

MAKROPROGRAMMIERUNG

DOMINIK VOGT

Kontakt: Dominik.Vogt@fu-berlin.de

VBA und Makroprogrammierung

- Unterlagen:

<http://userpage.fu-berlin.de/vodo/vbakurs>

- Aufbau

- Teil 1: Einführung
- Teil 2: VBA Grundlagen
- Teil 3: Excel-Objektmodell
- Teil 4: VBA Vertiefung

- Pausen:

10:45-11:00, 12:30-13:30, 15:15-15:30, Schluss: 17:00

1. TEIL: EINFÜHRUNG

Makros aktivieren und speichern

Makros aufzeichnen und verwenden

Makros selbst erstellen

Teil 1: Einführung

Makros aktivieren und speichern

- Makros zugänglich machen
 - Makro-Werkzeuge einblenden
 - Register «Entwicklertools» über Menüband anpassen
 - Makros in Excel erlauben
 - Sicherheitscenter: Makros aktivieren
 - Makros speichern
 - Makroarbeitsmappe: *.xlsm (oder *.xlsb)
 - Makros mit Arbeitsmappe *überall* verfügbar
 - Persönliche Arbeitsmappe
 - Makros an diesem Rechner *immer* verfügbar
 - Weitere Möglichkeit: Add-In (*.xlam)
-

Teil 1: Einführung

Makros aufzeichnen

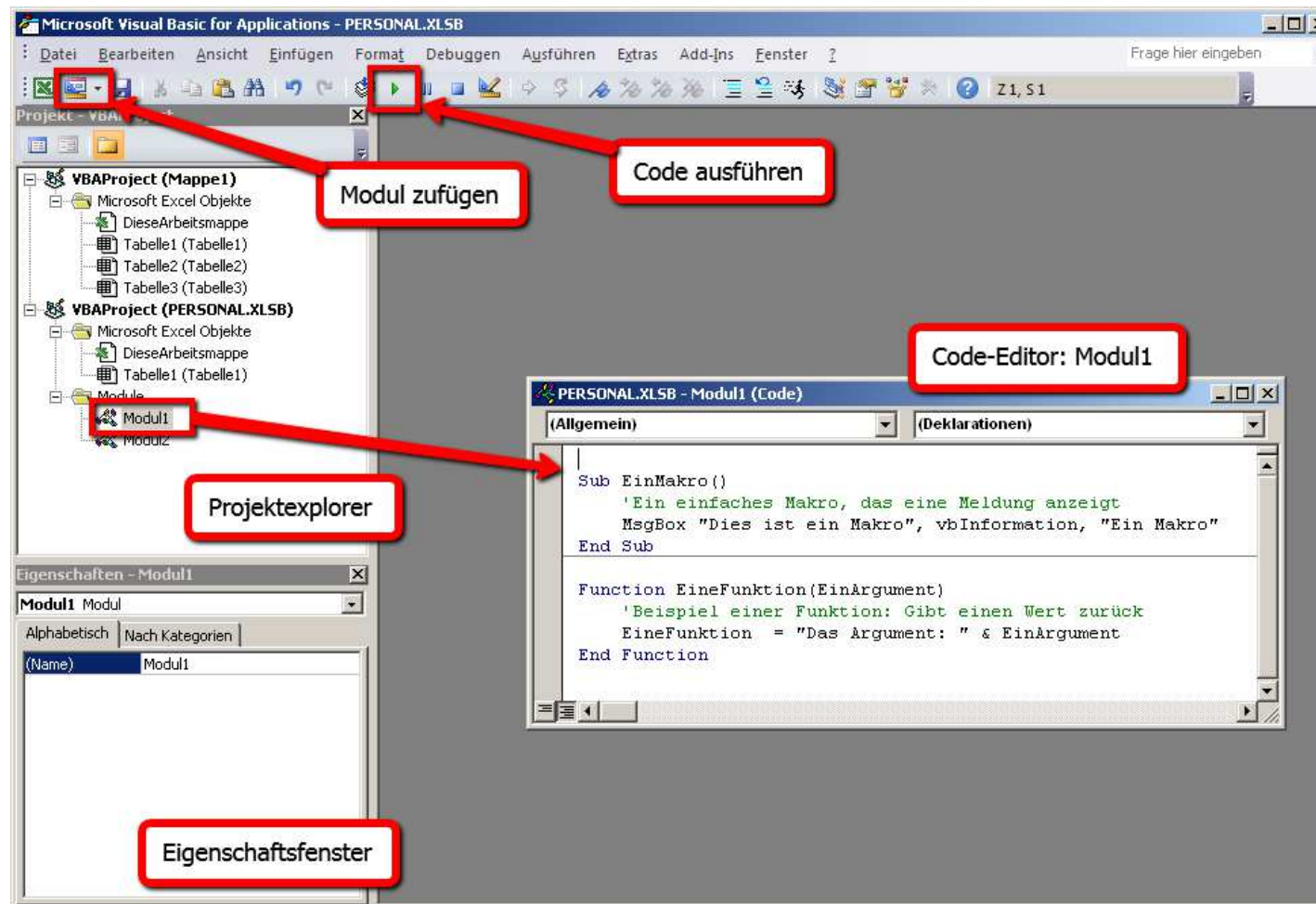
- Makro Aufzeichnung:
 - Entwicklertools: "Makro aufzeichnen"
 - Namen vergeben und Speicherort wählen
 - Namen: eindeutig, ohne Leer- und Sonderzeichen
 - Jede Aktion wird wiederholt
 - Auch Aktivieren eines Elementes → unerwünschte Effekte
 - Ausgangspunkt: aktives Element
 - Beim Abspielen werden alle Aktionen ausgehend vom aktiven Element ausgeführt: Zelle oder Teile eines Diagramms
 - Für Zellen: relative oder absolute Aufzeichnung
 - Makros Abspielen: Makro-Dialog (ALT+F8)
-

