

VBA (Visual Basic for Application)

Variablen, Arrays und Schleifen

- Ein Computerprogramm
 - ... setzt einen Algorithmus in eine Form um, die von einem Computer verarbeitet werden kann.
 - ... teilt dem Computer mit, was er und wie er eine Aufgabe zu lösen hat.
- Der Begriff Programm
 - ... wird für einen Text genutzt, der in einer Programmiersprache abgefasst ist. Dieser Text wird als Quelltext bezeichnet und kann von einem Menschen gelesen werden.
 - ... wird für den von einem Computer ausführbaren Maschinencode verwendet. Um aus dem Quelltext einen Maschinencode zu generieren, muss dieser übersetzt werden.
- In diesem Kurs werden Computerprogramme in der Programmiersprache VBA geschrieben.

- Genau definierte Verarbeitungsvorschrift zur Lösung einer Aufgabe.
- Beschreibung eines Schemas, welches unter Verwendung von endlich vielen Arbeitsschritten ein bestimmtes Problem löst.
- Endliche Folge von Anweisungen, die nacheinander ausgeführt werden.
 - Anweisungen bestehen aus Schlüsselwörtern aus VBA sowie aus Operatoren und Operanden.
 - Manche Anweisungen werden nur unter bestimmten Bedingungen durchlaufen.
 - Die Anweisungen können unter bestimmten Bedingungen wiederholt werden.

Beispiele aus dem täglichen Leben:

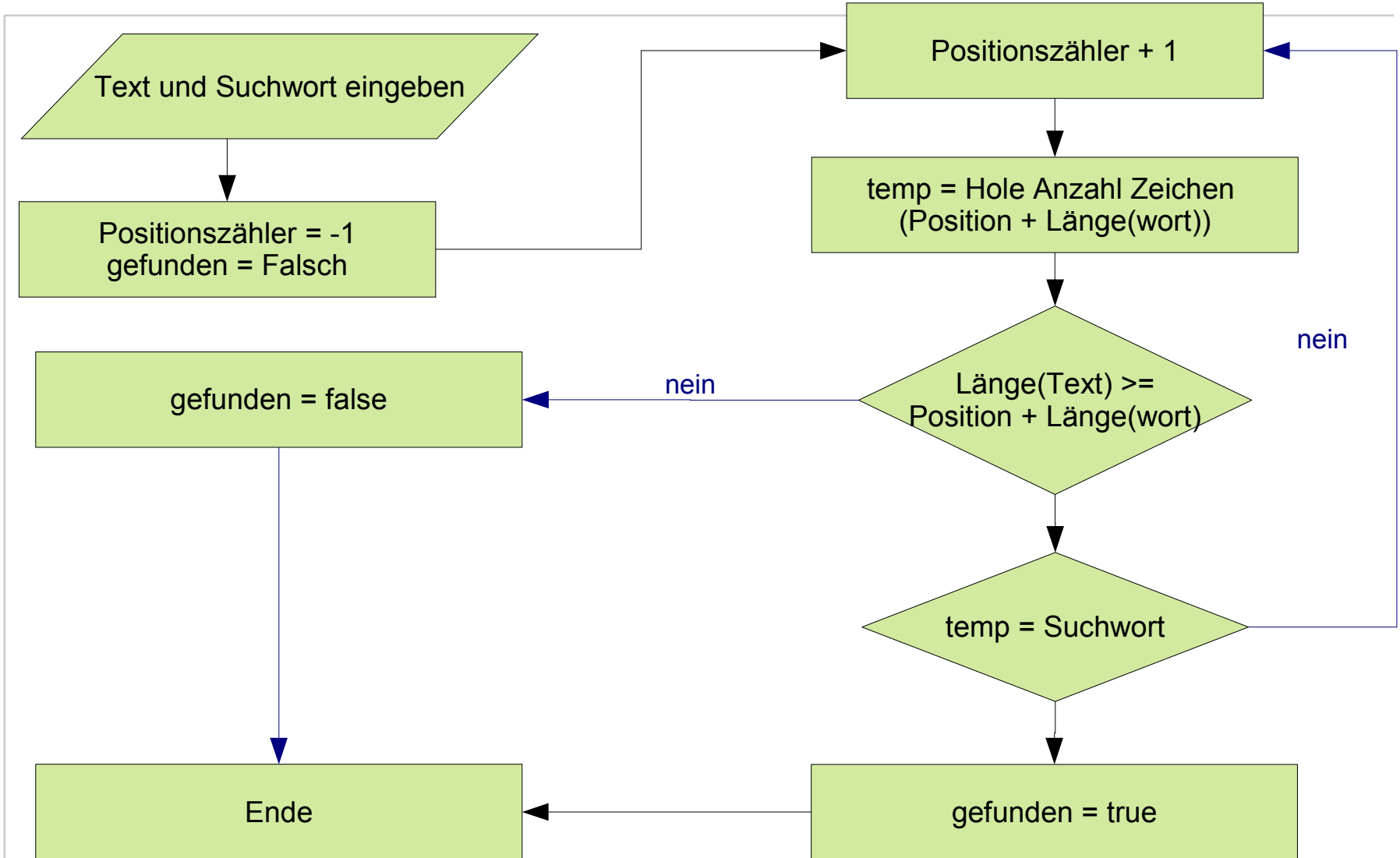
- Kochrezepte
- Steuerprogramme für technische Geräte (Waschmaschinen etc.)

- Tür der Waschmaschine öffnen.
- Max. 5 kg Wäsche (einer Farbe 😊) einfüllen.
- Tür der Waschmaschine schließen.
- Waschmittel passend zur Farbe der Wäsche in die kleine Schublade für den Hauptwaschgang füllen.
- Wasserzulauf öffnen.
- Waschprogramm wählen.
- Starttaste drücken.
- Waschvorgang abwarten.
- Nach Programm-Ende Maschine abstellen.
- Wasserzulauf schließen.
- Tür öffnen und Wäsche entnehmen.

- Ein Algorithmus benötigt endlich viele Arbeitsschritte.
- Ein Algorithmus ist beschreibbar.
- Jeder Arbeitsschritt ist ausführbar.
- Ein Algorithmus liefert unter identischen Startbedingungen immer das gleiche Endergebnis.
- Der Ablauf des Verfahrens ist eindeutig definiert.

- Aufgabe:
 - Suche ein bestimmtes Wort in einen Text.
- Eingangsdaten:
 - Variable *text*: Zeichenfolge (String), in der ein bestimmtes Wort gesucht werden soll.
 - Variable *wort*: Zeichenfolge, die gesucht wird.
- Ausgangsdaten:
 - Ist das Wort in der Zeichenfolge satz vorhanden?

Nach einem Wort suchen



- Operanden können
 - ... Variablen sein. Werte für Variablen können mit Hilfe der Tastatur eingegeben werden und durch das Programm verändert werden.
 - ... Konstanten sein. Konstanten bekommen einen Wert zugewiesen, der nicht durch das Programm verändert werden kann.
- Operatoren
 - Arithmetische Operatoren berechnen Werte aus ein, zwei oder drei Operanden.
 - Vergleichsoperatoren vergleichen zwei Werte.
 - Logische Operatoren verknüpfen verschiedene Werte.

Const PI = 3.14159265

Const Beschriftung = "Speichern"

- ... sind mit einem festen Wert verankert. Der Wert darf im ganzen Modul nicht mehr verändert werden.
- ... werden durch das Schlüsselwort *Const* gekennzeichnet.
- ... werden gleichzeitig deklariert und initialisiert.
 - Mit Hilfe des Gleichheitszeichen wird der Bezeichnung ein Wert zugewiesen.
 - Der Typ der Konstanten wird in Abhängigkeit des zugewiesenen Wertes ermittelt.
- Der Name einer Konstanten ist frei wählbar. Die Bezeichnung beginnt mit einem Großbuchstaben.
- ... werden für häufig genutzte Werte definiert. Die Werte können jederzeit zentral geändert werden.

- ... oder integrierte Konstanten werden von der Anwendung oder von Steuerelementen bereitgestellt.
- Das Präfix der Konstanten ist immer ein Hinweis auf seine Herkunft.
 - vb.. sind Konstanten, die direkt von der Programmiersprache definiert sind.
 - Mso... sind Konstanten, die von Microsoft Office bereitgestellt werden.
 - ac... sind Konstanten, die in der Access-Bibliothek definiert sind.

- Zahlen und Zeichen werden in Variablen gespeichert.
- ... werden durch das Programm verarbeitet.
- ... können durch Ausdrücke verändert werden.
- ... können mit Hilfe der Tastatur ein Wert zugewiesen bekommen.
- ... sind Platzhalter für einen bestimmten Typ von Wert.
- ... sind
 - ... in einem Kochrezept die Zutaten.
 - ... beim Waschen die Wäsche sowie das Pulver.
 - ... sind bei der Suche der zu durchsuchende Text, das Suchwort, der Positionszähler sowie die Anzeige "Gefunden oder nicht".

Sub varDef()

Dim faktor As Single

Dim wert As Integer

Dim multi As Single

Dim suchwort As String

Dim gefunden As Boolean

Pro Zeile wird eine Variable deklariert.
Jede Variable ist von einem bestimmten Datentyp.

faktor = 0.5

wert = 4

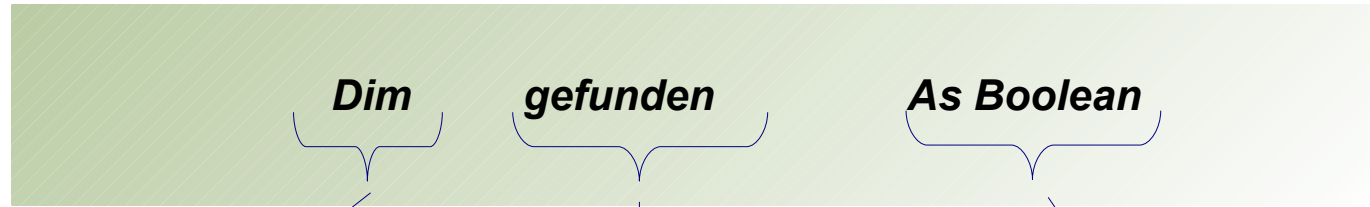
multi = faktor * wert

Mit Hilfe des Gleichheitszeichen wird den Variablen ein Wert zugewiesen.
Dieser Wert kann direkt angegeben oder mit Hilfe anderer Variablen berechnet werden.

suchwort = "VBA"

gefunden = False

End Sub



Wer kann auf die Variable zugreifen?

