

# Grundkurs für Excel – VBA – Part I

Nico Ludwig

# Themen

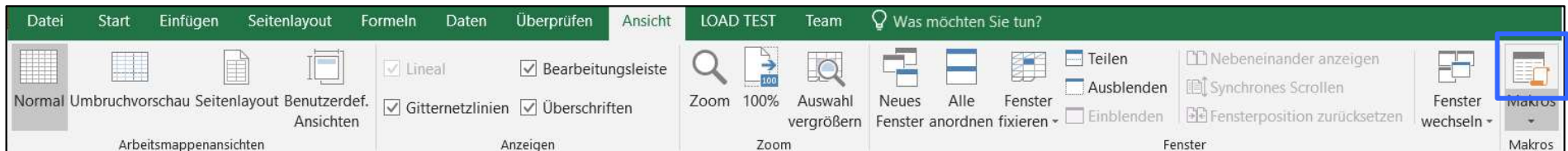
- Die VBA Entwicklungsumgebung
- Eingabe- und Ausgabedialoge
- Ausdrücke, Anweisungen und Blöcke
- Variablen
- Kommentare
- Mathematische und Logische Operatoren
- If-Anweisungen
- Select-Anweisungen
- Einfacher Zellzugriff
- Der Debugger

# Einführung

- Wir besprechen jetzt die Programmierung in Excel. Aber warum ist Excel-Programmierung eigentlich nötig?
- (1) Excel hat zwar sehr mächtige Funktionen, aber auch seine Grenzen.
  - Besonders komplexe Daten, oder Daten aus speziellen Quellen erfordern mehr "Zuarbeit".
  - Das kann man mit Programmierung eigener Excel-Makros fast immer lösen.
- (2) Wir müssen in vorhandenen Tabellen oft immer die gleichen Aktionen von Hand wiederholen.
  - Das ist erstmal nervig, aber auch fehleranfällig.
  - Um uns von solchen Routineaufgaben zu befreien, können wir Excel selbst mit Makros automatisieren.
- (3) Tabellen können so kompliziert werden, dass ein Benutzer eine Anleitung braucht, um die richtigen Daten an den richtigen Stellen einzutragen.
  - Statt einer Anleitung können wir den Benutzer mit Hilfe von Makros und Formularen durch die Dateneingabe interaktiv leiten.
- Excel-Makros erlauben uns Excel um neue Fähigkeiten zu erweitern.

# Ein VBA-Makro in Excel erstellen

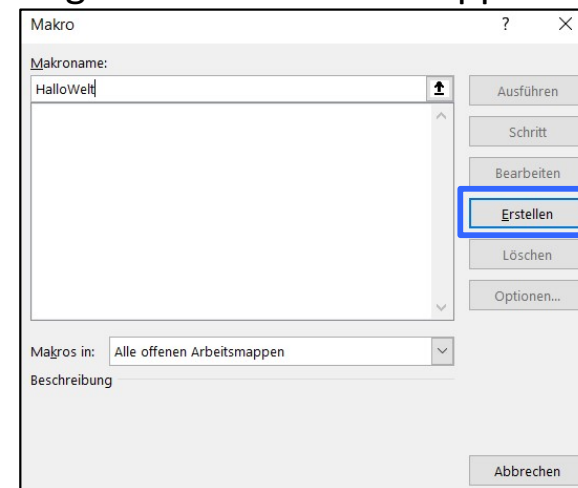
- Um ein Makro in Excel selbst zu schreiben, klicken wir die Schaltfläche "Makros" im "Ansicht"-Ribbon.



- Dann erscheint ein Dialogfenster, dass alle Makros der geöffneten Arbeitsmappen auflistet:

## Gut zu wissen:

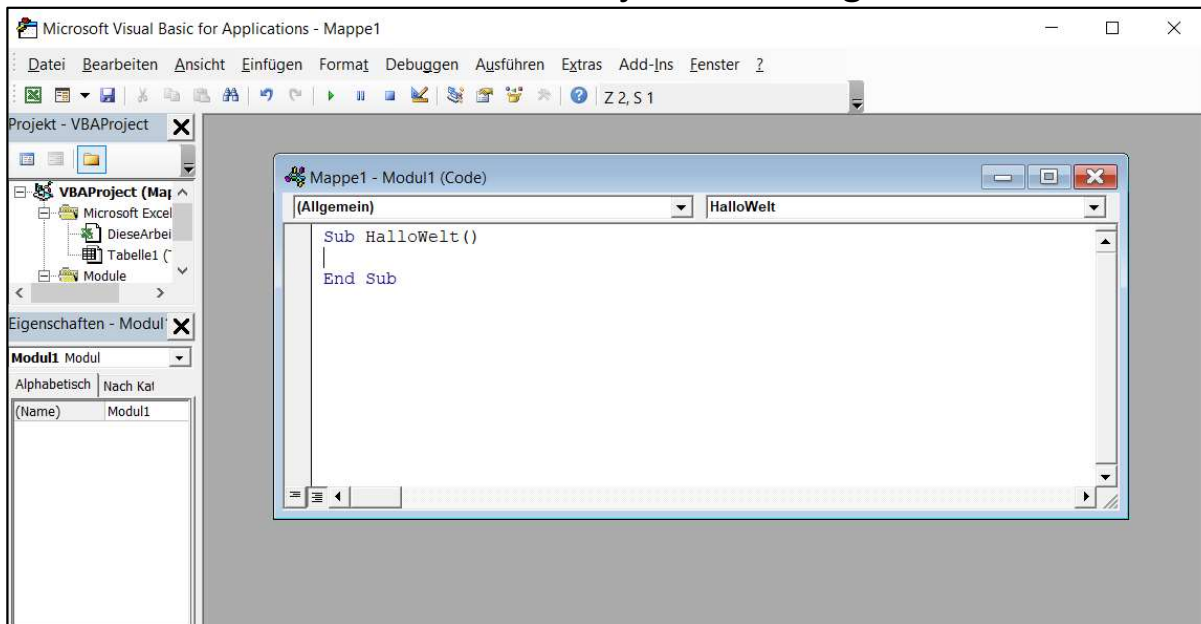
Namen für Makros (bzw. Prozeduren) müssen eindeutig sein, weniger als 255 Zeichen haben, sie müssen zusammen geschrieben werden und dürfen keine Sonderzeichen außer '\_' haben. Außerdem dürfen sie nicht mit einer Ziffer beginnen.



- Es wurde noch kein Makro erstellt, aber wir erstellen jetzt ein neues mit dem Namen "HalloWelt".