Teil 1: Einführung

Makros anpassen und erweitern

- Aufzeichnung von Makros als VBA-Befehle
 - Anpassen von Makros: VBA-Befehle ändern im VBA-Editor
 - "Entwicklertools": "Visual Basic"
 - Makros werden im VBA-Editor in Modulen geschrieben
 - Ein Modul ist Teil einer Arbeitsmappe
 - und kann ein oder mehrere Makros enthalten.
 - Zweck:
 - Kontrolle und Korrektur des aufgezeichneten Makros
 - Aufgezeichnete Befehle für eigene Makros verwenden
 - Methoden und Objekte für VBA finden
 - Ausführen eines Makros: F5 oder dem Start-Button
 - Einfügemarke muss im Makro sein
 - Schrittweise ausführen: F8 oder Menü: "Debuggen"
-

Teil 1: Einführung VBA-Entwicklungsumgebung



Hilfsmittel:

→ "Ansicht"

- Lokal-Fenster
- Direktfenster

→ "Debuggen"

- Einzelschritt
- Überwachung
- Haltepunkt

Code:

Blau: Schlüsselwörter

Grün: Kommentare

Schwarz: Anweisungen

Teil 1: Einführung

Beispiel: Makro anpassen

- Aufzeichnen HalloWelt-Makro: Start in A1, "Hallo Welt" in C4
 - Das HalloWelt-Makro: Sub HalloWelt() ... End Sub
 - Zwei Aktionen → zwei Anweisungen:
 - Die Zelle C4 wurde ausgewählt
 - Range("C4").Select
 - Objekt: "Range("C4")" → Ein Zellbereich im Arbeitsblatt
 - Methode: "Select" → Auswählen
 - Der Punkt ruft Methode oder Eigenschaft eines Objektes (Objektvariable) auf
 - Text wurde in die Zelle eingegeben

```
ActiveCell.FormulaR1C1 = "Hallo Welt"
```

 - Objekt: "ActiveCell" = die aktive Zelle
 - Eigenschaft: "FormulaR1C1" → Inhalt der Zelle, ihr wird ein Wert zugewiesen
 - Zwei Anpassungen:
 1. Keine Auswahl der Zelle: Text in aktive Zelle eingeben
 2. Text von C4 in aktive Zelle schreiben
-

Teil 1: Einführung

Makros programmieren

- Das **Makro** ist ein mehrzeiliger Textblock
 - beginnt mit der Zeile: "**Sub** *Name* ()"
 - und endet mit der Anweisung: "**End Sub**"
 - dazwischen stehen zeilenweise Anweisungen (Befehle)

```
Sub NameDesMakros()  
    meinVariable = "Dies ist ein Makro!"  
    MsgBox meinVariable  
End Sub
```
 - Der Name kann frei vergeben werden
 - keine Leer- oder Sonderzeichen
 - Nach dem Namen müssen leere Klammern stehen
 - In VBA ist ein Makro eine Prozedur, in den Klammern stehen zu übergebende Argumente. Ein Makro darf keine Argumente verlangen, daher leere Klammern
 - Makros werden in Modulen geschrieben
-

Teil 1: Einführung

Syntax und Namenskonventionen

- VBA gibt Regeln vor, wie Code formuliert werden muss: Syntax
 - **Grundlegende Syntax**
 - Eine Anweisung pro Zeile
 - **Doppelpunkt** : Anweisungstrennzeichen (zwei Anweisungen in einer Zeile)
 - **Unterstrich** _ Fortsetzungszeichen (eine Anweisung über zwei Zeilen)
 - Gross-/Kleinschreibung ist nicht signifikant
 - Das **Hochkomma** ' ist das Kommentarzeichen.
 - Alles in einer Zeile nach dem Hochkomma wird nicht verarbeitet.
 - Text als Wert muss in Anführungszeichen geschrieben werden: "Text"
 - **Namenskonventionen** zur Benennung von Variablen und Makros:
 - Erlaubte Zeichen: Buchstaben, Zahlen und Unterstrich (bis zu 255)
 - Das erste Zeichen muss ein Buchstabe sein
 - Manche Begriffe sind reserviert (Schlüsselwörter)
-

Teil 1: Einführung

Excel-Objekte verwenden

- Die Elemente von Excel sind als Variablen für sog. Objekte verfügbar.
- Objekt haben Eigenschaften und Methoden.
 - Mit einem Punkt nach der Objektvariable greift man auf diese zu:
`Objekt.Eigenschaft`
 - Standardeigenschaft: Ohne Punkt und Eigenschaftsname
 - Standardobjekte: Nur Eigenschaftsname (ohne Objekt)
`Range("A2") ↔ ActiveSheet.Range("A2").Value`
 - "**With**" fasst mehrere Anweisungen für ein Objekt zusammen:
`With Objektname`
`.Eigenschaft = wert`
`End With`
- Variablen für Objekte verweisen auf diese (Referenz, keine Kopie)
- Eigenschaften von Objekten können selbst Objekte sein
`EinObjekt.ObjektAlsEigenschaft.Eigenschaft`