Der Variablenname ist frei wählbar. Die Bezeichnung sollte aber die Funktion der Variablen erläutern.

Mit dem Schlüsselwort *As* wird die Typbezeichnung eingeleitet. Hier wird ein Datentyp genutzt, der nur die Werte falsch oder wahr besitzt.

- ... müssen mit einem Buchstaben beginnen.
- Folgende Zeichen dürfen nicht in Variablennamen genutzt werden: Punkt [.], Prozentzeichen [%], Dollar-Zeichen [\$], Ausrufezeichen [!], Hash-Zeichen [#], Add-Zeichen [@] oder kaufmännisches Und [&].
- Umlaute, Satzzeichen oder Sonderzeichen sollten nicht in Variablennamen genutzt werden.
- ... sind einzigartig. Sie kommen nur einmal in ihrem Gültigkeitsbereich vor.
- Als Bezeichnung dürfen keine Schlüsselworte aus VBA genutzt werden.
- ... können bis 255 Zeichen lang sein.

- Der Variablenname sollte über die Art und Nutzung des Wertes Auskunft geben.
Beispiel: Für die Berechnung eines Kreisradius wird eine Variable mit dem Namen radius erzeugt.
Ungeeignete Variablennamen sind x2, a22 oder ähnlich kryptische Bezeichner.
- Ein Variablenname sollte den Sachverhalt, den die Variable repräsentiert vollständig und genau beschreiben. Auf diese Weise ergeben sich oft bereits gute Namen.
 - Gute Namen sind einfach zu lesen sind, da sie keine geheimnisvollen Abkürzungen enthalten und eindeutig sind.
 - Verzichten Sie auf Namen, die keinerlei Beziehung zum Inhalt der Variablen erkennen lassen.
- Variablennamen sollten keine Tätigkeitsworte wie zum Beispiel "Get" enthalten. "Get" ist Prozeduren von Objekten vorbehalten, die einen Eigenschaftswert des Objekts zurückgeben.
- Variablennamen, die nur aus einem einzelnen Zeichen bestehen, dürfen nur als Zähler oder Indizes genutzt werden.

- Variablennamen sollten, zur Unterscheidung von Konstanten, immer mit Kleinbuchstaben geschrieben werden.
- Variablennamen werden teilweise aus mehreren Namen zusammengesetzt. In der ungarischen Notation beginnt der Variablennamen mit einem kleinen Buchstaben. Alle Teilnamen beginnen mit einem Großbuchstaben.

Beispiel: `lineBuffer`, `nErrFlag`.

Andere Möglichkeit: `line_buffer`, `n_err_flag`.

- Als Präfix (erste Buchstabe des Variablennamens) wird häufig eine Abkürzung für den Datentyp genutzt. Beispiel:
 - `booAusdruck` für einen boolschen Ausdruck,
 - `intKM` für eine Integer-Variable oder
 - `curPreis` für einen Währungswert.

Aufgabe der Variablen	... sollte den Namen haben:	& nie den Namen haben:
Auftragsnummer	auftragNr, auftragNummer, auftrag_Nr	afg, afgNr, ag
Farbe eines Autos	farbeAuto, farbe_Auto	faAu, faAuto, au_fa, auto, FarbeEinesAutos, AutoGelb
Rechnungsdatum	rechnungDatum, bonDatum,	rgDate, rech_Datum, rgD, rechnung, rechnungsDatumMaerz
Spalte einer Tabelle	tabelleSpalte, tabSpalte	tsp, tspa, spalte

- Finden Sie die Variable, die nicht mit den zwei anderen übereinstimmt
 - eyechartl eyechartl eyechartl
 - CONFUSION C0NFUSION CONFUSION
 - hard2read hardZread hard2read
 - GRANDTOTAL GRANDTOTAL GRANDT0TAL
- Folgende Zeichen sind sich sehr ähnlich:
 - 1 und l (Kleines L)
 - l und l (kleines L und großes l)
 - 1 und l (großes l)
 - 0 und O
 - 2 und Z
 - S und 5
 - G und 6
- Machen Sie einen Bogen um Variablennamen, die Verwechslungen herausfordern.

- ... sind Baupläne für die Art der Variablen.
- ... geben über das Format eines Wertes, der in einer Variablen gespeichert wird, Auskunft.
- ... legen Regeln für die Interpretation und Verwendung eines Wertes fest.
- Folgende Kategorien sind in VBA vorhanden:
 - Ganze Zahlen (Integer) und boolesche Werte.
 - Fließkommazahlen oder Dezimalzahlen (Floating Point).
 - Datumswerte.
 - Zeichenfolgen.

Datentyp	Größe (Bytes)	Standardwert	Datenbereich
AS Byte	1	0	0 – 255
AS Integer [zahl%]	2	0	-32768 – 32.767
AS Long [zahl&]	4	0	-2.147.483.648 -2.147.483.647
AS Boolean	2	False	-1 (True) und 0 (False)

Datentyp		Größe (Bytes)	Standard- wert	Wertebereich
AS Single	[zahl!]	4	0	Negativer Wertebereich: -3,4E ⁺³⁸ bis -1,4E ⁻⁴⁵ Positiver Wertebereich: und +1,4E ⁻⁴⁵ bis +3,4E ³⁸ Einfache Genauigkeit (ca. 7 Stellen)
AS Double	[zahl#]	8	0	Negativer Wertebereich: -1,8E ³⁰⁸ bis -4,94E ⁻³²⁴ Positiver Wertebereich: +4,94E ⁻³²⁴ bis +1,8E ³⁰⁸ Doppelte Genauigkeit (ca. 15 Stellen)
AS Currency	[zahl@]	8	0	-9.22 E ⁺¹⁴ bis 9.22E ⁺¹⁵ (Der Datentyp besitzt 15 Vor- und 4 Nachkommastellen)

- In VBA wird als Dezimaltrennzeichen ein Punkt genutzt. Das Dezimaltrennzeichen der Daten in Access ist aber von den Ländereinstellungen des Systems abhängig.
- Es können Rundungsfehler auftreten. Dezimalzahlen werden einem Wert nur angenähert.
- Führende Nullen werden entfernt.
- VBA berücksichtigt keine Maßeinheiten oder Gewichte. Die Zahl 4.5 kann für den Nutzer ein Geldbetrag oder aber auch die Breite eines bestimmten Objekts darstellen. Im Code spielen diese Beziehungen aber keine Rolle. Sie als Entwickler müssen für die korrekte Umrechnungen zwischen den verschiedenen Einheiten sorgen.

Dim heute As Date

Dim zeit As Date

heute = #11/30/2007#

zeit = #11:30:00 AM#

- ... haben eine Größe von 8 Bytes.
- ... werden in VBA wie im englischsprachigen Sprachraum interpretiert. Datums- und Zeitwerte in Access werden aber in Abhängigkeit der Ländereinstellungen des Systems interpretiert.
- Konstante Datumswerte werden durch Hash-Zeichen begrenzt.
- Zeitangaben zwischen 00:00:00 und 23:59:59 werden verwaltet.
- Datumangaben
 - ... zwischen 01.01.0100 und 31.12.9999 werden verwaltet.
 - ... richten sich nach dem Gregorianischen Kalender, der in Europa ab 1582 eingeführt wurde.
- Zweistellige Datumangaben sollten sehr vorsichtig genutzt werden.
 - Die zweistelligen Zahlen 0 bis 30 bezeichnen Jahre im 21. Jahrhundert.
 - Die zweistelligen Zahlen 30 bis 99 bezeichnen Jahre im 20. Jahrhundert.

```
Dim satz As String  
Dim plz As String * 5
```

```
satz = "Hello World"  
plz = "123456"
```

- ... sind ab Version 4 nur durch die Größe der Festplatte begrenzt.
- ... können auf eine bestimmte Länge begrenzt werden. Falls mehr Zeichen eingegeben werden, wird die Zeichenkette ohne Warnmeldung beschnitten.
- ... werden immer durch Anführungszeichen begrenzt.
- ... können alle ANSI-Zeichen enthalten. Viele Druck- und Steuerungszeichen sind als Konstanten vorhanden.
- ... können seit Access 2000 auch Unicode-Zeichen verarbeiten.
- ... müssen für Postleitzahlen oder Telefon-Vorwahlnummern genutzt werden.

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 = L	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 = }

126 = ~	144 = É	162 = ó	180 = †	198 = ã	216 = ï	234 = Û	252 = ³
127 = ∆	145 = æ	163 = ú	181 = Á	199 = Ã	217 = √	235 = Ù	253 = ²
128 = Ç	146 = fl	164 = ñ	182 = Â	200 = ℒ	218 = √	236 = ý	254 = ■
129 = ü	147 = ô	165 = Ñ	183 = À	201 = √	219 = ■	237 = Ý	255 =
130 = é	148 = ö	166 = ª	184 = ©	202 = √	220 = ■	238 = ¯	
131 = â	149 = ò	167 = °	185 = ¶	203 = √	221 = ¡	239 = ´	
132 = ä	150 = û	168 = ¸	186 = ¶	204 = √	222 = Ì	240 = -	
133 = à	151 = ù	169 = ®	187 = ¶	205 = =	223 = ■	241 = ±	
134 = å	152 = ÿ	170 = ¬	188 = ¶	206 = √	224 = Ó	242 = =	
135 = ç	153 = Ö	171 = ½	189 = ¢	207 = ♂	225 = ß	243 = ¾	
136 = ê	154 = Ü	172 = ¼	190 = ¥	208 = δ	226 = Ô	244 = ¶	
137 = ë	155 = ø	173 = ¡	191 = ¶	209 = Ð	227 = Ò	245 = §	
138 = è	156 = £	174 = «	192 = ∟	210 = Ê	228 = õ	246 = ÷	
139 = ï	157 = Ø	175 = »	193 = ⊥	211 = Ë	229 = Õ	247 = °	
140 = î	158 = ×	176 = ▒	194 = ∟	212 = È	230 = μ	248 = °	
141 = ì	159 = f	177 = ▒	195 = †	213 = √	231 = Þ	249 = ¨	
142 = Ä	160 = á	178 = ▒	196 = -	214 = Ì	232 = þ	250 = ·	
143 = Å	161 = í	179 =	197 = †	215 = Î	233 = Ú	251 = ¹	

Option Explicit

- ... steht immer am Anfang eines Moduls und gilt für das gesamte Modul.
- Bevor eine Variable genutzt werden kann, muss diese deklariert werden.
- ... kann automatisch gesetzt werden.
 - Wählen Sie das Menü Extras – Optionen aus.
 - Die Registerkarte Editor ist aktiv und liegt im Vordergrund.
 - Falls kein Häkchen angezeigt wird, klicken Sie in das Optionskästchen Variablendeklaration erforderlich. Anschließend ist die Option aktiv.
- Vorteile:
 - Keine Variable hat den gleichen Namen. Sobald eine Bezeichnung zweimal vergeben wird, wird eine Fehlermeldung angezeigt,
 - Die Lesbarkeit und damit die Wartbarkeit des Codes wird erhöht.

```
Sub Pruef()
```

```
Dim heute As Date
```

```
heute = Now
```

```
Debug.Print IsDate(heute)
```

```
Debug.Print IsNumeric(heute)
```

```
Debug.Print VarType(heute)
```

```
Debug.Print TypeName(heute)
```

```
End Sub
```

' Aktuelles Datum zuweisen

' Kann der Wert als Datum

' ... oder Zahl interpretiert werden?

