer wird um den angegebenen Wert nach unten verschoben. Die Spaltennummer wird um den angegebenen Wert addiert und dementsprechend nach rechts verschoben.
- einen negativen Wert. Die Zeilennummer wird um den angegebenen Wert nach oben verschoben. Die Spaltennummer wird um den angegebenen Wert subtrahiert und dementsprechend nach links verschoben.

Beliebig oft Anweisungen ausführen

```
Function getEmptyCell() As String
    Dim blatt As Worksheet
    Dim zelle As Range
    Dim spalte As Long
    Dim zeile As Long
    Set blatt = ThisWorkbook.ActiveSheet
    spalte = 1
    zeile = 1

    Do
        Set zelle = blatt.Cells(zeile, spalte)
        spalte = spalte + 1
    Loop While zelle.Value <> ""

End Function
```

Erläuterung

- Eine Schleife mit beliebigen Durchläufen beginnt mit Do.
- Die Schleife endet mit den Schlüsselwort Loop. Dieses Schlüsselwort startet den nächsten Durchlauf.
- Die Abbruchbedingung für die Schleife kann im Kopf (Do) oder im Fuß (Loop) stehen. In diesem Beispiel wird am Ende der Schleife die Bedingung abgeprüft. Die Anweisungen in der Schleife werden mindestens einmal ausgeführt.

Abbruchbedingung

While zelle.Value <> ""

- Solange die Bedingung erfüllt ist...
- In diesem Beispiel läuft die Schleife solange bis in der ersten Zeile die Zelle leer ist.

… hinter dem benutzten Bereich anfügen

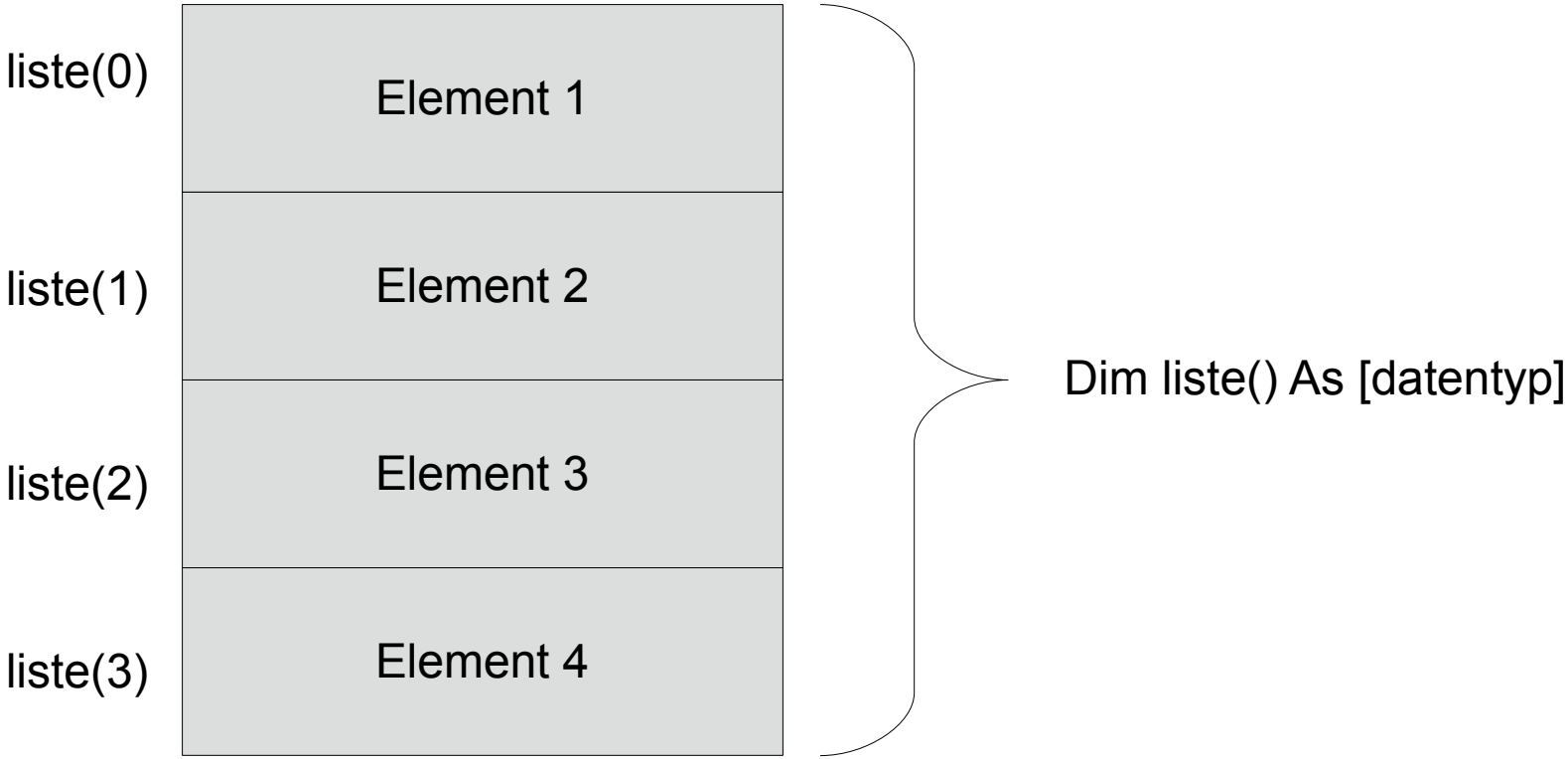
```
Function EmptyCell() As String
    Dim blatt As Worksheet
    Dim endZeile As Integer
    Dim zelle As Range

    Set blatt = ThisWorkbook.ActiveSheet

   ' Letzte benutzte Zeile ermitteln
    endZeile = blatt.UsedRange.SpecialCells(xlCellTypeLastCell).Row

  ' In der ersten Spalte, in der übernächsten Zeile wird der Import eingefügt
   Set zelle = blatt.Cells(endZeile + 2, 1)

    EmptyCell= zelle.Address
End Function
```