Teil 1: Einführung

Wichtige Verweise auf Excel-Objekte

- Eine Zelle als Excel-Objekt
 - **Range("A2")**: die Zelle A2
 - **Cells(2, 1)**: auch die Zelle A2
 - **ActiveCell**: Variable für die aktive Zelle
 - **Selection**: die aktuelle Auswahl (Zelle, Chart usw.)
- Wert einer Zellen abrufen oder auch ändern
 - `Range("A1").Value = 25`
- Andere Excel-Objektvariablen
 - **Arbeitsmappe**: `ActiveWorkbook` und `ThisWorkbook`
 - Alle Tabellen: `Worksheets`
 - **Tabelle**: `ActiveSheet` und `Worksheets("Table1")`
 - Alle Zeilen der Tabelle: `ActiveSheet.Rows`
 - **Diagramm**: `ActiveChart`

Teil 1: Einführung

Makros verwenden und einbinden

- Makros verwenden:
 - Aufrufen über Makro-Dialog: "Makros"
 - Tastenkürzel: Alt+F8
 - Einbinden: Tastenkürzel und Menüband
 - Makros über Schaltfläche auf Menüband
 - Optionen: "Menüband anpassen"
 - Makros direkt mit einem Tastenkürzel
 - Makro-Dialog: "Optionen"
 - Einbinden: Zeichenobjekt oder Steuerelement
 - Kontextmenü: "Makro zuweisen..."
 - Als Funktionen im Arbeitsblatt
 - In Zelle als Formel: "=EigeneFunktion(Argumente)"

Übung 1: Makro erstellen

Ein erstes eigenes Makro schreiben

- Aufgabe:
 - Erstellt ein Makro, das euch den Wert der aktiven Zelle in einem Meldungsfenster anzeigt.
 - Zellwert einer Variable zuweisen
 - In der Meldung dem Zellwert noch den Text "Der Wert der aktiven Zelle ist: " voranstellen
 - Den Wert der aktiven Zelle in die Zelle A1 schreiben
- Hinweise:
 - Aktive Zelle: `ActiveCell`; Zelle A1: `Range("A1")`
 - Methode um eine Meldung anzuzeigen: `MsgBox "Text der Meldung"`
 - Verkettungsoperator um Texte zu kombinieren: `&`
- Makro verwenden:
 - Testen mit F8 im VBA-Editor, Lokal-Fenster anschauen
 - Aufrufen durch Makro-Dialog
 - Makro durch Formularsteuerelement "Schaltfläche" auf Arbeitsblatt starten
 - Makro in Menüband einbinden

2. TEIL: VBA GRUNDLAGEN

Variablen und Funktionen

Kontrollfluss: Verzweigungen und Schleifen

Benutzerdefiniert Funktionen

Teil 2: VBA Grundlagen

Variablen

- Variablen speichern Werte oder verweisen auf Objekte
 - Gültig nur in der Prozedur, in der definiert
- Definition einer Variable durch Wertzuweisung mit "=":
`ErsteVariable = 5`
 - Textwerte (String) in doppelte Anführungszeichen:
`EineTextVariable = "Hallo"`
 - Leerer Text (Nullwert einer Textvariable): `""`
 - Bei Objektvariablen: Wertzuweisung mit "Set" und "=":
`Set objVar = ActiveCell`
- Eine explizite Definition ist mit "Dim" möglich
`Dim varName As String`
 - Explizite Definition ist bei Objektvariablen hilfreich, da dann die Autovervollständigung beim Schreiben hilft

Teil 2: VBA Grundlagen

Variablen verwenden

- Mit Variablen und Werten können Formeln geschrieben werden:
 - mathematischen Operatoren: + - * / ^ (Potenz)
 - Klammerung mit runden Klammern
`ZweiteVariable = ( 2 + ErsteVariable ) * 10`
 - Text kann mit dem Operator & verknüpft werden
`ZweiteTextVariable = EineTextVariable & " Welt"`
 - Eine Variable kann sich selbst zugewiesen werden:
`ZählerVariable = 1 + ZählerVariable`
 - Das Lokal-Fenster (unter "Ansicht) zeigen beim schrittweisen Ausführen (F8) den aktuellen Wert einer Variable
 - Ausgewählte Variablen: "Überwachung" (unter "Debuggen")
-

Teil 2: VBA Grundlagen

Funktionen (1)

- In VBA stehen vordefinierte Funktionen zu Verfügung. Idealtypisch führt eine Funktion eine Berechnung durch und gibt dessen Resultat als Rückgabewert zurück.
- Verwendet wird eine Funktion wie der Rückgabewert:
 - In einer Variablenzuweisung rechts des Gleichheitszeichens
 - Als Argument einer anderen Funktion
- Funktionen verlangen meist Argumente
 - Argumente stehen in Klammern nach dem Funktionsnamen
 - Mehrere Argumente werden durch Kommas getrennt
 - Argumente müssen in der richtigen Reihenfolge stehen
 - oder als benannte Argumente: `NameDesArguments := wert`
- Beispiel: Gerundete Wurzel (verschachtelte Funktionen):
`EineVariable = Round(Sqr(2), 2)`