' Der Datentyp als Zahl

' Der Datentyp als String

- ... ist ein universeller Datentyp.
- ... benötigt bei numerischen Werten ca. 16 Bytes und bei Strings 22 Bytes plus die Textlänge.
- ... wird genutzt, wenn kein Datentyp angegeben wird.
- ... ist sehr speicheraufwendig.
- ... kann zu unnötigen Fehlermeldungen führen. Die Variable kann jeden beliebigen Datentyp annehmen. Es findet keine Typüberprüfung statt.

```
Sub Pruef()  
  Dim wert As Variant  
  
  wert = NULL                                ' NULL kennzeichnet ungültige Variablen  
  
  Debug.Print IsEmpty(wert)                 ' Ist die Variable leer?  
  
  Debug.Print IsNull(wert)                  ' Hat die Variable ungültige Daten?  
  
  Debug.Print TypeName(wert)               ' Der Datentyp NULL wird ausgegeben  
  
  wert = 5  
  Debug.Print TypeName(wert)               ' Der Datentyp Integer wird ausgegeben  
End Sub
```

- ... besteht aus Operanden und Operatoren, die nach bestimmten Regeln zusammengesetzt werden.
- ... ist eine Verarbeitungsvorschrift, die einen Wert als Ergebnis liefert.
- ... verändert den Wert von Variablen entsprechend des angegebenen Datentyps.
- Beispiele:
 - `preis * 0.16`
 - `messpunkt > 0`
 - `addition(1,2)`
 - `(a >= b) AND (a >= c)`

- ... bestehen aus einem Ausdruck, der automatisch nach dem Verlassen der Zeile abgeschlossen ist.
- ... werden meist sequentiell ausgeführt.
- ... symbolisieren eine bestimmte Aktion, die zu einem Algorithmus gehört.
- ... sind die kleinste Einheit einer Prozedur.

Sub addition()

Const zahl = 10

Dim wert As Integer

Dim summe As Integer

Dim ausgabe As String

wert = InputBox("Bitte geben Sie einen Wert ein:")

' Wert einlesen

summe = wert + zahl

Debug.Print summe

ausgabe = wert & " + " & zahl & " = " & summe


' String bilden

MsgBox(summe)

' String ausgeben

End Sub

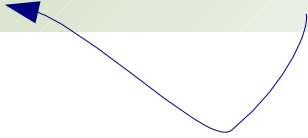
- Öffnen Sie das Direktfenster mit Hilfe des Menüs Ansicht – Direktfenster.
- Die Anweisung *Debug.Print variable* schreibt den Wert einer Variablen in das Direktfenster.
- Mit Hilfe von <STRG>+<A> wird der Inhalt des Direktfensters markiert und kann mit <ENTF> gelöscht werden.
- Sie können auch direkt im Direktfenster arbeiten.
 - Klicken Sie mit den Mauszeiger auf eine freie Stelle im Fenster.
 - Die Eingabe *?Variable* gibt den Wert einer Variablen aus, sobald Sie die Anweisung mit <ENTER> abgeschlossen haben.
 - Die Anweisung *variable = 20* verändert den Wert der Variablen.
 - Die Anweisung *?Left(zeichen, 4)* führt die Funktion *Left* aus und das Ergebnis wird in der nächsten Zeile ausgegeben. Im Direktfenster können Sie vordefinierte Funktionen ausprobieren, um ihre Arbeitsweise zu verstehen.

- ... beginnen immer mit dem Apostroph.
- ... enden automatisch sobald eine neue Zeile beginnt.
- ... können an beliebiger Stelle platziert werden.
 - Eine Prozedur sollte immer mit einem Kommentar, der die Aufgabe erläutert, beginnen.
 - Ein Kommentar kann oberhalb oder rechts neben einer Anweisung platziert werden, um deren Bedeutung zu erklären.
- Mehrere Zeilen können mit den entsprechenden Symbolen  in der Symbolleiste Bearbeiten auskommentiert werden.

- Kommentare sollen dazu führen, dass man das Programm ohne fremde Hilfe verstehen kann.
- Offensichtliche Dinge sollten nicht in Prosa wiederholt werden. Ein negatives Beispiel:
 - `a = c + b;` 'a ist die Summe von c und b
- Kommentare müssen bei Änderungen am Programm aktualisiert werden.
- Kommentare sollten sich auf das Warum beziehen und weniger auf das Wie.
- Wenn die Funktionalität des Programms aus dem Code klar zu erkennen ist, wird ein Kommentar nicht benötigt, .
- Keine redundanten oder überflüssigen Kommentare.

- Einstellige Operatoren:
 - Vorzeichen für positive und negative Zahlen.
- Zweistellige Operatoren:
 - Arithmetische Operatoren.
 - Vergleichsoperatoren.
 - Logische Operatoren zum Verknüpfen von Ausdrücken.
 - Zuweisungsoperator.
 - Textverknüpfungen.

```
wert = 4  
faktor = 0.5  
ausgabe = "Hello World"
```



- Mit Hilfe des Gleichheitszeichens wird der Variablen ein Wert zugewiesen.
- Die Variable wird nicht mit einem Wert verglichen!
- Dem Ausdruck links vom Gleichheitszeichen wird das Ergebnis des Ausdrucks rechts vom Gleichheitszeichen zugewiesen.
- Konstanten dürfen in Anweisungen nur rechts vom Gleichheitszeichen stehen. Der Wert einer Konstanten darf nicht verändert werden.

Operator	Rechenart	Beispiel
\wedge	Potenzrechnung	$7^3 = 343$
$/$	Division	$7 / 3 = 2.33$ $7 / 0 = \text{Fehler}$
\backslash	Division mit Integer-Zahlen	$7 \backslash 3 = 2$ $7.2 \backslash 3.1 = \text{Fehler}$
Mod	Rest einer Division mit Integer-Zahlen	$7 \text{ Mod } 3 = 1$ $7.2 \text{ Mod } 3.1 = \text{Fehler}$
$*$	Multiplikation	$7 * 3 = 21$
$+$	Addition	$7 + 3 = 10$
$-$	Subtraktion	$7 - 3 = 4$

- Bei Ausdrücken gelten die Rechenregeln der Mathematik:
 - Klammer vor
 - Potenz vor
 - Punktrechnung vor
 - Strichrechnung.
- Beispiele:
 - Liefert der Ausdruck $(2 + 4 * 2)$ das gleiche Ergebnis wie $((2 + 4) * 2)$?
 - Liefert der Ausdruck $(2^4 / 4)$ das gleiche Ergebnis wie $(2^{(4 / 4)})$?
- Um fehlerhafte Ausdrücke zu vermeiden, sollten große Ausdrücke geklammert werden!

Dim zeichenfolge As String

zeichenfolge = "Eis"

zeichenfolge = zeichenfolge & "bär"

zeichenfolge = 10 & " " & zeichenfolge

zeichenfolge = zeichenfolge & vbCrLf

- Mit Hilfe des kaufmännischen Unds (&) können Zeichenfolgen miteinander verknüpft werden.
 - Zahlenwerte werden automatisch in Strings umgewandelt.
 - Es können Variablen und / oder Konstanten miteinander verknüpft werden.
- In einigen Büchern wird auch das Pluszeichen genutzt. Aber `10 + " " + zeichenfolge` erzeugt einen Fehler. Hier wird versucht 10 zu einem String zu addieren.

Konstanten	ANSI-Zeichen	Beschreibung
vbCrLf	Chr(13) & Chr(10)	Kombination aus Wagenrücklauf und Zeilenvorschub
vbNewLine	Chr(13) & Chr(10)	Plattformspezifischer Zeilenumbruch
vbCr	Chr(13)	Wagenrücklauf (Carriage Return)
vbLF	Chr(10)	Zeilenvorschub (Line Feed)
vbTab	Chr(9)	Tabulatorzeichen
vbBack	Chr(8)	Rückschrittzeichen

- Steuerzeichen sind die nicht druckbaren Zeichen eines Zeichensatzes. Sie werden zur Formatierung der Bildschirmausgabe benötigt werden.
- Beispiele:
 - Den Zeilencursor an den Anfang der nächsten Zeile setzen: *vbCrLf* oder *Chr(13) & Chr(10)* .
 - *Chr(34)* ermöglicht die Darstellung des Anführungszeichens.

■ Chr(...)

- ... wird ein Integer-Wert von 0 bis 255 übergeben.
- Die Funktion liefert das dazugehörige ANSI-Zeichen zurück.
- Zum Beispiel *Chr(65)* liefert "A" zurück.

■ Asc(...)

- ... wird ein einzelnes Zeichen übergeben.
- Die Funktion liefert die dazugehörige Dezimalzahl zurück.
- Zum Beispiel *Asc("A")* liefert 65 zurück.

- ... oder implizite Typumwandlung.
- ... bei verschiedenen numerischen Datentypen in einem Ausdruck:
 - VBA versucht einen Datentyp zu finden, in dem alle Werte dargestellt werden können.
 - Die Umwandlung verursacht meist keine Fehler.
- ... bei Strings, die als Zahlen interpretiert werden:
 - In Berechnungen werden Strings wie zum Beispiel "10" automatisch in einem numerischen Datentyp umgewandelt?
- ... bei Zahlen, die mit Strings verknüpft werden:
 - Wenn das kaufmännische Und als Verknüpfungsoperator genutzt wird, werden Zahlen automatisch als String interpretiert.