# Integrierte Entwicklungsumgebung

- Zusätzlich zu Excel öffnet sich jetzt das Programm "Microsoft Visual Basic for Applications" (VBA)



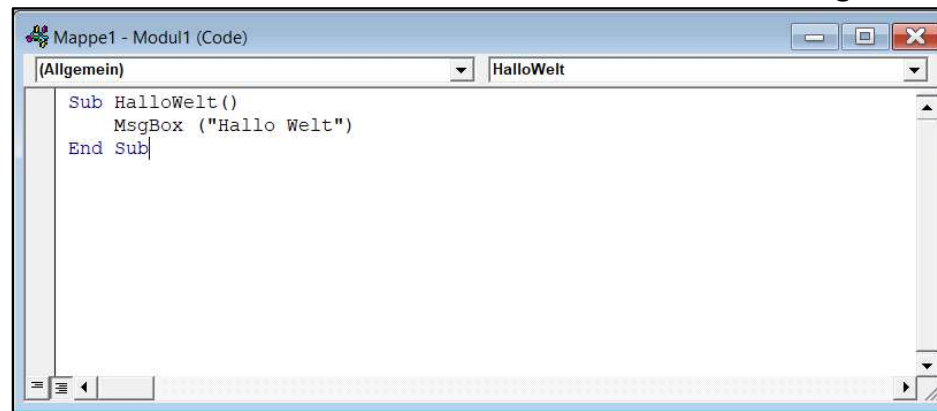
## Gut zu wissen:

Der gängige englische Begriff für "Integrierte Entwicklungsumgebung" lautet "Integrated Development Environment", abgekürzt "IDE".

- Mit diesem Programm können wir ganz bequem Makros programmieren, laufen lassen, testen und ggf. Fehler suchen.
- Ein Programm, das alle Funktionen rund um die Programmierung zusammenfasst heißt auch Integrierte Entwicklungsumgebung.
- Das Fenster zur Eingabe der VBA Makros heißt einfach Editor.

# Ein Makro starten

- Das soeben erstellte Makro, genauer gesagt die Prozedur *HalloWelt()* macht noch gar nichts!
  - Die Prozedur ist leer, wir müssen hier erst noch was programmieren, was dann ausgeführt werden soll.
  - In einem ersten Schritt schreiben wir zwischen die Zeilen, die mit **Sub** und **End Sub** beginnen den Text *MsgBox("Hallo Welt")*:

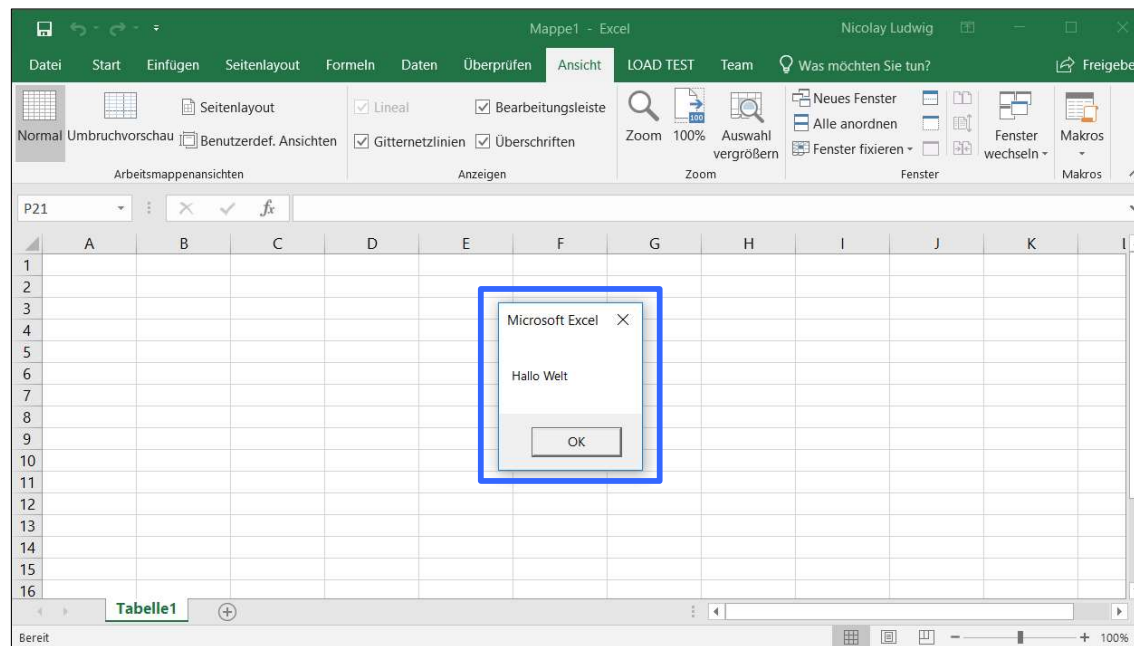


- Wenn wir die Schaltfläche "Sub/UserForm ausführen" klicken, wird das ausgewählte Makro/Prozedur ausgeführt.
  - In unserem Falle ist die Prozedur *HalloWelt()* ausgewählt:



# Das Makro bei der Arbeit

- Die Ausführung von *HalloWelt()* führt dann zur Anzeige eines Dialogfensters mit dem Text "Hallo Welt":



- Das Dialogfensters muss bestätigt werden (z.B. mit Klick auf "OK" ), erst dann schließt es sich.

# Die Prozedur *HalloWelt()*

- Schauen wir uns also das Programm selbst an, es wird repräsentiert durch die Prozedur *HalloWelt()*:

```
Sub HalloWelt()  
    MsgBox ("Hallo Welt")  
End Sub
```

## Gut zu wissen:

Der VBA-Programmtext wird oft nur "Code" oder "Listing" genannt.

- Die Textzeilen zwischen *Sub HalloWelt()* und *End Sub* geben an, was die Prozedur tun soll.
  - Der Name der Prozedur entspricht der Bezeichnung hinter dem ersten *Sub* ohne Klammern, also eigentlich nur "*HalloWelt*".
- VBA's Editor zeigt einige Wörter des Programmtexts in dunkelblauer Farbe, z.B. *Sub* und *End*.
  - Diese Wörter sind sogenannte Schlüsselwörter für VBA, d.h. sie haben eine von VBA vorgegebene Bedeutung.
  - VBA erlaubt Schlüsselwörter groß oder klein zu schreiben. Der Editor ändert aber automatisch auf Großschreibung.
- In *HalloWelt()* wird eine sogenannte Funktion, nämlich *MsgBox()* aufgerufen
  - MsgBox()* wird ein Text, "Hallo Welt", in Klammern übergeben, der dann in einem Dialogfenster (engl. "message box") angezeigt wird.
  - Das Dialogfenster (kurz "der Dialog") muss "weggeklickt" werden, bevor man Excel weiter bedienen kann. Man sagt, der Dialog ist modal.
  - Wenn ein Programm einen Hinweis anzeigt, der den Benutzer informiert und eine Eingabe verlangt, z.B. "OK klicken", nennt man den Hinweis auch Prompt (von engl. "Aufforderung").



# Compile-Zeit-Fehler

- Programmierung ist eine komplizierte Sache.
  - Bei der Art und Weise, wie man Code hinschreiben muss, so dass es einen Sinn ergibt, der sogenannten Syntax, kann man Fehler machen:

```
Sub HalloWelt()  
    MsgBox ("Hallo Welt")  
Ende Sub
```

' Das ist falsch, es muss "End Sub" heißen

- Die VBA IDE macht uns auf diesen Fehler aufmerksam und stoppt das weitere Bearbeiten des Codes:



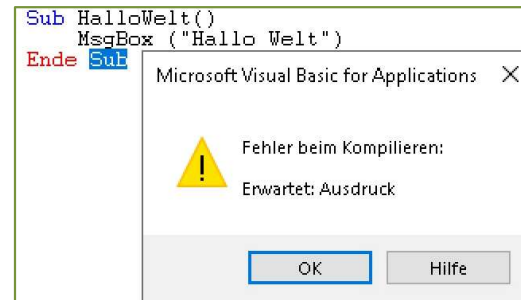
## Gut zu wissen:

Unter der Syntax (gr. Zusammensetzung) eines VBA-Konstrukts versteht man seine Schreibweise, unter seiner Semantik (gr. Bedeutungslehre), das was das Konstrukt bewirkt.