Teil 2: VBA Grundlagen

Funktionen (2)

- Funktionen ohne Rückgabewert
 - mit "Call" aufgerufen:
`Call MsgBox("Dies ist der Text der Meldung")`
 - Oder ohne Call und ohne Klammern
`MsgBox "Die Meldung"`
 - Optionale Argumente: müssen angegeben werden
 - in Dokumentationen meist in eckigen Klammern.
 - Beispiel: `MsgBox(Meldung, [Gestaltung], [Titel])`
 - Aufruf mit leeren Kommas (Reihenfolge) oder benannt
`Call MsgBox ("Meldungstext", , "Der Titel")`
`Call MsgBox (Title:="Titel", Prompt:="Meldung")`
 - Die Excel-Funktionen sind keine VBA-Funktionen
 - Excel-Funktionen: WorksheetFunction-Objekt (langsamer)
-

Teil 2: VBA Grundlagen

Funktionen finden

- Funktionen sind in Funktionsbibliotheken gespeichert: "Objektkatalog" (unter Ansicht oder F2), Bibliothek: "VBA"
 - Einige wichtige Funktionen:
 - **Für Zahlen:** Abs(), Sgn(), Fix(), Round(), Rnd()
 - **Für Texte:** Len(), InStr(), Mid(), Replace(), Trim(), Left(), Right(), LCase(), UCase(), StrReverse() , Split(), Join(), Chr()
 - **Mathe:** Log(), Exp(), Sqr(), Sin(), Cos(), Tan()
 - **Meldungen:** MsgBox(), InputBox()
 - **Array:** Array(), LBound(), UBound()
 - **Datum:** Date(), Time(), Now(), DateSerial(), DateValue(), ...
-

Teil 2: VBA Grundlagen

Verzweigung mit IF..ELSE

- Eine IF-Anweisung steuert den Ablauf eines Makros.
 - Sie teilt den Makro-Code in Blöcke von Anweisungen und steuert über eine Bedingung, welcher Block verarbeitet wird.
 - Das Gerüst einer IF-Anweisung. Der Else-Block ist optional.
IF Bedingung **THEN**
 Anweisungen, wenn Bedingung wahr ist
[**ELSE**
 Anweisungen, wenn Bedingung falsch ist]
END IF
 - Bedingungen sind Ausdrücke, die wahr oder falsch sind:
 - Vergleichsoperatoren: =, >, <, <> (Ungleich), LIKE, IS
 - Logischen Operatoren: NOT, AND, OR
 - Funktionen, die wahr oder falsch zurück geben: IsNumeric(var)
 - Ergänzung: zweite Bedingung prüfen: **ELSEIF** Bedingung **THEN**
-

Übung 2: Verzweigungen

Zellen bedingt formatieren

- Aufgabe
 - Aktive Zelle rot formatieren, wenn sie leer ist, sonst gelb
- Hinweis:
 - Inhalt der aktiven Zelle: "ActiveCell.Value"
 - Inhalt einer Variable zuweisen (ist nicht zwingend)
 - Leer ist eine Zelle, wenn sie keinen Text enthält:
`ActiveCell.Value = ""` (leere Anführungszeichen)
 - für Formatierung Makros aufzeichnen und Befehle kopieren
 - Achtung: beim Aufzeichnen wird die Formatierung auf "Selection" nicht auf "ActiveCell" angewendet

Teil 2: VBA Grundlagen

Schleifen mit DO..LOOP

- **Do ... Loop**

- wiederholt den Anweisungsblock,
- bis sie explizit verlassen wird
- dazu wird eine Bedingung überprüft

- Mit einer If-Anweisung und **Exit Do**

DO

[Anweisungen]

zähler = zähler + 1

IF zähler > 10 THEN EXIT DO

[Anweisungen]

LOOP

- mit **Until** oder **While** bei der Do- oder Loop-Anweisung

LOOP UNTIL zähler > 10

- **Vorsicht: Endlosschleife, unterbrechen mit Strg-Pause**
-

Übung 3: Schleifen

Durch Zellen iterieren

- Aufgabe: Alle Werte eines Zellbereichs addieren. Summe in Meldung anzeigen
 - Hinweise:
 - Zellbereich: `Range("A1:A5")`; einer Variable zuweisen (mit "Set")
 - Tipp: Variable explizit definieren: `"Dim varZellen as Range"`
 - Do-Loop über alle Zellen:
 - Zähler/Index-Variable definieren, Schleife verlassen, wenn `Index > Anzahl Zellen`
 - Anzahl Zellen: `varZellen.Cells.Count`
 - Einzelne Zelle: `varZellen.Cells(index)`
 - Wert der einzelnen Zelle: `.`
 - Für Summe Variabel verwenden und diese mit Meldung-Funktion: `"MsgBox(meldung)"` anzeigen
-

Teil 2: VBA Grundlagen

Benutzerdefinierte Funktionen

- benutzerdefinierten Funktionen sind spezielle "Makros"
 - Sie stehen als Arbeitsblatffunktionen in Excel zur Verfügung
 - Im Excel-Funktionsassistenten unter "Benutzerdefiniert"
 - Grundgerüst: mit "**Function**" statt "Sub"

```
Function NameDerFunktion (arg1, arg2)
    NameDerFunktion = arg1 + arg2
End Function
```