- `ergebnis = "10" + 20`
 - ... erzeugt keinen Fehler.
 - 10 wird automatisch als numerischer Wert interpretiert. Als Ergebnis wird 30 ausgegeben.
- `ergebnis = "A" + 20`
 - ... erzeugt einen Fehler.
 - Mit Strings kann nicht gerechnet werden.
- `ergebnis = ASC("A") + 20`
 - ... erzeugt keinen Fehler.
 - Die Funktion liefert einen numerischen Wert mit dem gerechnet werden kann.
- `ergebnis = "10 Eisbären" + 20`
 - ... erzeugt einen Fehler.
 - Mit einem String kann nicht gerechnet werden.

```
Debug.Print CInt(2.3)           ' Ergebnis 2  
Debug.Print CByte(256)        ' Fehler: Überlauf  
Debug.Print CBool(-1)         ' Ergebnis wahr  
Debug.Print CSng(22.4567878994556) ' Ergebnis 22,45679  
Debug.Print CDbI(24)          ' Ergebnis 24.0  
Debug.Print CCur(2.37894)    ' Ergebnis 2,3789  
Debug.Print CDate("2.4.07")  ' Ergebnis #04.02.2007#  
Debug.Print CStr(2.3)         ' Ergebnis "2,3"
```

```
Dim result As VbMsgBoxResult
```

```
result = MsgBox(prompt [, buttons] [, title] [, helpfile, context ])
```

- ... ist ein Dialogfeld mit Schaltflächen zur Anzeige von Text.
- ... kann für kurze Status- oder Fehlermeldungen genutzt werden.
- ... bekommt folgende Parameter übergeben.
 - *prompt* ist eine Zeichenkette, die im Dialogfenster angezeigt wird.
 - *buttons* legt fest, welche Schaltflächen angezeigt werden. Alle Möglichkeiten werden mit Hilfe von vordefinierten Konstanten eingestellt. Falls keine Angaben gemacht werden, wird eine Schaltfläche zur Bestätigung der Meldung angezeigt.
 - *title* ist eine Zeichenkette, die in der Titelleiste des Dialogfensters angezeigt wird.
- ... gibt *result* zurück. *result* ist ein Integer-Wert, der die gedrückte Schaltfläche symbolisiert.

```
Const titelText = "Addition"  
Dim result As VbMsgBoxResult  
Dim x As Integer  
Dim message As String
```

```
MsgBox("Danke, ...")
```

```
message = "Die Variable x hat den Wert " & x  
MsgBox(message, vbOKOnly)
```

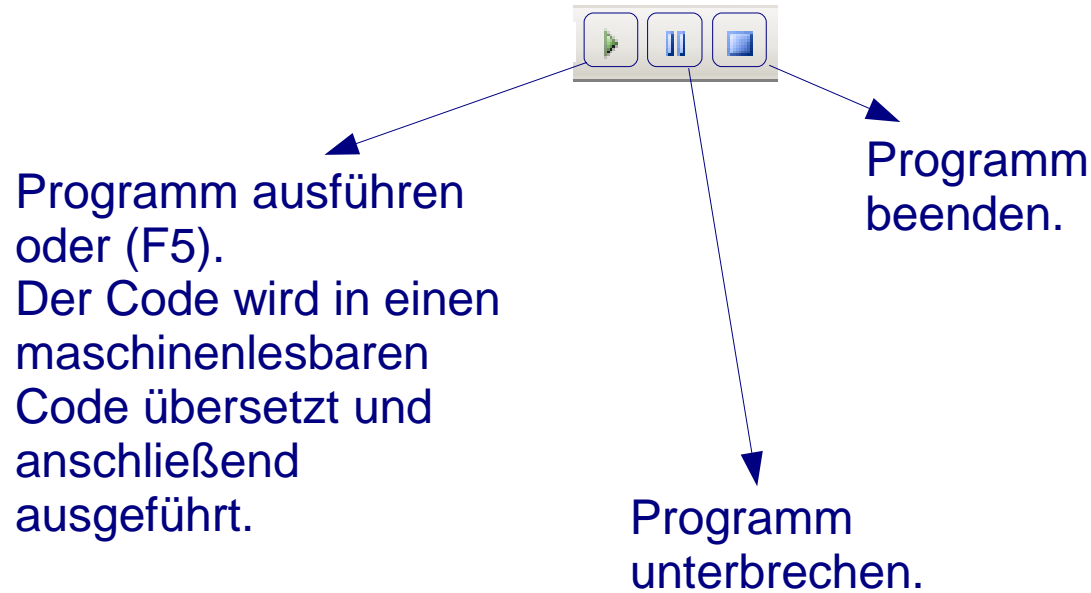
```
message = "Die Addition hat das Ergebnis" & (x + 3)  
result = MsgBox(message, , titelText)
```

```
result = MsgBox(message, vbYesNo , titelText)
```

Dim result As String

```
result = InputBox(prompt [, title] [, default] [, xPos] [, yPos] [, helpfile, context ])
```

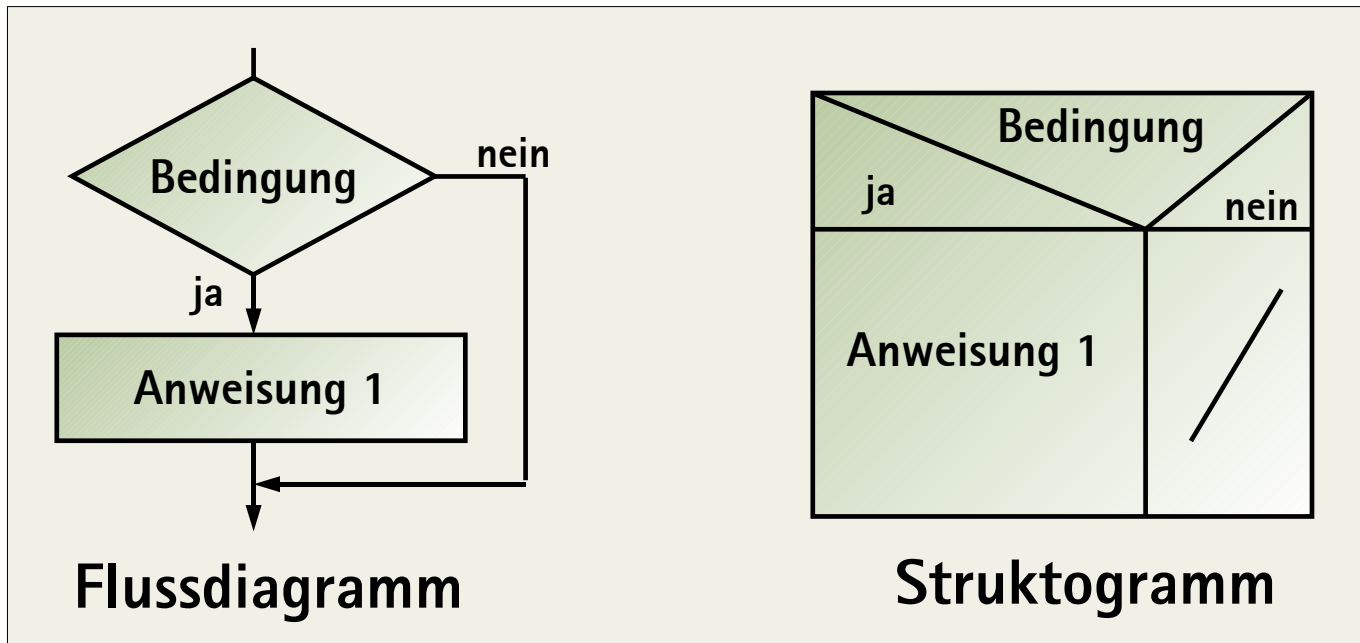
- ... ist ein Dialogfeld mit Schaltflächen zum Einlesen von Text.
- ... kann vom Benutzer Daten abfragen.
- ... bekommt folgende Parameter übergeben:
 - *prompt* ist eine Zeichenkette, die im Dialogfenster angezeigt wird.
 - *title* ist eine Zeichenkette, die in der Titelleiste des Dialogfensters angezeigt wird.
 - *default* gibt ein Standardwert vor.
 - *xPos* und *yPos* legen die Position der linken, oberen Ecke des Dialogfeldes fest. Die Angaben werden in Twips per Pixel gerechnet (20 Twips sind 1 Pixel).
- ... gibt *result* zurück. *result* ist ein String. Als Trennzeichen zwischen Vor- und Nachkommastellen wird ein Komma statt einem Punkt genutzt.



- Mit Hilfe von Kontrollstrukturen kann der Programmablauf beeinflusst werden.
- In Abhängigkeit vom Wert einer oder mehrerer Variablen wird der Programmablauf gesteuert.
- Es gibt folgende Möglichkeiten:
 - Auswahlanweisungen:
 - Bedingte Anweisungen; if – then - Anweisungen
 - Fallunterscheidungen; select - case - Anweisungen
 - Schleifen (Iterationsanweisungen):
 - Kopfgesteuerte Schleifen: while – wend – Schleifen oder do – loop -Schleifen
 - Fußgesteuerte Schleifen; do – loop - Schleifen
 - Zählschleifen; for – next - Schleifen
- Schleifen können vom Programmierer unter bestimmten Bedingungen unterbrochen werden.

```
If (bedingung) Then  
Anweisung  
End If  
  
Anweisung
```

- ... oder Selektionsanweisungen.
- Dem Schlüsselwort *If* (wenn) folgt eine Bedingung.
- Wenn diese Bedingung erfüllt ist, dann (*Then*) werden die Anweisungen bis zum *End If* ausgeführt.
- Andernfalls werden die Anweisungen nach dem *End If* ausgeführt.



```
bestellPreis = (bestellmenge * preis)
```

```
rabatt = 0.0
```

```
If (bestellmenge > 100) Then
```

```
    rabatt = 0.03
```

```
    rabattPreis = bestellPreis * rabatt
```

```
    bestellPreis = bestellPreis - rabattPreis
```

```
End If
```

```
Debug.Print "Endsumme: " & bestellPreis
```

- ... sind Ausdrücke, die einen boolschen Wert zurückliefern.
- ... vergleichen mit Hilfe von bestimmten Operatoren zwei Werte.
- ... sind zum Beispiel:
 - Wenn die Bestellmenge eine gewisse Höchstmenge überschreitet...
 - Wenn der Kontostand dem Dispo entspricht...
 - Wenn die Strecke A doppelt so lang ist wie Strecke B...
 - Wenn die Warenmenge eine Mindestmenge unterschreitet...