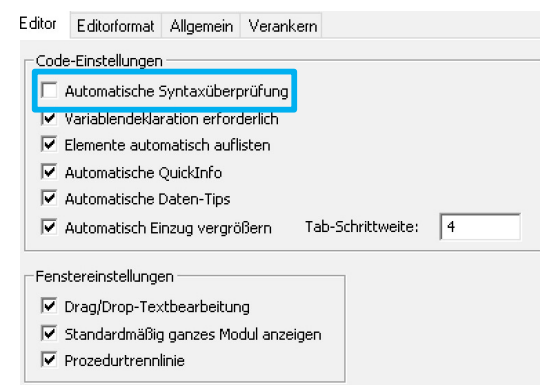
- VBA nennt das Problem "Fehler beim Kompilieren". Die Prüfung und Übersetzung von VBA-Code in "Maschinencode" wird als Kompilierung bezeichnet.
- Man spricht auch von einem Kompile-Zeit-Fehler (engl. to compile, etwas zusammenstellen).
- Allgemein spricht man in der Programmierung auch von einem Syntaxfehler.
- Aber ganz einfach ausgedrückt handelt es sich hierbei um einen Schreibfehler, also wir haben falschen VBA-Code geschrieben.
- Wir müssen den Schreibfehler korrigieren, um in der IDE weiter programmieren zu können.

# Wenn die Syntax-Prüfung stört

- Während der Programmierung kann die "ständige" Prüfung des Codes durch die IDE schnell stören.
  - Das Problem beim VBA-Editor ist hierbei, dass wir laufend Dialoge wegklicken müssen, um weiterzuarbeiten:



- Das Problem können wir beheben, in dem wir die automatische Syntaxprüfung abschalten.
  - Das machen wir in den Optionen (Menu Extras/Optionen) der IDE:



# In Zellen schreiben

- Statt der Anzeige eines Dialogs, können wir auch direkt in eine Zelle schreiben.

```
Sub HalloWelt()  
    MsgBox ("Hallo Welt")  
End Sub
```



```
Sub HalloWelt()  
    Range("A1") = "Hallo Welt!"  
End Sub
```

- Das Ausführen der neuen Variante von *HalloWelt()* schreibt "Hallo Welt" direkt in A1:

	A	B	C	D	E
1	Hallo Welt				
2					
3					

- Der große Unterschied: wir müssen keinen Dialog wegklicken.

# Die Funktion *Range()*

- Die veränderte Prozedur macht es offensichtlich ganz anders:

```
Sub HalloWelt()  
    Range("A1") = "Hallo Welt!"  
End Sub
```

- Auch hier verwenden wir eine Funktion, und zwar *Range()*.
  - Wir übergeben an *Range()* die Koordinaten der Zelle A1 als Argument in Textform, also in doppelten Anführungszeichen.
  - VBA-Funktionen wie *MsgBox()* und *Range()* haben nichts mit Arbeitsmappen-Funktionen, wie *SUMME()* zu tun!
  - *Range()* liefert einen Ergebnis, nämlich einen Bereich, hier einen Bereich mit genau einer Zelle, A1.
  - Dieser Zelle weisen wir mit dem =-Operator den Wert "Hallo Welt" zu.
  - D.h. der Wert "Hallo Welt" wird durch =-Zuweisung direkt in die Zelle A1 geschrieben.

# Kommentare

- VBA Code kann kompliziert werden. Daher dürfen wir direkt in den Code erklärenden Text schreiben:

```
Sub HalloWelt()  
    ' Hier wird eine Nachricht als Dialog ausgegeben:  
    MsgBox ("Hallo Welt")  
End Sub
```

- Syntax: Diese sogenannten Kommentare müssen mit einem einfachen Hochkomma eingeleitet werden.
  - Der Text, der hinter dem ' steht wird von VBA übersprungen und ist nur für den Programmierer als Beschreibung gedacht.
  - Der VBA Editor zeigt Kommentare in grüner Farbe an.
  - Alternativ zu ' kann auch das **Rem**-Schlüsselwort verwendet werden (Rem für engl. remark):

```
Sub HalloWelt()  
    Rem Hier wird eine Nachricht als Dialog ausgegeben:  
    MsgBox ("Hallo Welt")  
End Sub
```

- Vor allem für Beginner in der Programmierung sind viele Kommentare sinnvoll.

# Kurz-Syntax für den Zellzugriff

- Excel VBA kennt eine Kurz-Syntax mit eckigen Klammern für den Zellzugriff:

```
Sub HalloWelt()  
    Range("A1") = "Hallo Welt!"  
End Sub
```



```
Sub HalloWelt()  
    ' Kurzschreibweise mit []  
    [A1] = "Hallo Welt!"  
End Sub
```

- Wir können mit *Range()* auch alle Zellen eines Bereiches mit dem selben Wert füllen:

```
Sub HalloWelt()  
    Range("A1:C2") = "Hallo Welt!"  
End Sub
```

	A	B	C
1	Hallo Welt	Hallo Welt	Hallo Welt
2	Hallo Welt	Hallo Welt	Hallo Welt

- Auch dafür gibt es eine Kurz-Syntax:

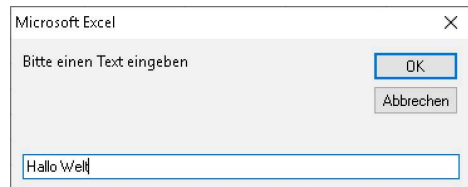
```
Sub HalloWelt()  
    [A1:C2] = "Hallo Welt!"  
End Sub
```

# Der Eingabedialog

- Als nächstes wollen wir einen Eingabedialog verwenden.
  - Wir dem Ergebnis von *Range()* nicht den festen Wert "Hallo Welt" zu, sondern fragen den Benutzer:

```
Sub HalloWelt()  
    [A1] = InputBox("Bitte einen Text eingeben")  
End Sub
```

- Bei der Ausführung von *HalloWelt()* fragt uns nun ein Dialog nach dem gewünschten Inhalt:



	A	B	C
1	Hallo Welt		
2			

- Durch den Eingabedialog ist *HalloWelt()* deutlich mächtiger, weil der Text in A1 durch den Benutzer frei wählbar ist.
- Die Funktion *InputBox()* öffnet einen Dialog, der den Benutzer mit dem Prompt "Bitte einen Text eingeben" zur Eingabe auffordert.
- Wenn ein Programm eine Benutzereingabe wiederholt nennt man das Echo.