 - Rückgabewert dem Funktionsnamen wie einer Variable zuweisen
 - Argumente in den Klammern nach dem Namen
 - Sie können auch in VBA selbst verwendet werden

```
eineVariable = MeineEigeneFunktion(EinArgument)
```

 - Gilt auch für Makros, dann auch mit Argumenten
-

Übung 4: Eigene Funktionen

Funktion Vektorlänge

- Aufgabe:
 - Es soll eine Funktion zur Berechnung einer Vektorlänge erstellt werden.
 - Formel: $\sqrt{a^2 + b^2}$
 - Die Funktion verlangt zwei Argumenten: a und b
- Hinweis:
 - Die Wurzel wird mit der Funktion Sqr(wert) berechnet.
 - Operator für Potenz ist: ^
 - Rückgabewert dem Funktionsnamen zuweisen
- Verwendung:
 - Werte für a und b in zwei Spalten schreiben, mittels Funktionsassistent Funktion in eine dritte Spalte einfügen.
 - Zweite Methode (Sub) schreiben und Rückgabewert in Meldung anzeigen

3. TEIL:

DAS EXCEL-OBJEKTMODELL

Objekte allgemein: Eigenschaften und Methoden

Arbeitsmappe: Workbook und Worksheet

Zellen: das Range-Objekt

Teil 3: Excel-Objektmodell

Objekte: Eigenschaften und Methoden

- Objekte bilden die "wirklichen" Gegenstände ab
- Objekte haben Eigenschaften und Methoden
 - Auf diese wird mittels "." zugegriffen: `ActiveCell.Value = ...`
 - Eigenschaften können selbst Objekte sein
- Objekte können auch Variablen zugewiesen werden.
 - Wertzuweisung bei Objektvariablen: `SET name = instanz`
 - Deklaration einer Objektvariable: `DIM name AS ObjektTyp`
- Auflistungen von Objekten: `Collection`
 - Element abrufen: Index oder Schlüssel in Klammern
 - Alle Elemente abrufen: `For Each...` (später)
- Objektvariable ohne Objekt: `Nothing`
 - `IF meinObj IS Nothing THEN ...`

Teil 3: Excel-Objektmodell

Hierarchie des Modells: Application

- VBA versucht alle Eigenschaften und Fähigkeiten von Excel abzubilden.
 - Eine solche Eigenschaft ist, dass Arbeitsmappen offen sind (.Workbooks)
 - Eine Fähigkeit wäre, diese zu berechnen (.Calculate)
 - Ein Workbook ist dann selbst ein Objekt mit Eigenschaften
 - Ausgehend von Excel selbst (Application) sind die Objekte hierarchisch:
 - Application
 - Workbook(s)
 - Worksheet(s)
 - Cells/ Rows/Columns
 - Range
 - ChartObjects
 - Chart
 - Shape(s)
 - Chart(s)
 - Window(s)
 - Excel selbst (Instanz)
 - Arbeitsmappen (Objekt und Auflistung)
 - Tabellenblätter (Objekt und Auflistung)
 - Zellen/Zeilen/Spalten (Auflistung)
 - Zellen oder Zellbereiche (Objekt und Instanzen)
 - eingebettete Diagramme (Auflistung)
 - Diagramm (Objekt, bzw. Instanz)
 - Zeichnungsebenen (Objekt und Auflistung)
 - Diagrammblätter (Objekt und Auflistung)
 - Fenster (Objekt und Auflistung)
- Spezielle Instanzen: ActiveWorkbook, ActiveCell, Selection usw.
- Das übergeordnete Objekt eines Objektes: ".Parent"
-

Teil 3: Excel-Objektmodell



Teil 3: Excel-Objektmodell

Workbooks und Workbook

- Workbook: Eine Arbeitsmappe als VBA-Klasse
 - Workbooks: Auflistung aller offenen Arbeitsmappen
 - Verwendung:

```
Dim MeineMappe as Workbook
Set MeineMappe = Workbooks("NameDerMappe")
```
 - Neue Arbeitsmappe

```
Set MeineMappe = Workbooks.Add
```
 - Speichern und schliessen

```
MeineMappe.SaveAs "MeineMappe.xlsx"
MeineMappe.Close
```
 - Öffnen einer Arbeitsmappe:

```
Set MeineMappe = Workbooks.Open(Pfad)
```
 - Zugreifen auf eine geöffnete Mappe

```
Set MeineMappe = Workbooks(Name|Index)
```
 - Aktive Arbeitsmappe: `ActiveWorkbook`
-

Teil 3: Excel-Objektmodell

Worksheets und Worksheet

- Worksheet: ein Tabellenblatt als VBA-Objekt
 - Worksheets: Auflistung aller Tabellen einer Workbook-Instanz (Eigenschaft)

```
Set eineTabelle = ActiveWorkbook.Worksheets("Tabelle1")
```
 - Tabellenblatt zufügen: Worksheets.Add
 - Aktives Arbeitsblatt: ActiveSheet
 - Achtung: kann auch Chart-Objekt sein
 - Aktivieren mit: .Activate
 - Eigenschaften von Worksheet:
 - Auflistungen von Zellen: .Rows, .Columns, .Cells
 - Ein Zellbereich: .Range
 - Name der Tabelle: .Name
 - Diagramme (. ChartObjects); Zeichnungsobjekte (.Shapes)
-