Operator	Rechenart	Beispiel
=	ist gleich	$(7 = 3) \Rightarrow \text{False}$
<	ist kleiner als	$(7 < 3) \Rightarrow \text{False}$
<=	ist kleiner gleich als	$(7 <= 3) \Rightarrow \text{False}$
>	ist größer als	$(7 > 3) \Rightarrow \text{True}$
>=	ist größer gleich als	$(7 >= 3) \Rightarrow \text{True}$
<>	ist ungleich	$(7 <> 3) \Rightarrow \text{True}$

Dim zeichen As String
Dim gleich As Boolean

gleich = zeichen LIKE "World"
gleich = (zeichen = "World")

- **Like**
 - ... ist ein Synonym für "ist gleich".
 - ... kann mit Platzhalten arbeiten.
 - ... ist abhängig von der Anweisung *Option Compare* am Anfang des Moduls.
- **Folgende Vergleichsarten werden für Strings genutzt:**
 - *Option Compare Binary* oder als Konstante *vbBinaryCompare* ist die Standardeinstellung. Die Gewichtung der Zeichen ist von der Position in der ANSI-Zeichentabelle abhängig (A < B < C... < Z < a < b < c < z).
 - *Option Compare Text* oder als Konstante *vbTextCompare* führt einen Textvergleich durch. Hierbei wird folgende Gewichtung genutzt: A = a < B = b ...
 - *Option Compare Database* oder als Konstante *vbDatabaseCompare* führt einen Vergleich in Abhängigkeit der Einstellungen der Datenbank durch.

Dim zeichen As String
Dim gleich As Boolean

gleich = zeichen LIKE "Eisscholle"

' Das Sternchen ersetzt eine beliebige Anzahl von Zeichen
gleich = zeichen LIKE "Ei*e"

' Das Fragezeichen ersetzt einen Buchstaben
gleich = zeichen LIKE "Fisch?"

' Das Hash-Zeichen ersetzt einen numerischen Wert von 0 bis 9
gleich = zeichen LIKE "#1234"

Dim zeichen As String
Dim gleich As Boolean

gleich = zeichen LIKE "Eisscholle"

' In eckigen Klammern wird eine Liste von Zeichen als Platzhalter angegeben.
' Die Listenelemente werden durch Kommata getrennt.

gleich = zeichen LIKE "B[a, u]ch"

' In den eckigen Klammern wird ein Bereich von Zeichen angegeben.
' In diesem Fall werden die Buchstaben B bis D als Anfangsbuchstaben genutzt.
' Anschließend können beliebig viele Zeichen folgen.

gleich = zeichen LIKE "[B-D]"*

' In diesem Fall wird mit Hilfe des Ausrufezeichens B ausgeschlossen.
' Als zweiter Buchstabe kann ein a oder u folgen.
' Mit der Zeichenfolge ch endet das Wort.

gleich = zeichen LIKE "[!B][a, u]ch"

Dim zeichen As String
Dim gleich As Boolean
Dim vergleich As Integer

gleich = zeichen LIKE "World"
vergleich = StrComp(zeichen, "World", vbTextCompare)

- Mit Hilfe der vordefinierten Funktion *StrComp()* wird eine Zeichenfolge in Abhängigkeit einer anderen Zeichenfolge verglichen.
- Der Funktion wird die zu vergleichende Zeichenfolge sowie das Muster übergeben. Falls gewünscht, kann die Sortierreihenfolge der Zeichen übergeben werden.
- Folgende Werte werden von der Funktion zurückgegeben
 - 0, wenn die Zeichenfolge lexikographisch gleich sind.
 - - 1, wenn die Zeichenfolge lexikographisch vor dem Muster liegt.
 - 1, wenn die Zeichenfolge lexikographisch hinter dem Muster liegt.
 - NULL, wenn das Muster und / oder Zeichenfolge leer sind.

- ... oder relationale Operatoren.
- ... verknüpfen zwei oder mehr Bedingungen miteinander.
- Folgende Möglichkeiten sind vorhanden:
 - AND (Und, Konjunktion) ist nur wahr, wenn alle Bedingungen wahr sind.
 - OR (Oder, Disjunktion) ist wahr, sobald eine der Bedingungen wahr ist.
 - NOT (Negation) invertiert den booleschen Wert der Bedingung.
 - XOR (Antivalenz) ist wahr, wenn zwei Bedingungen ein unterschiedliches Ergebnis liefern.
 - EQL (Äquivalenz) ist wahr, wenn zwei Bedingungen das gleiche Ergebnis liefern
 - IMP (Implikation) ist wahr, wenn der erste Parameter falsch ist oder der zweite Parameter wahr ist.

Bedingung							
a	b	Not a	a AND b	a OR b	a XOR b	a EQV b	a IMP b
false	false	true	false	false	false	true	true
false	true	true	false	true	true	false	true
true	false	false	false	true	true	false	false
true	true	false	true	true	false	true	true

```
Dim zahlA As Integer  
Dim zahlB As Integer  
Dim result As Integer
```

```
If NOT(zahlB >= 0) Then  
    Debug.Print "Division durch Null nicht erlaubt"  
Else  
    result = zahlA / zahlB  
End If
```

```
If (zahlB <> 0) Then  
    result = zahlA / zahlB  
Else  
    Debug.Print "Division durch Null nicht erlaubt"  
End If
```

```
bestellPreis = (bestellmenge * preis)
```

```
rabatt = 0.0
```

```
If ((bestellmenge > 100) AND (bestellmenge < 500)) Then
```

```
  rabatt = 0.03
```

```
  rabattPreis = bestellPreis * rabatt
```

```
  bestellPreis = bestellPreis - rabattPreis
```

```
End If
```

```
Debug.Print "Endsumme: " & bestellPreis
```

```
Dim zahl As Integer  
Dim faktor As Integer
```

```
If ( (zahl < 10) OR (zahl > 20) ) Then  
    Debug.Print "Die Zahl liegt nicht im gewünschten Zahlenraum"  
End If
```

- **Beispiel:** `(var1 <> var2) AND (var2 > 10)`
 - Zuerst wird die linke Bedingung `(var1 <> var2)` ausgewertet.
 - Anschließend wird die rechte Bedingung `(var2 > 10)` ausgewertet.
 - Beide Ergebnisse müssen in diesen Fall wahr sein, um die if-Anweisung auszuführen.
- Sie können die Operatoren beliebig oft in beliebiger Mischung in einer Bedingung nutzen.
- Um die Lesbarkeit zu erhöhen, sollten die verschiedenen Elemente der Bedingung mit runden Klammern zusammengefasst werden.
- Es sollten nicht mehr als drei logische Operatoren in einer Bedingung genutzt werden.
- Falls verschiedene Operatoren gemischt werden, muss die Bindung der Operatoren beachtet werden. NOT hat eine höhere Bindung als AND. AND hat eine höhere Bindung als OR. Anschließend folgen XOR, EQV und IMP.

Not ((5 > 4) AND (3 <> 2))

wahr

wahr

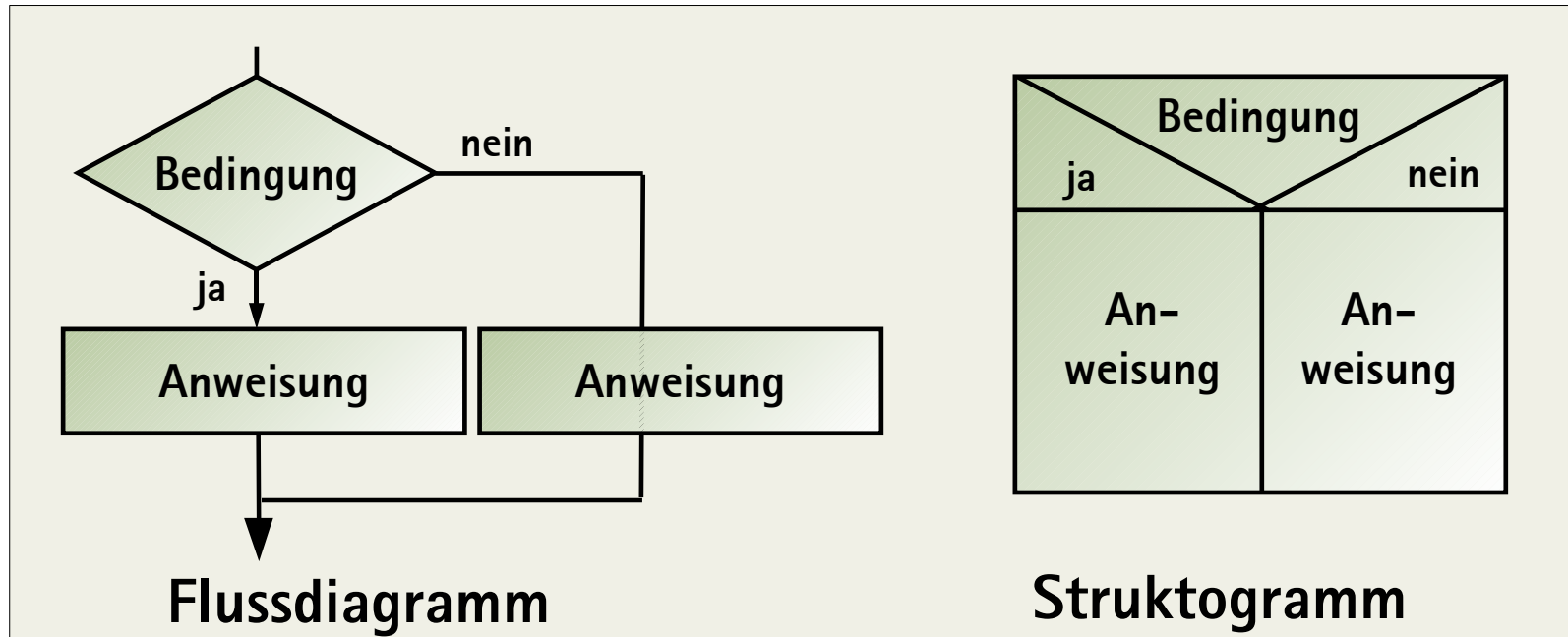
wahr

false

	Hoch									Niedrig
Hoch	^	&	<>	=	<	<=	=>	>	Like	NOT
	+ - (Vorzeichen)									AND
	* /									OR
	\									XOR
	Mod									EQV
Niedrig	+ -									IMP

```
If (bedingung) Then  
    Anweisung  
Else  
    Anweisung  
End If  
  
Anweisung
```

- Dem Schlüsselwort *If* (wenn) folgt eine Bedingung.
- Wenn diese Bedingung erfüllt ist, dann (*Then*) werden die Anweisungen bis zum *Else* ausgeführt.
- Andernfalls werden die Anweisungen in dem *Else* (Ansonsten) - Zweig ausgeführt.



```
bestellPreis = (bestellmenge * preis)
```

```
If (bestellmenge > 100) Then
```

```
    rabatt = 0.03
```

```
    rabattPreis = bestellPreis * rabatt
```

```
    bestellPreis = bestellPreis - rabattPreis
```

```
Else
```