# Einfache Operatoren

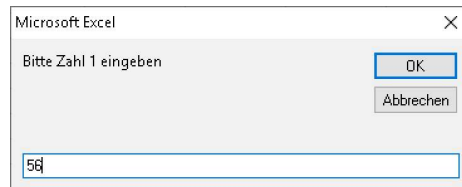
- Im nächsten Schritt wollen wir zwei Zahlen mit einem Makro addieren.
  - (Wir behalten zunächst den Namen *HalloWelt* bei.)

```
Sub HalloWelt()  
    [A1] = InputBox("Bitte Zahl 1 eingeben")  
    [B1] = InputBox("Bitte Zahl 2 eingeben")  
    [C1] = [A1] + [B1] ' Addition mit +  
End Sub
```

## Wichtige Operatoren in VBA:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
&	Textverkettung
Mod	Divisionsrest

- Das Makro fragt hierzu zwei Zahlen vom Benutzer mit zwei Eingabedialogen ab und speichert sie in den Zellen A1 und B1.
- Der Inhalt der Zellen A1 und B1 wird dann wieder abgerufen, addiert und das Ergebnis in der Zelle C1 gespeichert.
- Die VBA-Syntax bietet den aus der Mathematik bekannten + - Operator an, um eine Addition von Zahlen zu bewirken.



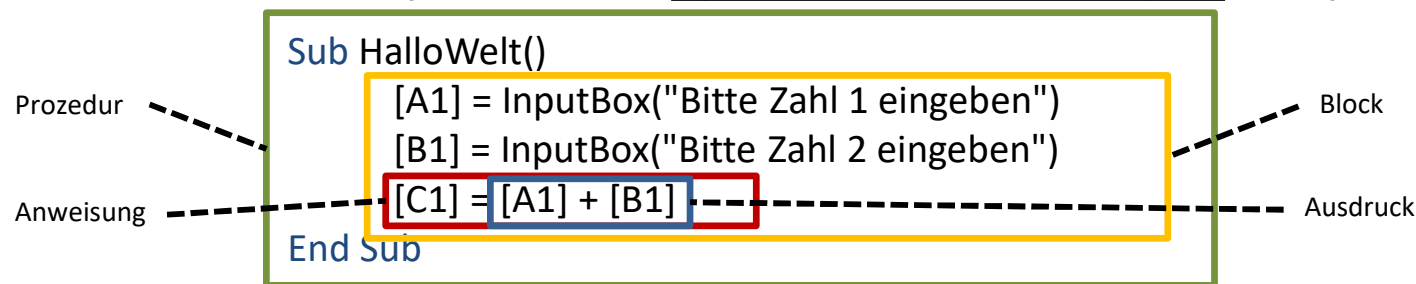
	A	B	C
1	56	89	145
2			

- Wir müssen manchmal auch Klammern in Berechnungsausdrücken verwenden, denn es gilt "Punkt vor Strich"!



# Anweisungen, Blöcke und Ausdrücke

- Wir müssen noch ein paar neue syntaktische Konzepte besprechen.



- HalloWelt()* besteht jetzt aus drei Anweisungen.
  - VBA erlaubt beliebig viele Anweisungen in einer Prozedur.
  - Im Englischen wird eine Anweisung als Statement bezeichnet.
  - Eine Serie von Anweisungen, also hier von drei Anweisungen, wird als Block bezeichnet.
  - Eine Anweisung kann mit einem angehängten ' ' auf mehrere Zeilen verteilt werden.
- Eine Anweisung besteht aus einem oder beliebig vielen Ausdrücken.
  - Im Deutschen würde man einen Ausdruck auch als Term bezeichnen, im Englischen heißt er Expression.

# Arbeiten mit Variablen – Teil I

- Im letzten Beispiel wurden die erste und zweite Zahl in Zellen gespeichert.
  - Das ist aber nicht sinnvoll:
    - Die Verwendung von Zellen als Speicher wäre für den Benutzer als unnötiges Echo sichtbar.
    - Die Verwendung von Zellen als Speicher könnte wichtige Daten in den Zellen überschreiben!
    - Wir sind ja nur am "Endwert" der Zelle C1 interessiert.
- Mit Variablen erlaubt uns VBA Werte nur im Speicher und nicht in Zellen abzuspeichern:

```
Sub HalloWelt()  
    Dim wert1, wert2 As Integer  
    wert1 = InputBox("Bitte Zahl 1 eingeben")  
    wert2 = InputBox("Bitte Zahl 2 eingeben")  
    [C1] = wert1 + wert2  
End Sub
```

- Hier werden nun die Zwischenwerte als Ergebnisse der Eingabedialoge in den Variablen *wert1* und *wert2* gespeichert.<sup>18</sup>

# Arbeiten mit Variablen – Teil II

```
Sub HalloWelt()  
    Dim wertEins, wertZwei As Integer  
    ...  
End Sub
```

- Mit der **Dim**-Anweisung werden Variablen vor ihrer Anwendung bekanntgegeben.
  - Man sagt dazu auch, dass Variablen definiert werden. (In Basic spricht man manchmal auch vom "Dimensionieren".)
  - In den folgenden Beispielen/Übungen werden wir Variablen immer vor ihrer Anwendung definieren.
  - **Dim**, **As** und **Integer** sind VBA Schlüsselwörter.
- Die Variablendefinitionen, die wir verwenden, beschreiben Namen und Typen von Variablen.
  - Hinter dem **Dim**-Schlüsselwort schreiben wir ein oder mehrere kommagetrennte Variablennamen.
  - Hinter das **As**-Schlüsselwort schreiben wir den Typen, den alle Variablen haben sollen.
  - Also in diesen Fall haben die Variablen *wertEins* und *wertZwei* den Typ **Integer**.
  - **Integer**-Variablen können nur ganze Zahlen speichern (engl. integer für Ganzzahl).<sup>19</sup>

# Arbeiten mit Variablen – Teil III

- Der Variablentyp gibt an, welche Werte in einer Variablen gespeichert werden dürfen.
  - Wie eben besprochen, schreiben wir den Variablentyp hinter das **As**-Schlüsselwort der **Dim**-Anweisung.
  - Hier eine Auswahl möglicher Typen:

Typ	Beschreibung
Integer	kleine Ganzzahl (-32768 bis 32767)
Long	große Ganzzahl (-2147483648 bis 2147483648)
Double	Dezimalzahl (3.14, anstatt dem Komma schreiben wir einen <u>Punkt</u> )
String	Text ("Hallo!", Text muss in <u>doppelte Anführungszeichen</u> gesetzt werden)
Boolean	Wahr/Falsch (nur die Werte <b>True</b> und <b>False</b> sind zulässig)
Date	Datum (#1/31/2019# (es gibt auch alternative Schreibungen in VBA))
Object	Eingebettete Objekte

# Bezeichner – Teil I

- In den bisherigen Beispielen haben wir schon einige Elemente aus VBA kennengelernt:
  - Prozeduren, wie *HalloWelt()*.
  - Funktionen, wie *MsgBox()*, *Range()* und *InputBox()*.
  - Typen, wie *Integer* und *String*.
  - Variablen, wie *wertEins* und *wertZwei*.
- Die Namen dieser Elemente, die Bezeichner, folgen einem Schema in der Groß- und Kleinschreibung:
  - Im Englischen wird die Groß- und Kleinschreibung als "letter case" oder auch "casing" bezeichnet.
  - Bezeichner von Prozeduren, Funktionen und Typen beginnen mit einem Großbuchstaben, jedes weitere Wort wird mit einem Großbuchstaben begonnen. – Diese Schreibweise wird im Englischen manchmal PascalCase genannt.
    - Das ist nämlich eine übliche Schreibweise von Bezeichnern in der Programmiersprache Pascal.
  - Bezeichner von Variablen beginnen mit einem Kleinbuchstaben und jedes weitere Wort wird wieder mit einem Großbuchstaben begonnen. – Diese Schreibweise wird im Englischen manchmal camelCase genannt.
- Die Groß- und Kleinschreibung ist in VBA irrelevant, also *wertEins* und *Werteins* sind gleiche VBA-Bezeichner.
- Dennoch folgen wir den VBA-Bezeichner Schemata, um den Überblick zu behalten.