Felder (Array) ...

```
Dim allZeilen() As String


allZeilen = Split(dateiinhalt, Chr(13) & Chr(10))
allZeilen(1) = "Text"
```

- sind benutzerdefinierte Listen.
- sind Container für Elemente von einem bestimmten Datentyp.
- können beliebig lang sein.
- Die Elemente in einem Feld werden von 1 bis n nummeriert. Jedes Element hat einen eindeutigen Index,

Aufbau


liste(0)
Element 1
liste(1)
Element 2
liste(2)
Element 3
liste(3)
Element 4
Dim liste() As [datentyp]

… definieren

```
Dim allZeilen() As String
Dim zielSpalte() As Integer
Dim feldname(4) As Long
```

```
Dim allZeilen As String
Dim zielSpalte As Integer
Dim feldname As Long
```

- Jedes Feld hat wie eine Variable …
  - einen eindeutigen Namen. Dieser Name ist frei wählbar.
  - einen bestimmten Datentyp (As). Die verschiedenen Datentypen werden in der Hilfe unter *Visual Basic-Sprachverzeichnis - Datentypen* aufgelistet.
- Aber dem Feldnamen folgen im Gegensatz zu Variablen die runden Klammern. In diesen runden Klammern wird die Anzahl der Elemente angegeben oder nicht.

Felder unbekannter Länge dimensionieren

```
Dim zielSpalte() As Integer


ReDim zielSpalte(10)
ReDim Preserve zielSpalte(10)
```

- Die Anzahl der Elemente wird immer in runden Klammern als Ganzzahl angegeben.
- Mit Hilfe von ReDim wird eine Liste mit x Elementen erzeugt. Die Elemente sind leer.
- Mit Hilfe von ReDim Preserve wird eine Liste mit x Elementen erzeugt. Vorhandene Elemente werden nur gelöscht, wenn die Liste gekürzt wird.

… vollständig durchlaufen

```
Dim element As Variant


For  Each element In allZeilen
    txtZeile = txtZeile & allZeilen
Next
```

- For [element] In [Feld]
- Mit Hilfe von Next wird der nächste Schleifendurchlauflauf gestartet.
- Die Liste wird von dem ersten Element bis zum letzten Element für das angegebene Feld durchlaufen.
- Die Variable, die als Platzhalter für die Elemente arbeitet, muss vom Typ Variant sein.

… mit Hilfe einer Zählschleife durchlaufen

```
Dim spalte As Integer


For  spalte = 1 In Ubound(zielSpalte)
    zelle.Value = zielSpalte(spalte)
Next spalte
```

- Der Zählvariablen wird ein Anfangswert zugewiesen (spalte = 1).
- Es wird das Intervall 1 bis Index des letzten Elementes (Ubound([feld])) durchlaufen.
- Mit Hilfe von Next wird der nächste Schleifendurchlauflauf gestartet.