Teil 3: Excel-Objektmodell

Range

- Range ist die Klasse für Zellen oder Zellbereiche
- Die Worksheets-Auflistungen Cells, Columns und Rows geben ein Range-Objekt zurück
- Aktiver Zellbereich: ActiveCell
 - "Selection": die aktuelle Auswahl. Kann die aktive Zelle sein, aber auch ein anderes aktives Objekt wie ein Diagramm
- Range() ist eine Eigenschaft von Application oder Worksheet. Sie gibt die Instanz eines Zellbereichs zurück. Das Argument ist die Adresse in A1 Schreibweise.

```
Dim EinBereich as Range
Set EinBereich = ActiveCell
Set EinBereich = Range("A4:C10")
Set EinBereich = ActiveSheet.Cells(2,4)
Set EinBereich = ActiveSheet.Columns(2)
Set EinBereich = ActiveSheet.Columns("B:D")
```

Teil 3: Excel-Objektmodell

Range: Eigenschaften und Methoden

Wichtige Eigenschaften:

- | | |
|-------------------------|--|
| • Cells/Rows/Columns | Auflistung der Zellen |
| • Resize | Vergrößert den Bereich |
| • Offset | Verschiebt den Bereich |
| • Count | Anzahl Zellen im Zellbereich |
| • Value | Wert der Zelle oder einen Array der Werte |
| • Formula | Die Formel des Bereiches (Wert oder Array) |
| • Borders/Font/Interior | Formatierungen (selbst Objekte) |

Wichtige Methoden

- | | |
|--------------|---------------------------------------|
| • Select | Markiert den Bereich |
| • Clear | Löscht alle Inhalte |
| • Delete | Löscht den Bereich |
| • Insert | Fügt einen entsprechenden Bereich ein |
| • Copy/Paste | Kopieren/Einfügen eines Bereiches |
-

Teil 3: Excel-Objektmodell

Objekte erkunden

- Um die Excel-Objekte kennenzulernen gibt es Tricks:
 - Makros aufzeichnen:
 - Beim Aufzeichnen schreibt Excel die Methoden oder Eigenschaften von Objekten
 - Achtung: die aufgezeichneten VBA-Befehle sind nicht immer optimal
 - Explizite Definition der Objektvariable
 - VBA bietet dann Autovervollständigung
 - Gibt man den Punkt nach der Variable ein, erscheint eine Liste möglicher Eigenschaften und Methoden. Auswahl mit Tabulatortaste.
 - Schreibt man die öffnende Klammer einer Methode erscheinen Hinweise auf Argumente.
 - Überwachungsfenster (oder Lokal-Fenster):
 - Im Überwachungsfenster werden die Eigenschaften (nicht immer alle) und deren Werte angezeigt
 - Objektkatalog. Bibliothek: "Excel" und "Office"

Übung 5: Excel-Objektmodell

Arbeitsmappe und Tabellen

- Aufgabe:
 - Neue Tabelle zufügen und ihr einen Namen geben
 - In einem zweiten Makro auf diese Tabelle zugreifen
 - Text in Zelle C2 schreiben
 - Dann Tabelle aktivieren und Zelle auswählen
- Hinweise
 - Neue Tabelle zufügen: "Worksheets.Add".
 - direkt mit Set in einer Worksheet-Variable speichern (mit Dim definieren)
 - Name der Tabelle: .Name
 - Zugriff mit Tabellennamen über Worksheets-Auflistung
 - Zugriff auf Zelle mit .Range("C2")
 - Einer Variable ("Dim" mit Typ: Range) mit Set zuweisen
 - Tabelle aktivieren: .Activate
 - Zelle auswählen: .Select (für Zell-Objekt Range)

4. TEIL: WEITERFÜHRENDES VBA

Module und Prozeduren

Ereignisgesteuerte Programmierung und Formulare

Variablen und Datentypen

Datenfelder (Arrays) und Auflistungen (Collection)

Kontrollfluss: For..Next-Schleifen

Fehlerbehandlung

Teil 4: VBA

Module und Prozeduren

- **Prozeduren** sind Teile eines grösseren Programmes, die unabhängig ausgeführt werden können.
 - Zwei Typen: Funktionen oder Methoden
 - Methode: Sub ... End Sub

```
Sub name ([arg1, arg2])  
    ...  
End Sub
```
 - Funktion: Function ... End Function

```
Function name ([arg1, arg2]) [as Datentyp]  
    ...  
    name = wert  
End Function
```
 - Vorzeitiges Verlassen einer Prozedur: **Exit Sub | Function**
 - Prozeduren können aus anderen Prozeduren aufgerufen werden.
 - auch sich selbst: Rekursion.
 - Vorsicht: Variablen, als Argumente übergeben, werden verändert
 - **Module** fassen Prozeduren zusammen
 - Genauer: Module sind Objekte mit Prozeduren als Methoden
 - Verweis auf Prozeduren in anderen Modulen:
Modulname.Prozedurname
-