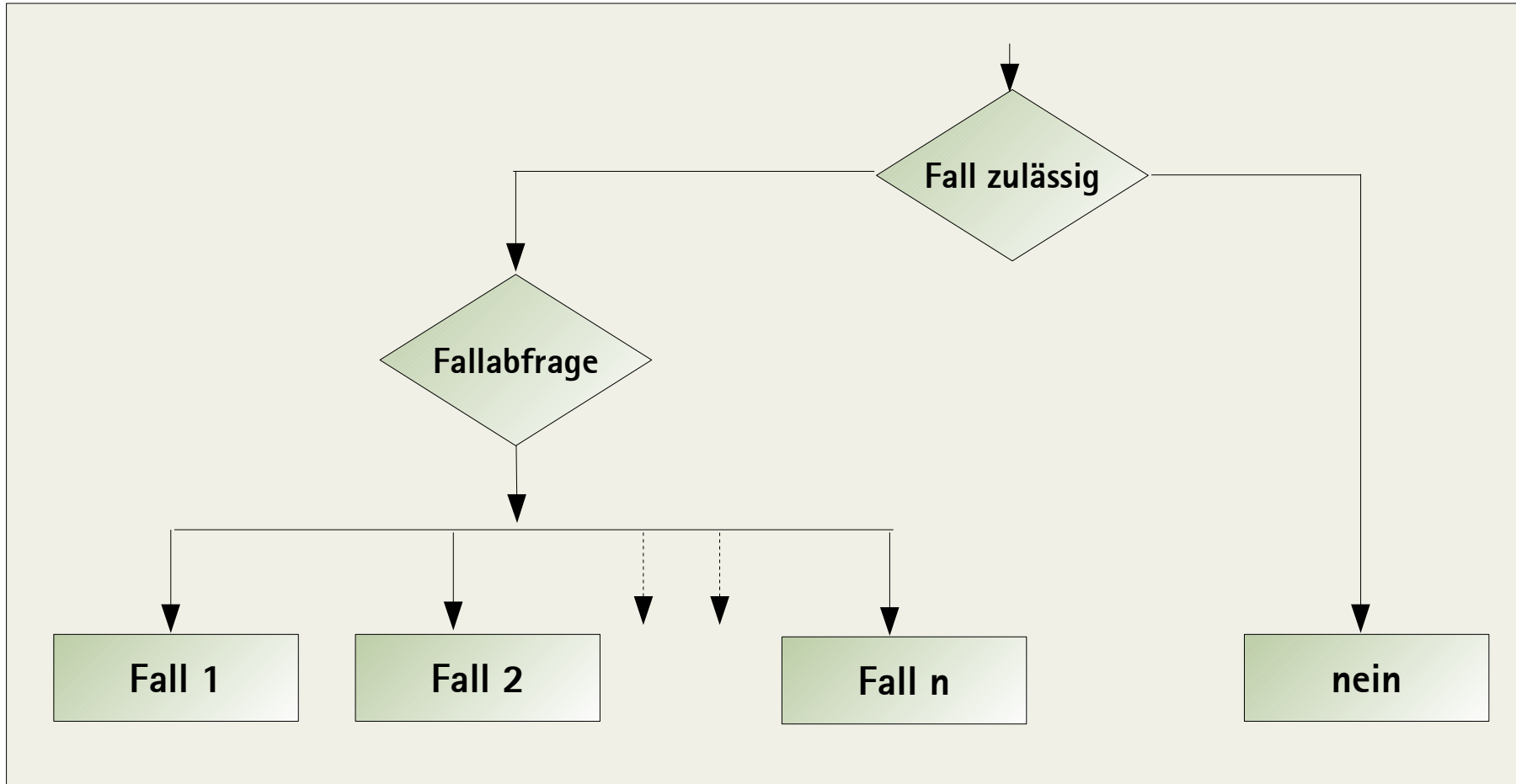
```
    rabatt = 0.0
```

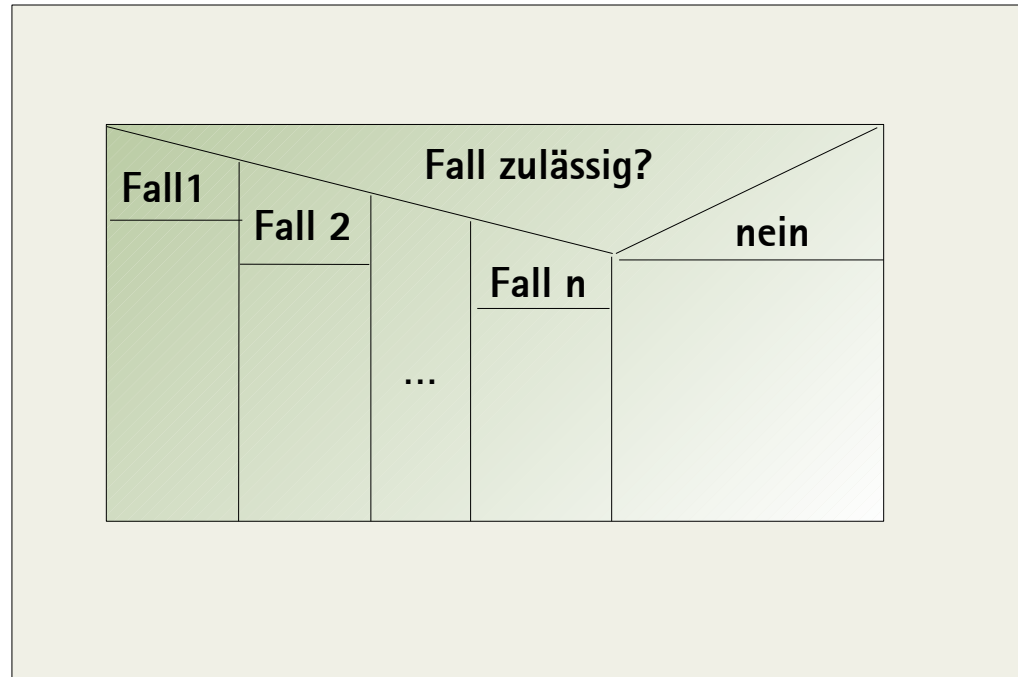
```
End If
```

```
Debug.Print "Endsumme: " & bestellPreis
```

```
If (bedingung) Then  
    Anweisung  
Elseif (Bedingung) Then  
    Anweisung  
Else  
    Anweisung  
End If  
  
Anweisung
```

- Der if-Anweisung folgt eine weitere if-Anweisung.
- Die if-Anweisungen werden wie Sprossen in Leitern aufgereiht.





```
bestellPreis = (bestellmenge * preis)
```

```
If (bestellmenge > 100) AND (bestellmenge < 500) Then
```

```
  rabatt = 0.03
```

```
  rabattPreis = bestellPreis * rabatt
```

```
  bestellPreis = bestellPreis – rabattPreis
```

```
Elseif (bestellmenge >= 500) AND (bestellmenge < 1000) Then
```

```
  rabatt = 0.05
```

```
  rabattPreis = bestellPreis * rabatt
```

```
  bestellPreis = bestellPreis – rabattPreis
```

```
Else
```

```
  rabatt = 0.0
```

```
End If
```

```
Debug.Print "Endsumme: " & bestellPreis
```

- Innerhalb einer if-Anweisung wird eine weitere if-Anweisung aufgerufen.
- Eine else-Anweisung bezieht sich immer auf die vorgehende if-Anweisung.
- Wegen der besseren Lesbarkeit sollten die verschiedenen Ebenen eingerückt werden.

```
If (bedingung) Then  
    Anweisung
```

```
If (bedingung) Then  
    Anweisung
```