# Bezeichner – Teil II

- Um ein Programm für Menschen lesbar zu machen, ist der Name einer Variablen noch wichtiger als ihr Typ!
  - Man sagt, dass Variablennamen "sprechend" sein müssen.
  - Für unsere kleine Prozedur ist das nicht relevant, aber bei großen Prozeduren sind gute Variablennamen ein Muss!
- Es gibt in VBA allerdings auch Vorschriften für Variablennamen:
  - Sie müssen eindeutig sein.
  - Sie müssen weniger als 255 Zeichen haben.
  - Sie müssen zusammen geschrieben werden.
  - Sie dürfen keine Sonderzeichen außer '\_' haben.
  - Außerdem dürfen sie nicht mit einer Ziffer beginnen.
  - => Die Vorschriften entsprechen denen für Prozedurnamen, bzw. denen für alle VBA-Bezeichner!

# Verzweigungen – Einführung – Teil I

- Wir haben uns bisher Code angeschaut, der "von oben nach unten" ausgeführt werden.
  - Man sagt, solcher Code ist sequenziell.

```
Sub Prozedur1()  
    Anweisung1  
    Anweisung2  
    ...  
End Sub
```

- Aber die meisten Programme sind komplexer, z.B. wenn Entscheidungen getroffen werden müssen.
  - Dann wird Code nicht sequentiell, sondern in Abhängigkeit von Entscheidungen ausgeführt.
  - Beispiel: Nur wenn der Wert in A1 größer ist als der Wert in B1 wird deren Differenz in B2 ausgegeben:

	A	B
1	15	2
2	Ergebnis:	13

# Verzweigungen – Einführung – Teil II

- Also: Nur wenn der Wert in A1 größer ist als der Wert in B1 wird deren Differenz in B2 ausgegeben.
  - Die zugehörige Prozedur sieht so aus:

```
Sub EinfacheVerzweigung()  
    If [A1] > [B1] Then  
        [B2] = [A1] - [B1]  
    End If  
End Sub
```

	A	B
1	15	2
2	Ergebnis:	13

- Die Schlüsselwörter **If Then** und **End If** sind neu für uns.
  - Wenn die Bedingung, die zwischen **If** und **Then** wahr ist, wird der Code zwischen **Then** und **End If** ausgeführt.
  - D.h. der Programmablauf verzweigt nur dann in den **If**-Code, wenn die zugehörige **If**-Bedingung wahr ist.
  - D.h. auch: wenn die Bedingung falsch ist, wird der **If**-Code übersprungen. In *EinfacheVerzweigung()* passiert dann einfach gar nichts!
- Hier sehen wir eine einfache Verzweigung und kein sequentielles Programm mehr.
  - Im Gegensatz zu den einfachen Verzweigungen gibt es in Excel auch verschachtelte Verzweigungen.
  - Wir sehen uns auch noch verschachtelte Verzweigungen an, aber zunächst erweitern wir noch unser Beispiel.



# Verzweigungen – Einführung – Teil III

- Nun wandeln wir die Problemstellung etwas ab:
  - Wenn der Wert in A1 größer ist als der Wert in B1, wird deren Differenz in B2 ausgegeben.
  - Wenn der Wert in A1 kleiner ist als der Wert in B1, wird deren Summe in B2 ausgegeben.
- Wir können *EinfacheVerzweigung()* mit unseren bisherigen Kenntnissen anpassen:

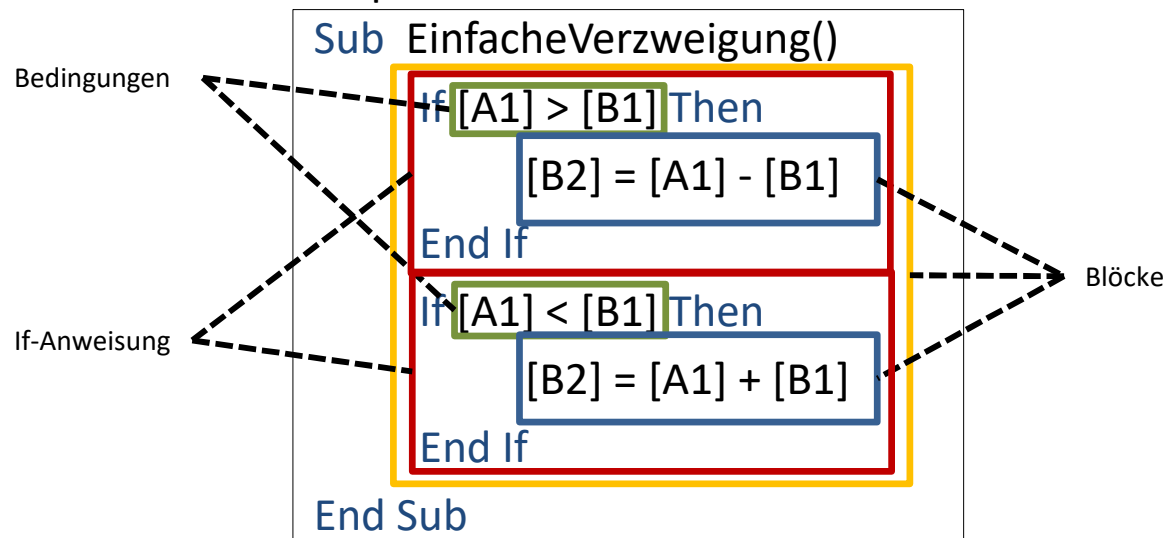
```
Sub EinfacheVerzweigung()  
  If [A1] > [B1] Then  
    [B2] = [A1] - [B1]  
  End If  
  If [A1] < [B1] Then  
    [B2] = [A1] + [B1]  
  End If  
End Sub
```

	A	B
1	2	15
2	Ergebnis:	17

- Die Lösung besteht also darin, einfach zwei **If**-Programmteile einzusetzen, die zwei Bedingungen "behandeln".
- Wichtig ist dabei, dass beide **If**-Programmteile hintereinander, d.h. auf gleicher Ebene geschrieben werden.

# Verzweigungen – Struktur und Begriffe

- Es ist wieder Zeit ein Paar Konzepte zu erläutern:



- Wir sprechen von If-Anweisungen, eine If-Anweisung spezifiziert eine Bedingungen.
- Der Code zwischen Then und End If kann ein oder mehrere Anweisungen enthalten, wir nennen ihn If-Block.
- D.h. sowohl die Anweisungen auf der Prozedurebene als auch auf den If-Ebenen sind Blöcke!
- Und daraus ergibt sich, dass Blöcke verschachtelt werden können!