Teil 4: Formulare

Ereignisgesteuerte Programmierung

- Manche Objekte lösen Ereignisse aus.
 - Ereignisse werden mit speziellen Prozeduren abgefangen.
Das Objekt ruft diese Prozedur automatisch auf
 - Die Prozedur wird im Objektmodul geschrieben
 - Ereignisse gibt es:
 - in Excel:
 - wenn eine Arbeitsmappe geöffnet oder geschlossen wird
 - wenn eine Zelle ausgewählt oder das Tabellenblatt gewechselt wird
 - in selbst entworfenen Formularen
 - wenn ein Benutzer in einem Formular Text eingibt oder einen Button drückt
 - Formulare können erstellt werden:
 - In einem eigenen Fenster: UserForm
 - In einem Arbeitsblatt: ActiveX-Steuerelemente
-

Teil 4: Formulare

UserForm erstellen

- Formular erstellen: UserForm einfügen ("Einfügen")
 - Steuerlemente auf Userform platzieren (Werkzeugsammlung)
 - Eigenschaften von Form und Elementen festlegen (Eigenschaftsfenster)
 - Code für UserForm in sog. Objekt-Modul
 - Eigenschaften von Steuerelementen abrufen oder setzen
`MeinControl.Caption = "Neuer Wert"`
 - Ereignis-Prozedur für Steuerelement erstellen:
 - Doppelklick auf Steuerelement
 - DropDowns oben im Objektmodul-Fenster (eines für Element, eines für Ereignis)
`Private Sub MeinControl_Ereignis([argumente])`
 - Starten mit F5 oder mit einem Makro: `MeineForm.Show`
 - UserForm beenden: `Unload Name|Me`
 - "Me" referenziert das Objekt selbst in einem Objekt-Modul
-

Übung 6: Userform

Formular erstellen und aufrufen

- Userform erstellen, anzeigen und entladen:
 - Userform erstellen mit
 - zwei Buttons ("Ok" und "Abbrechen"),
 - einem Textfeld und einem Label
 - dem Formular einen Titel geben
 - Ereignis für
 - "Ok": Text des Textfeldes im Label (.Caption) darstellen
 - "Abbrechen": Formular entladen (Unload Me)
 - Funktion zum Aufruf des Formulars in normalem Modul (.show)
-

Teil 4: VBA

Variablen und Datentypen

- Explizite Deklaration:
Dim name as Datentyp
 - Variable wird mit einem Nullwert initialisiert
 - Datentypen und ihre Nullwerte:
 - **Integer**: Ganze Zahlen - 0
 - **Double**: Gleitkommazahlen - 0
 - **Boolean**: Wahrheitswerte (true und false) - False
 - **String**: Zeichenketten - "" (zwei Anführungszeichen)
 - **Variant**: undefinierter Datentyp - Empty
 - **Object**: undefiniertes Objekt - Nothing
 - Datentyp definiert die Art des Wertes einer Variable
 - Explizite Definition nicht zwingend, aber sinnvoll - vor allem bei Variablen für Objekte
 - Weitere Typen: Byte, Long, Single, Currency, Date
-

Teil 4: VBA

Datenfelder: Array

- Variable mit mehreren Werten: Datenfeld oder Array.
 - Elemente werden über ihren Index abgerufen. Der Index steht in runden Klammern:
`meinArray(2)`
 - Ein Array kann auch mehrere Dimensionen haben. Die Indizes der Dimensionen werden durch Kommas getrennt:
`meinArray(3, 4)`
 - Arrays begegnen als Rückgabewert von Funktionen oder Eigenschaften (Zuweisung an Variant-Variable)
 - Die Funktion "Array(ele1, ele2...)" erstellt einen Array
 - Bei Zellbereichen ist Range.Value ein Array (zeile, spalte)
 - Funktion zur Bestimmung der Größe eines Arrays:
 - Obergrenze: `UBound(meinArray [, Dimension])`
 - Untergrenze: `LBound(meinArray [, Dimension])`
-

Teil 4: VBA

Arrays definieren

- Array-Variablen müssen explizit mit "Dim" erstellt werden:

```
Dim meinArray(5)
```

- In Klammern gibt man die Grösse durch den kleinsten und grössten Index an; der kleinste ist optional (dann 0):

```
Dim meinArray([1 to] 10) [as Datentyp]
```

- Bei mehrdimensionalen Arrays: Grösse jeder Dimension durch Komma getrennt:

```
Dim meinArray(5, 8)
```

- Hinweis: der Array von Range.Value beginnt mit 1.

- Es ist möglich Arrays unbestimmter Grösse zu definieren (leere Klammern); nachträglich mit "ReDim" Grösse festlegen.