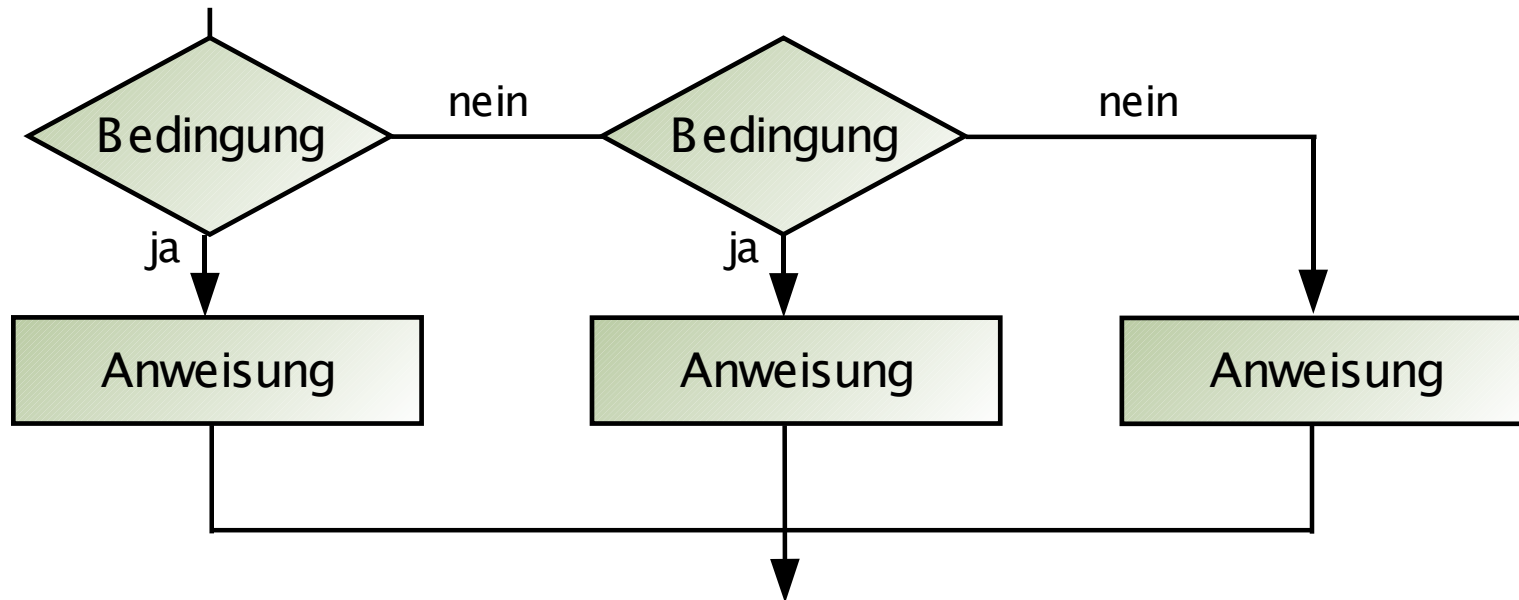
```
Else  
    Anweisung
```

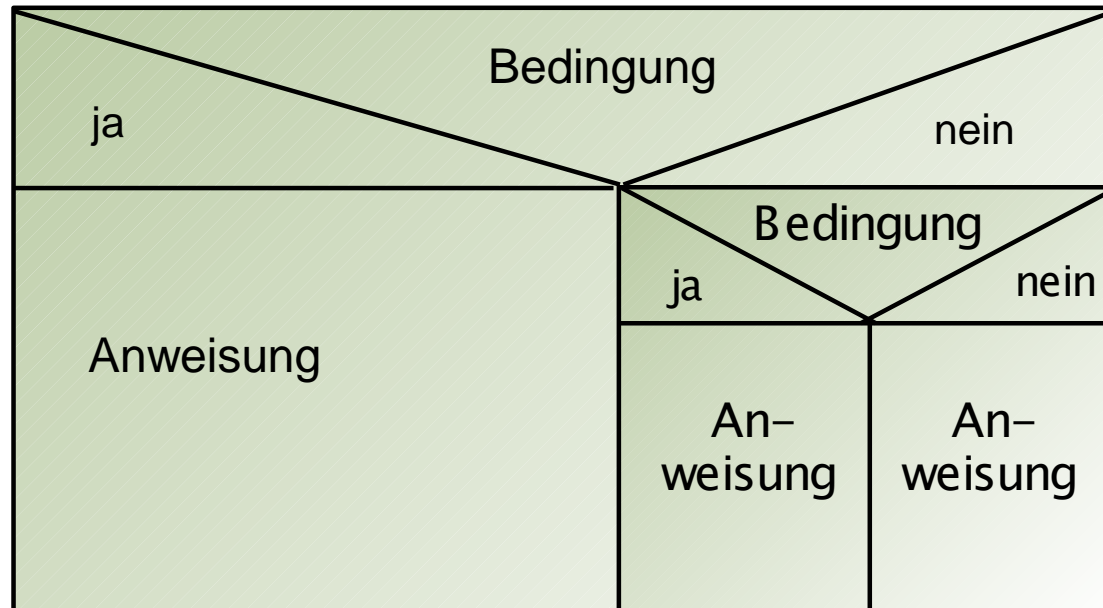
```
End If
```

```
Else  
    Anweisung
```

```
End If
```

```
Anweisung
```





```
Dim zahlA As Integer  
Dim zahlB As Integer  
Dim result As Integer  
  
result = zahlA MOD zahlB  
  
If (result = 0) Then  
    If (zahlA = zahlB) Then  
        Debug.Print "Die Zahlen sind gleich."  
    Else  
        Debug.Print "Division ohne Rest"  
    End If  
Else  
    Debug.Print "Rest der Division: " & result  
End If
```

- Jede Ebene einer verschachtelten if-Anweisung wird mit Hilfe des Tabulators eingerückt.
- Nutzen Sie nicht mehr als fünf Ebenen, um die Übersicht zu behalten.
- Vermeiden Sie unterschiedliche Datentypen auf beiden Seiten eines Vergleichsoperators.
- Bei Dezimalzahlen sollte ein Test auf Gleichheit vermieden werden.
- Verzichten Sie auf Elself-Anweisungen, wenn Sie switch-Anweisungen nutzen können.

```
Select Case variable  
  Case konstante  
    Anweisung  
  Case konstante  
    Anweisung  
  Case Else  
    Anweisung  
End Select  
  
Anweisung
```

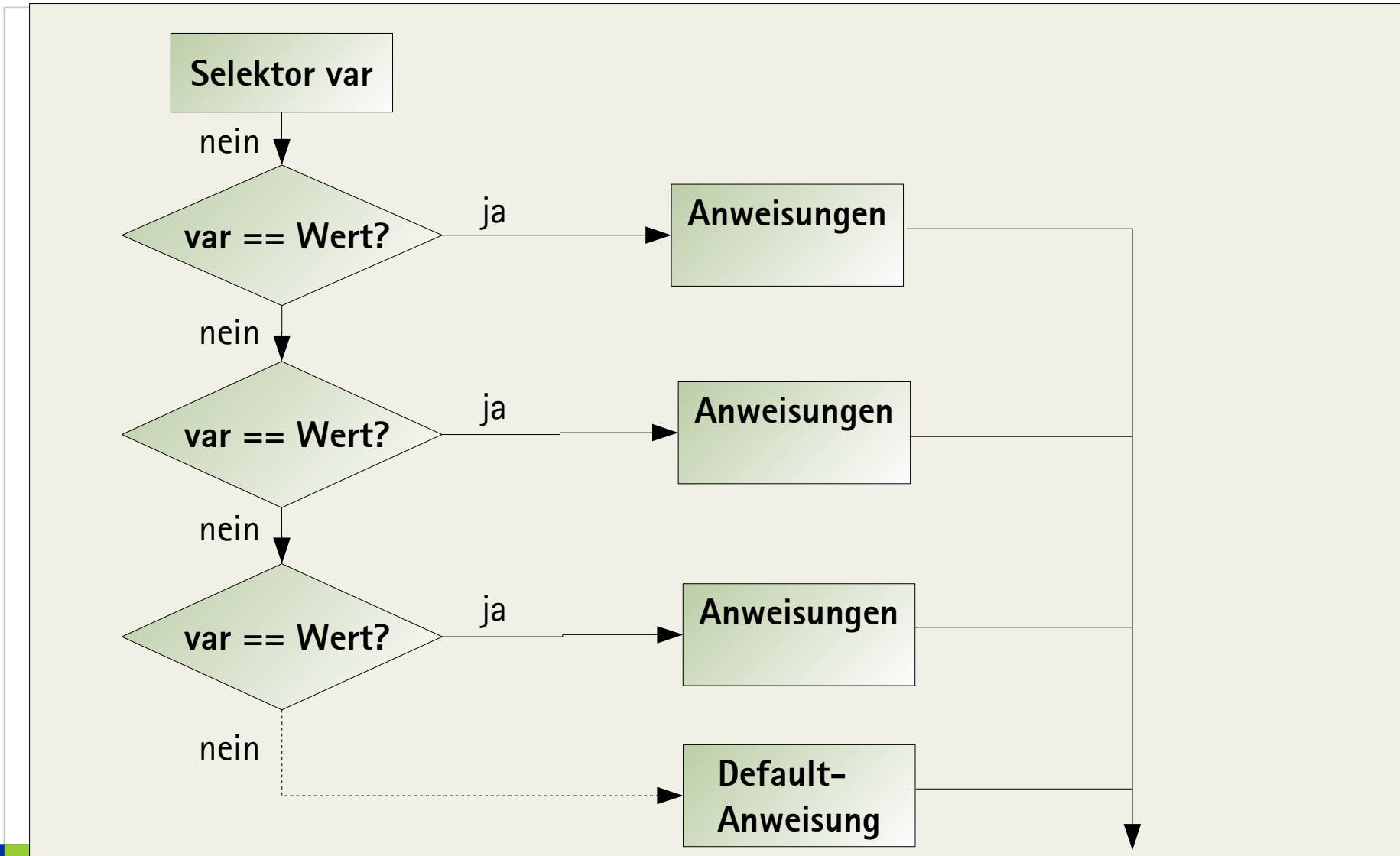
- ... sind eine andere Form von Elself-Anweisungen.
- Der Wert der *variablen* wird mit anderen Werten verglichen. Falls der Wert übereinstimmt, werden die nachfolgenden Anweisungen ausgeführt.
- Als Vergleichswerte können numerische Werte oder Strings genutzt werden.
- Der Standardfall *Case Else* muss nicht angegeben werden.

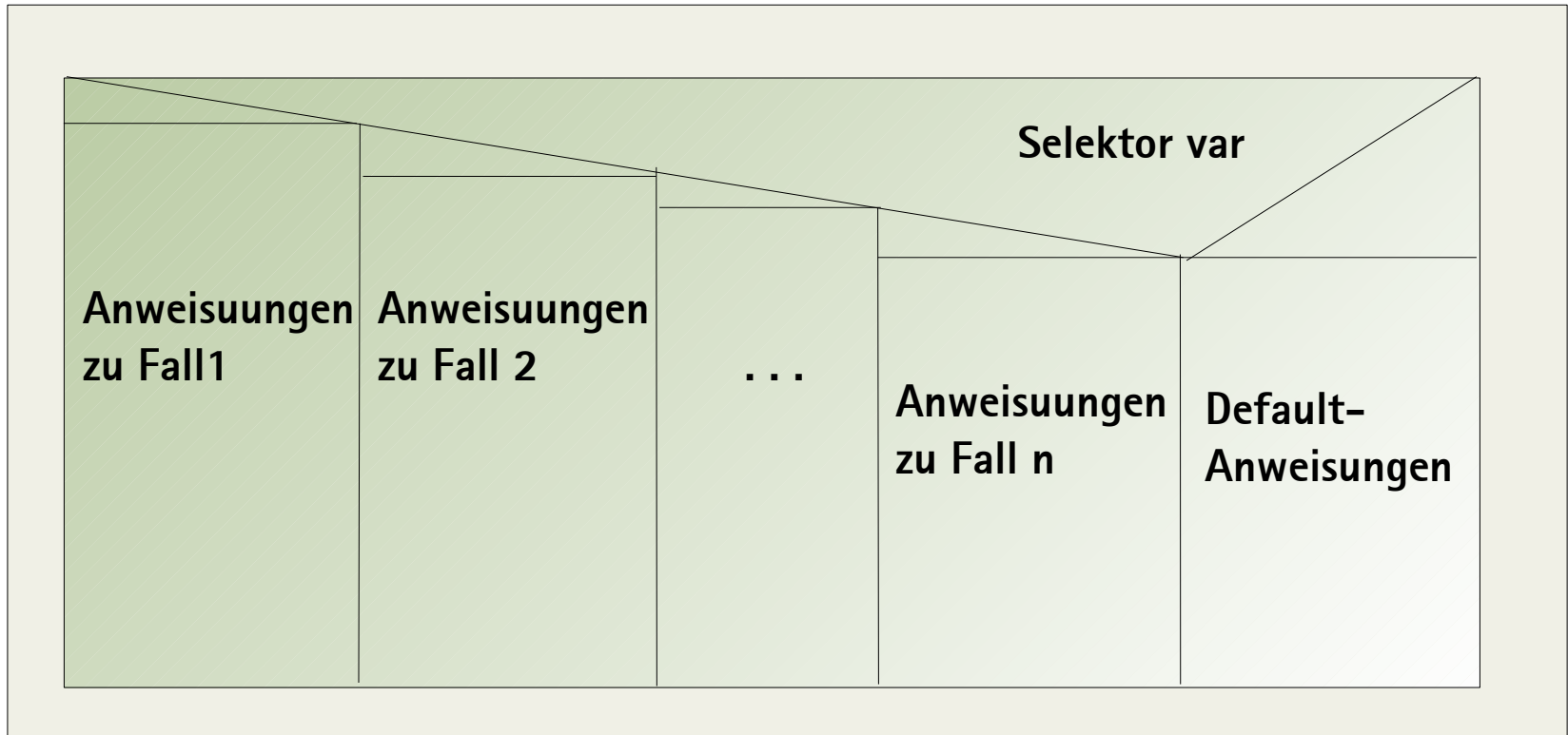
```
Select Case variable  
  Case konstante  
    Anweisung  
  Case konstante  
    Anweisung  
  Case Else  
    Anweisung  
End Select  
Anweisung
```

Selektiere die Variable und vergleiche den Wert der Variablen mit...