# Verzweigungen – Einrückung

- Da wir das Konzept der Verschachtelung eingeführt haben, reden wir noch über die Schreibweise:

```
Sub EinfacheVerzweigung()  
    If [A1] > [B1] Then  
        [B2] = [A1] - [B1]  
    End If  
    If [A1] < [B1] Then  
        [B2] = [A1] + [B1]  
    End If  
End Sub
```

1. Einrückung

2. Einrückung

- Um die Lesbarkeit zu besseren werden verschachtelte Blöcke im Programmtext syntaktisch eingerückt.
- VBA erlaubt es grundsätzlich auf Blockeinrückung zu verzichten, aber darauf verzichten wir in diesem Kurs nicht!
- Im Englischen wird die Einrückung Indentation genannt.
- Es gibt Programmiersprachen wie Python, die die Blockeinrückung vorschreiben.

# Vergleichsoperatoren

- Um Bedingungen zu formulieren bietet VBA spezielle, sogenannte logische Operatoren an:
  - = und <> stehen für gleich und nicht gleich.
  - >, >=, <, <= stehen für die mathematischen ungleich-Operatoren.
  - **And** und **Or** dienen zur Verknüpfung von Bedingungen.
  - **Not** dient zum negieren von Bedingungen, also wir können damit **True** zu **False** oder **False** zu **True** machen.
  - Logische Ausdrücke müssen manchmal Klammern verwenden, denn Not hat Vorrang vor And, und And Vorrang vor Or!
- Zusätzlich gibt es noch Prüffunktionen, die uns sagen, ob eine Eigenschaft zutrifft:
  - *IsEmpty()* prüft, ob der übergebene Ausdruck, z.B. eine Zellreferenz, leer ist.
  - *IsNumeric()* prüft, ob der übergebene Ausdruck, z.B. eine Zellreferenz, eine Zahl enthält.
  - *WorksheetFunction.IsText()* prüft, ob das übergebene Argument, z.B. eine Zellreferenz, einen Text enthält.
  - All diese Funktionen sind keine Schlüsselwörter in VBA. Sie werden uns aber von der VBA Funktionsbibliothek bereitgestellt.
  - Funktionen, die ein Argument entgegennehmen und eine Eigenschaft des Arguments prüfen, werden Prädikate genannt.
    - Ein Prädikat gibt **True** oder **False** zurück, je nachdem ob die Eigenschaft zutrifft oder nicht.

## **Gut zu wissen:**

*WorksheetFunction* ist eine globale Eigenschaft, die uns die Funktionen der Arbeitsmappe zur Verfügung stellt, d.h. alle Excel-Funktionen, die wir schon kennen (z.B. SUMME()) können wir so in VBA wiederverwenden, indem wir sie nach dem Punkt-Operator aufrufen.

# Verzweigungen mit Else-Block

- Im vorherigen Beispiel würde der Fall wenn die Zahlen in A1 und B1 gleich sind gar nicht behandelt!
  - Dann passiert nichts, denn keine der Bedingungen in *EinfacheVerzweigung()* ist erfüllt und kein *If*-Block wird ausgeführt!
- Daher legen wir die Problemstellung etwas anders fest:
  - Wenn der Wert in A1 größer ist als der Wert in B1, wird deren Differenz in B2 ausgegeben,
  - sonst wird deren Summe in B2 ausgegeben:

```
Sub EinfacheVerzweigung()  
    If [A1] > [B1] Then  
        [B2] = [A1] - [B1]  
    Else  
        [B2] = [A1] + [B1]  
    End If  
End Sub
```

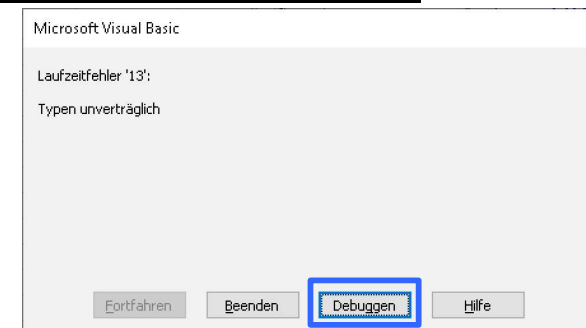
	A	B
1	15	15
2	Ergebnis:	30

- Anstatt einer zweiten *If*-Anweisung erweitern wir die erste um einen *Else*-Block ohne Bedingung.
  - Wenn also die Bedingung der *If*-Anweisung nicht zutrifft, wird der *Else*-Block aufgeführt.

# Verschachtelte Verzweigungen – Teil I

- Allerdings gibt es in *EinfacheVerzweigung()* eine Einschränkung: sie funktioniert nur mit Zahlen!
  - Wenn z.B. eine der Zellen A1 oder B1 einen Text enthält, schlägt *EinfacheVerzweigung()* mit einem Laufzeitfehler fehl:
  - Im Gegensatz zu einem Syntaxfehler tritt ein Laufzeitfehler erst auf, wenn das Programm schon läuft.

	A	B
1	Hallo	15
2	Ergebnis:	




- Ein Klick auf "Debuggen" zeigt das Problem in der Prozedur:

```
(Allgemein)  Einfac
Sub EinfacheVerzweigung()
  If [A1] > [B1] Then
    | [B2] = [A1] - [B1]
  Else
    [B2] = [A1] + [B1]
  End If
End Sub
```

- Das Problem liegt darin, dass Textwerte nicht mit dem - -Operator voneinander "abgezogen" werden können.

# Verschachtelte Verzweigungen – Teil II

- Wie lösen wir das Problem? Erstmal verlassen wir den Debugger wieder: 
- Wir lösen das "programmatisch", in dem wir die Eingaben prüfen, bevor wir damit rechnen.
- Die Prüfung von Eingabedaten ist ein Grundprinzip der defensiven Programmierung.
  - Defensiv bedeutet in diesem Fall "abwehrend".
  - Abwehrend meint hier, dass wir "unpassende" Daten nicht akzeptieren und deren Verarbeitung verweigern.
- Die Prüfung von Eingabedaten von "menschlichen" Benutzern ist besonders wichtig!
  - (1) Bei "menschlichen" Benutzern muss man mit allen Arten von Fehleingaben rechnen.
  - (2) Bei so einem Fehler muss der Benutzer eine Rückmeldung bekommen.
  - => Wenn wir (1) und (2) nicht beherzigen, sind "menschliche" Benutzer mindestens verärgert!
- Die Prüfung und Behandlung von Eingaben kann ein sehr komplizierter Programmteil sein!

# Verschachtelte Verzweigungen – Teil III

- Eine einfache Lösung ist die Verwendung einer weiteren "umgebenden" Verzweigung:
- Die äußere Verzweigung lagert die Prüfungen *IsNumeric()* und auch noch *Not IsEmpty()* vor.

```
Sub VerschachtelteVerzweigung()  
    If IsNumeric([A1]) And IsNumeric([B1]) And Not IsEmpty([A1]) And Not IsEmpty([B1]) Then  
        ' alles "in Ordnung":  
        If [A1] > [B1] Then  
            [B2] = [A1] - [B1]  
        Else  
            [B2] = [A1] + [B1]  
        End If  
    Else ' eine Fehleingabe:  
        MsgBox ("In A1 und B2 müssen Zahlen stehen!")  
    End If  
End Sub
```

- Die Berechnung wird jetzt nur unter der Bedingung ausgeführt, dass die Eingaben "in Ordnung" sind.
- Ansonsten sagt ein Dialog denn Benutzer, was mit den Eingaben nicht stimmt und die Prozedur wird beendet.