Teil 4: VBA

Kontrollfluss: Schleifen "For..Next"

- Die For..Next-Anweisung führt einen Anweisungsblock mit einer bestimmten Anzahl Wiederholungen aus.
- Bei jeder Iteration wird eine Variable um eins hochgezählt

```
For ii = 1 To 10
```

```
...
```

```
Next [ii]
```

- Schrittgrösse festlegen mit "Step", darf auch negativ sein

```
For Zähler = Anfang To Ende Step Schritt
```

- Schleife vorzeitig verlassen mit **Exit For**

- Ideal für Arrays (durch einen Array iterieren):

```
For ii = LBound(mArr, 1) To UBound(mArr, 1)
```

Übung 7: For-Next und Value-Array

Funktion Verkettewenn – schwierig

- Aufgabe:
 - Text in einer Spalte abhängig von einer Bedingung in der Zeile verketteten. Analog zu Excels "Summewenn()".
 - Funktion mit vier Argumenten
 - Zwei Range-Argumente für Text- und Kriterienspalte
 - Zwei Text-Argumente für Kriterium und Trennzeichen
 - Hinweis
 - Spalten mit .Value in Array(zeile, spalte) speichern
 - Anzahl Zeilen mit Funktion UBound(SpaltenArray, 1)
 - Mit For-Next die Kriterien-Spalte mit Kriterium vergleichen (mit If)
 - Bei Übereinstimmung den entsprechende Wert der Text-Spalte in Textvariable speichern. For-Zähler entspricht der Zeile.
 - Achtung: bei der ersten Übereinstimmung noch kein Trennzeichen
 - Verwendung: Arbeitsblatffunktion
-

Teil 4: VBA

Auflistung: Collection

- Collection - eine geordnete Folge verschiedener Elemente mit Index oder Schlüssel.
 - Collection begegnen oft als Auflistungen gleicher Excel-Objekte
 - Beispiel: "Workbooks", "Worksheets" oder "Cells"
 - Methoden:
 - **.Item**(index|schlüssel): Element abrufen
 - **.Count**: Anzahl der Elemente in Auflistung
 - **.Add**: Neues Element zufügen
 - Element abrufen: ".Item" kann weggelassen werden
 - [Set] ObjektName = EineKollektion.Item(2)
 - "Set" nur nötig, wenn das Element ein Objekt ist (meist der Fall)
 - Eine Collection kann auch selbst erstellt werden (selten).
-

Teil 4: VBA

Kontrollfluss: Schleifen – For Each

- Die For-Each-Anweisung ist eine spezielle Schleife für Auflistungen (Collections)
 - Für jedes Element einer Auflistung wird der Anweisungsblock einmal ausgeführt.
 - Bei jeder Iteration speichert die ForEach-Anweisung das aktuelle Element der Auflistung in einer Objekt-Variable

```
For Each objVar in ObjectAuflistung
    ...
Next [objVar]
```
 - Schleife vorzeitig verlassen mit **Exit For**
 - Kann auch für einen Array verwendet werden.
-

Teil 4: VBA

Fehlerbehandlung und Sprungmarke

- **GoTo** und Sprungmarke: Die Anweisung GoTo setzt Ausführung an einer durch eine Sprungmarke bestimmten Zeile fort.
 - Eine Sprungmarke ist eine Name (Bezeichner) gefolgt von einem Doppelpunkt

```
GoTo zeilenmarke
...
zeilenmarke:
...
```
 - Fehlerbehandlung:
 - Bei Fehlern Ausführung fortsetzen: **On Error Resume Next**
 - Mit `If Err <> 0 then` kann geprüft werden, ob ein Fehler aufgetreten ist.
 - Bei Fehler zu einer Fehlerbehandlung springen (Sprungmarke):

```
On Error Goto fehlerbehandlung
...
Exit Sub|Function
fehlerbehandlung:
...
```

 - Rückkehr: **Resume** oder **Resume Next**|sprungmarke
 - Informationen zum Fehler im Err-Objekt: `.Description` oder `.Number`
 - Fehler selbst erzeugen: `Err.Raise`
 - Text in den Direktbereich schreiben: `Debug.Print text`
-

Übung 8: ForEach und Fehler Tabelle aufteilen – schwierig

- Aufgabe:
 - Eine zweiseitige Tabelle (Auswahl) soll nach dem Wert der ersten Spalte auf einzelne Tabellenblätter kopiert werden.
 - Die erste Spalte enthält den Zieltabellennamen. Neue Tabellen müssen zugefügt werden.
 - Hinweise:
 - Selection in Variable speichern
 - Durch Zeilen ".Rows" mit ForEach iterieren.
 - Die Iterations-Variable ist ein Range für die ganze Zeile. Mit Cells auf die erste Spalte zugreifen .Cells(1) oder .Cells(1,1)
 - Zieltabelle über Tablennamen aufrufen. Fehler abfangen falls entsprechende Tabelle nicht existiert und diese zufügen.
 - Worksheets(Tabellenname) und Worksheets.Add
 - In Zieltabelle mit For..Next erste leere Zeile suchen
 - For..Next damit man die Zeilennummer hat
 - Mit Zeilennummer und .Cells den Wert der zweiten Spalte der Zeilen-Variable in Zieltabelle schreiben
-

Ausblick: Weitere Möglichkeiten mit VBA

- Zugriff auf Dateisystem:
 - Lesen und Schreiben von Textdateien
 - Löschen und verschieben von Dateien und Ordnern
 - Zugriff auf andere Office-Anwendungen
 - Jeder Teil von Office ein eigenes VBA
 - Diese können eingebunden werden
 - Zugriff auf andere Programme und Befehlsbibliotheken
 - Datenaustausch mit einer Datenbank
 - Steuern eines Mediaplayers
 - Verwenden von R oder SPSS
 - Zugriff auf Webseiten
 - Entwerfen von eigenen Objekten (Klassen)
 - Fortgeschrittenes Programmieren
-