# Der Debugger – Teil I

- Wir lernen noch ein Werkzeug der VBA IDE kennen, um unseren Code zur Laufzeit zu analysieren.
- Die VBA IDE bietet uns dafür also einen sogenannten Debugger. Mit dem Debugger können wir
  - VBA-Code zur Laufzeit anhalten und schrittweise durchlaufen
  - und den Wert von Zellen und Variablen einsehen.
- Die eigentliche Aufgabe des Debuggers ist aber die Fehlersuche in Programmen.
- Der Name "Debugger" hat seine Herkunft aus den Anfängen der Computer.

**Gut zu wissen:**

Die ersten Computer füllten ganze Räume und wurden mit Elektronenröhren betrieben. Elektronenröhren erzeugten sehr viel Wärme und lockten Insekten an. Diese Insekten konnten Störungen und Kurzschlüsse in den Schaltungen verursachen. Aus diesem Grunde wurden Mitarbeiter, sog. Debugger (= "Ent-Käferer"/Käfersammler) beauftragt, regelmäßig die Insekten in den Computern abzusammeln. Auch heute werden Programme zum "abklappern" eines Programmes als Debugger bezeichnet. Und Laufzeitfehler werden auch heute noch als "Bug" bezeichnet.

# Der Debugger – Teil II

- Wir wenden den Debugger jetzt mit diesen Eingabewerten an:

	A	B
1	5	acht
2	Ergebnis:	

- Zunächst müssen wir im Editor einen Haltepunkt (engl. breakpoint) setzen:

```
Sub VerschachtelteVerzweigung()  
    If IsNumeric([A1]) And IsNumeric([B1]) And Not IsEmpty([A1]) And Not [B1]  
        If [A1] > [B1] Then  
            [B2] = [A1] - [B1]  
        Else  
            [B2] = [A1] + [B1]  
        End If  
    Else  
        MsgBox ("In A1 und B2 müssen Zahlen stehen!")  
    End If  
End Sub
```

- Dazu klicken wir auf der Höhe der Zeile, an der wir stoppen wollen auf die linke Seitenleiste.
- Dort erscheint ein brauner Punkt und die entsprechende Zeile wird braun hinterlegt.
- Dann starten wir die Prozedur:



# Der Debugger – Teil III

- Die Ausführung von *VerschalteteVerzweigung()* stoppt im Debugger sofort am Haltepunkt:

```
Sub VerschalteteVerzweigung()  
    If IsNumeric([A1]) And IsNumeric([B1]) And Not IsEmpty([A1]) And Not [B1] Then  
        If [A1] > [B1] Then  
            [B2] = [A1] - [B1]  
        Else  
            [B2] = [A1] + [B1]  
        End If  
    Else  
        MsgBox ("In A1 und B2 müssen Zahlen stehen!")  
    End If  
End Sub
```

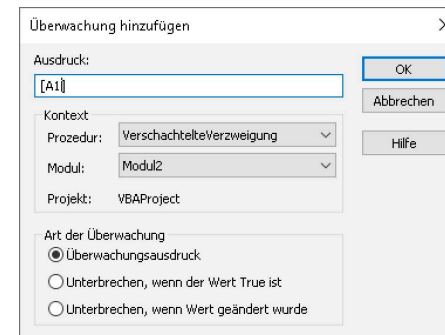
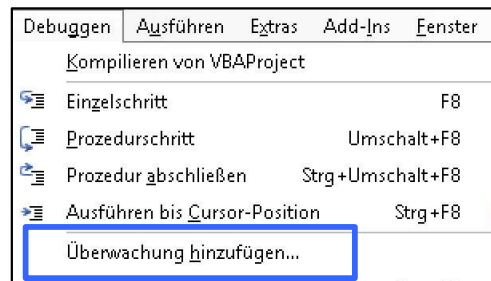
- Mit der F8-Taste können wir den Debugger den nächsten Programmschritt ausführen lassen:

```
Sub VerschalteteVerzweigung()  
    If IsNumeric([A1]) And IsNumeric([B1]) And Not IsEmpty([A1]) And Not [B1] Then  
        If [A1] > [B1] Then  
            [B2] = [A1] - [B1]  
        Else  
            [B2] = [A1] + [B1]  
        End If  
    Else  
        MsgBox ("In A1 und B2 müssen Zahlen stehen!")  
    End If  
End Sub
```

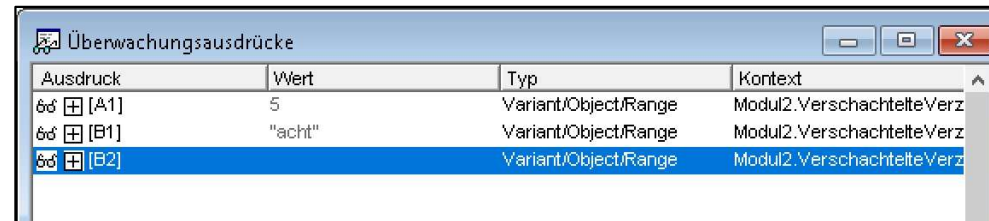
- Also, die Ausführung "landet" im **Else**-Block der Eingabeprüfung! – Aber warum?
- Um das zu verstehen, schauen wir uns die Werte in den Zellen A1, B1, also den Eingabedaten aber auch das Ergebnis in B2 an.

# Der Debugger – Teil IV

- Wir können im Debugger Zellen und Variablen einsehen, in dem wir sie überwachen.
  - Dazu erstellen wir Überwachungen (engl. watch) für die Zellen A1, B1 und B2:



- Der Debugger zeigt nun im Fenster "Überwachungsausdrücke" alle erstellen Überwachungen an:



- Wir sehen hier, warum die Eingabeprüfung im Else-Block landete: der Inhalt von B1 ist keine Zahl, das Prädikat *IsNumeric*("acht") trifft nicht zu!
- Wir können den Debugger wieder verlassen:



# Mehrfache Ifs und Elself

- Besprechen wir noch eine weitere Aufgabenstellung, wenn es viele Verzweigungen geben soll:
  - Die neue Prozedur soll einer Schulnote als Zahl von 1 bis 6 in A1 deren Textform zuordnen und in B1 ausgeben.
  - Ohne irgendwelche Eingabeprüfungen sind das 6 Verzweigungen:

	A	B
1	2	gut

- Wir verwenden hier die neue Elself-Anweisung.
  - Elself kombiniert die Else- und If-Anweisungen zu einer Anweisung.
  - Mit Elself können wir "Verzweigungsketten" ganz einfach hinschreiben.

```
Sub VieleVerzweigungen()  
  If [A1] = 1 Then  
    [B1] = "sehr gut"  
  Elself [A1] = 2 Then  
    [B1] = "gut"  
  Elself [A1] = 3 Then  
    [B1] = "befriedigend"  
  Elself [A1] = 4 Then  
    [B1] = "ausreichend"  
  Elself [A1] = 5 Then  
    [B1] = "mangelhaft"  
  Elself [A1] = 6 Then  
    [B1] = "ungenügend"  
  End If  
End Sub
```

# Fallunterscheidung – Select...Case

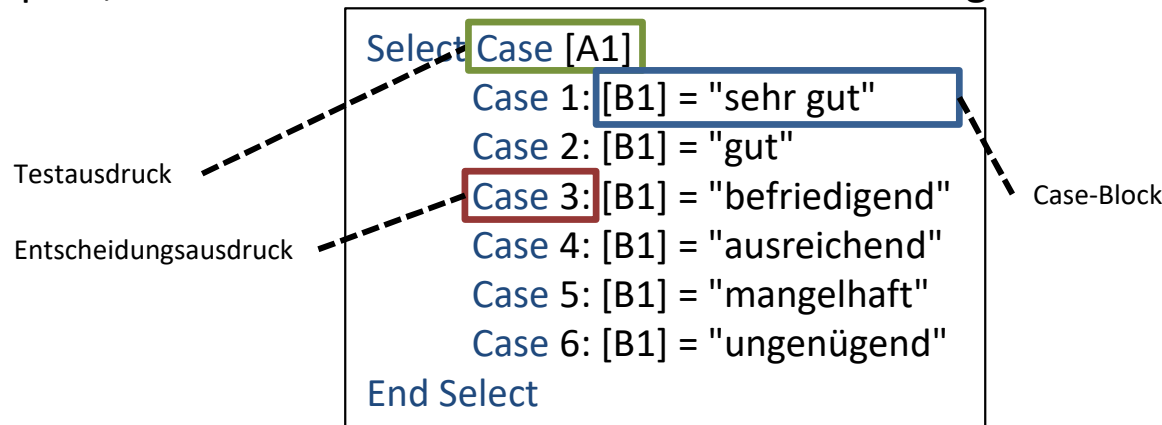
- Wenn alle Bedingungen einer "Verzweigungskette" den selben Wert prüfen, können wir das vereinfachen:

```
Sub Fallunterscheidung()  
    Select Case [A1]  
        Case 1: [B1] = "sehr gut"  
        Case 2: [B1] = "gut"  
        Case 3: [B1] = "befriedigend"  
        Case 4: [B1] = "ausreichend"  
        Case 5: [B1] = "mangelhaft"  
        Case 6: [B1] = "ungenügend"  
    End Select  
End Sub
```

- Die neue Anweisung heißt Select...Case-Anweisung, oder auch nur Select-Anweisung.
  - Diese Konstruktion wird im Gegensatz zur (Mehrfach-)Verzweigung als Fallunterscheidung bezeichnet.
- Die Select-Anweisung macht *Fallunterscheidung()* viel kompakter als *VieleVerzweigungen()*.
  - Auch Programmierfehler lassen sich damit vermeiden.

# Fallunterscheidung – Select...Case

- Weil so kompakt, ist die "Basisanatomie" der **Select**-Anweisung ziemlich einfach:



- Der Unterschied zur **If**-Anweisung ist, dass nur das Ergebnis eines Testausdrucks (Eingabe) gegen feste Werte geprüft wird.
  - D.h. die **If**-Anweisung muss für komplexere Prüfungen weiterhin bevorzugt werden.
    - Komplexe wahr/falsch-Bedingungen mit **And**, **Or** oder Prädikaten wie `IsEmpty()` kann man nicht mit der **Select**-Anweisung verwenden.
  - Nur der "passende" **Case**-Block der **Select**-Anweisung wird ausgeführt!
  - In diesem Beispiel hat jeder **Case**-Block nur eine Anweisung, aber es dürfen zwischen zwei **Cases** beliebig viele stehen.
  - Wenn ein **Case**-Block nur eine Anweisung hat, wird die Anweisung oft direkt hinter den Entscheidungsausdruck geschrieben.
  - Man kann **Select**-Anweisungen problemlos mit anderen **Select**- oder **If/Else/Elseif**-Anweisungen verschachteln.

# Fallunterscheidung – Listen, To und Case Else

- VBA bietet für die Entscheidungsausdrücke der **Select**-Anweisung noch ein paar Ergänzungen an:

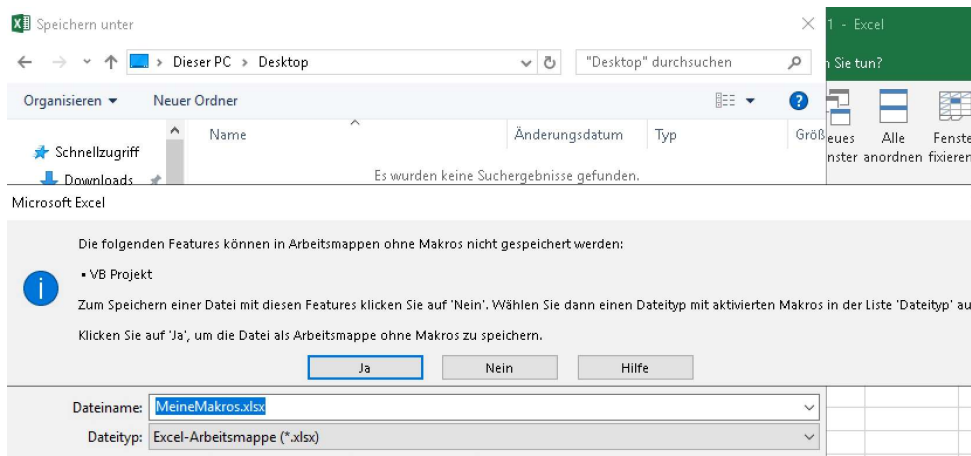
```
Select Case [A1]
    Case 1, 2: [B1] = "ordentliche Leitung"
    Case 3: [B1] = "gute Basis"
    Case 4 To 6: [B1] = "wir müssen dran arbeiten"
    Case Else: MsgBox ("Nur Noten von 1 - 6 werden akzeptiert!")
End Select
```

- Folgt **Case** eine kommasetrennte Liste von Werten, wird der **Case**-Block ausgeführt, wenn einer davon dem Wert des Testausdrucks entspricht.
  - Folgt **Case** eine Bereichsangabe mit **To**, wird der **Case**-Block ausgeführt, wenn der Wert des Testausdrucks dazwischen oder inklusive liegt.
  - Folgt **Case** das **Else**-Schlüsselwort, wird der **Case**-Block ausgeführt, wenn kein anderer Entscheidungsausdruck zutrifft.
- Manche dieser Ergänzungen können auch kombiniert werden.
  - Auch die **Select**-Anweisung können mit diesen Ergänzungen irgendwann unleserlich werden.



# Arbeitsmappen mit Makros speichern

- Wenn wir versuchen eine Arbeitsmappe mit Makros zu speichern, gibt es ein Problem:



## Warum ist das so?

VBA ist so mächtig, weil es viele Funktionen des Windows-Betriebssystems direkt verwenden kann. Das bedeutet aber auch, dass ein laufendes Excel-Makro großen Schaden außerhalb von Excel anrichten kann. Es gibt sogar Makroviren, die absichtlich Schaden anrichten sollen! Daher wurde für Excel ein neues Dateiformat entwickelt, das klar zeigt, dass es Tabellen und Makros enthält.

- Kurzum: das geht gar nicht!
  - Lösung: Stattdessen müssen wir die Datei in einem anderen Dateiformat, nämlich **\*.xlsm** und nicht \*.xlsx speichern.