Falls einer der Fälle (Case) mit den Wert der zu überprüfenden Variablen übereinstimmt, führe nachfolgende Anweisungen aus.

Falls keiner der Fälle zutrifft, führe den Standardfall aus.





```
Dim wert As Integer
```

```
Select Case wert
```

```
Case 1
```

```
Debug.Print "Speichern"
```

```
Case 2
```

```
Debug.Print "Ausdruck"
```

```
Case Else
```

```
Debug.Print "Falsche Eingabe"
```

```
End Select
```

Die Variable und die zu vergleichenden Werte müssen vom gleichen Datentyp sein. Mit Hilfe von `Case` wird ein Vergleich auf Gleichheit implementiert.

```
Dim produkt As String  
Dim kategorie As String
```

```
Select Case produkt  
  Case "Apfel"  
    kategorie = "Obst"
```

```
  Case "Weißkohl"  
    kategorie = "Gemüse"
```

```
  Case Else  
    kategorie = ""  
End Select
```

```
Debug.Print kategorie
```

Sobald ein Wert mit dem Wert der Variablen übereinstimmt, werden die nachfolgenden Anweisungen abgearbeitet. Anschließend werden die Anweisungen nach dem Schlüsselwort *End Select* abgearbeitet.

```
Dim produkt As String  
Dim kategorie As String
```

```
Select Case produkt  
  Case "Apfel", "Bananen", "Birnen"  
    kategorie = "Obst"  
  
  Case "Weißkohl", "Bohnen", "Erbsen"  
    kategorie = "Gemüse"  
  
  Case Else  
    kategorie = ""  
  
End Select
```

Hier wird eine Liste von Strings genutzt. Die einzelnen Elemente werden durch ein Komma getrennt.

Sobald ein Element der Liste mit den Wert der Variablen übereinstimmt, werden die nachfolgenden Anweisungen ausgeführt. Hier wird eine ODER-Verknüpfung von Vergleichswerten implementiert.

```
Dim bestellmenge As Integer  
Dim rabatt As Single
```

```
Select Case bestellmenge  
Case 1 To 50
```

```
rabatt = 0.0
```

```
Case 51 To 100
```

```
rabatt = 0.03
```

```
Case Else
```

```
Debug.Print "Falsche Bestellmenge"
```

```
bestellmenge = 0
```

```
rabatt = 0.0
```

```
End Select
```

Hier muss der Wert der zu vergleichenden Variablen in einem bestimmten Zahlenbereich liegen.
Ein Bereich wird folgendermaßen angegeben: *Case min To max.*

```
Dim bestellmenge As Integer
```

```
Select Case bestellmenge
```

```
Case Is < 10
```

```
Debug.Print "Mindestbestellmenge unterschritten"
```

```
Case Is > 100
```

```
Debug.Print "Lieferung nur gegen Vorkasse"
```

```
Case Else
```

```
Debug.Print "Bestellung ausführen"
```

```
End Select
```

Das Schlüsselwort *Is* ist ein Platzhalter für die zu vergleichende Variable. Das Schlüsselwort muss genutzt werden, wenn nicht auf Gleichheit geprüft wird.

```
Dim bestellmenge As Integer  
Dim rabatt As Double
```

```
Select Case bestellmenge
```

```
  Case Is >= 50 And (bestellmenge < 100)  
    rabatt = 0.03
```

```
  Case Is > 100 And (bestellmenge < 200)  
    rabatt = 0.05
```

```
  Case Else  
    rabatt = 0.0
```

```
End Select
```

Mit Hilfe von logischen Operatoren können verschiedene Bedingungen verknüpft werden.

In der ersten Bedingung wird der zu untersuchende Variablenname durch das Schlüsselwort *Is* ersetzt. In allen nachfolgenden Bedingungen muss der Variablenname genutzt werden.

Wenn mit Dezimalzahlen gearbeitet wird, sollten *if-else*-Leitern genutzt werden.

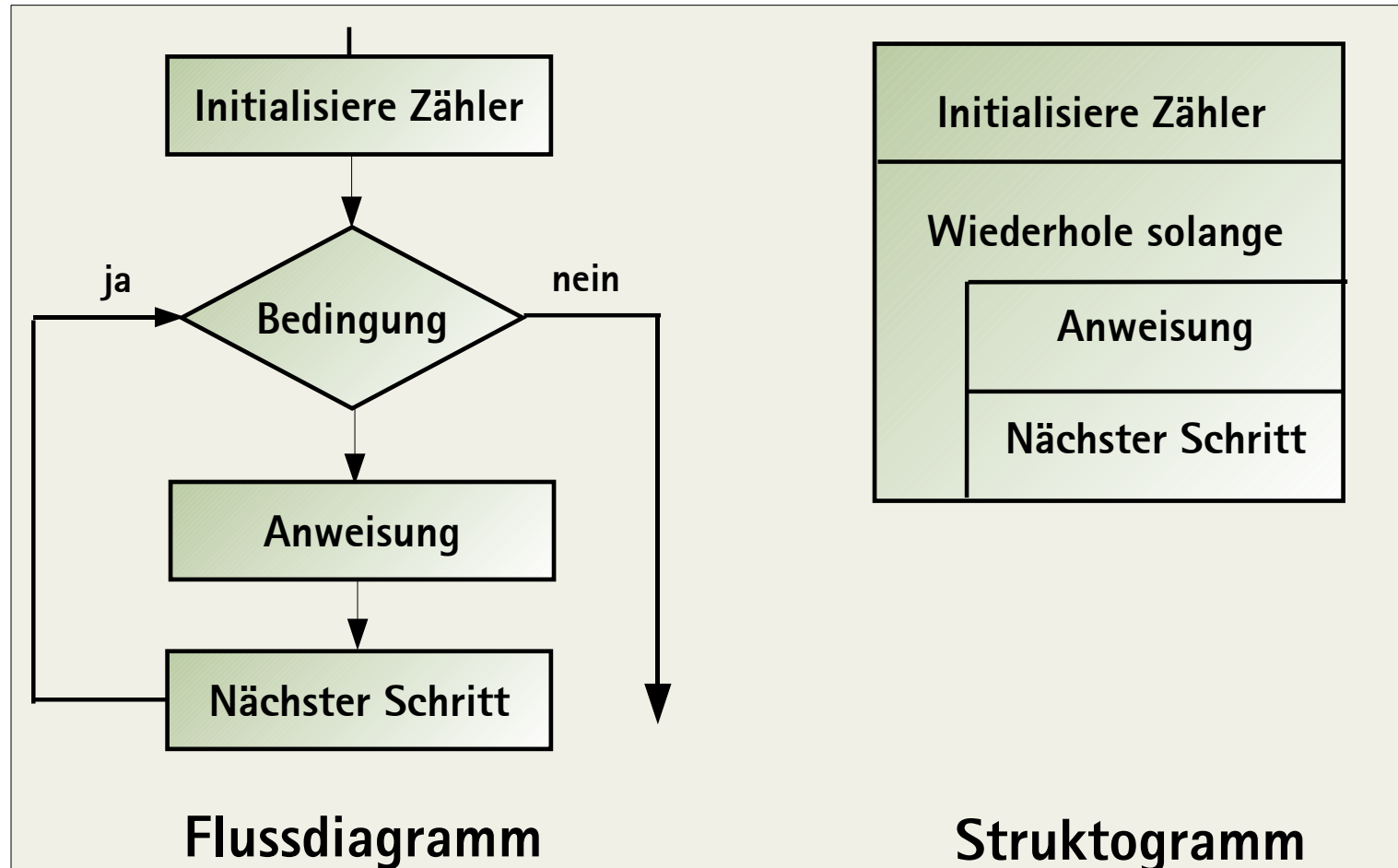
- ... sind Wiederholungen bestimmter Anweisungen.
- Die Anzahl der Durchläufe wird durch eine Bedingung bestimmt. Falls diese Bedingung nie wahr wird, wird die Schleife als Endlosschleife bezeichnet.
- ... können vorzeitig abgebrochen werden.
- Folgende Schleifen sind in VBA implementiert:
 - Für eine bestimmte Anzahl von Schleifendurchläufen wird die For-Next-Schleife genutzt. Diese Art von Schleife wird auch als Zählschleife bezeichnet.
 - Falls die Anzahl der Schleifendurchläufe nicht bekannt ist, können Do-Loop- oder While-Wend-Schleifen genutzt werden. Für den Schleifenabbruch wird eine Bedingung angegeben.

***For zaehler = startwert To endwert
Anweisung
Next zaehler***

- Die Zählschleife beginnt mit dem Schlüsselwort *For*.
- Der Variablen *zaehler* wird ein Startwert zugewiesen. Die Variable wird mit diesen Wert initialisiert.
- Die Schleife läuft solange bis der Endwert erreicht ist.
- Start- und Endwert entsprechen dem Datentyp der Variablen.
- Das Schlüsselwort setzt *zaehler* auf den nächsten Wert.

***For zaehler = startwert To endwert Step schrittweite
Anweisung
Next zaehler***

- *zaehler* wird mit Hilfe der Anweisung *Next* immer um *schrittweite* erhöht oder vermindert.
- Die Angabe der Schrittweite sollte dem Datentyp der Zählvariablen entsprechen. Möglichkeiten: Ganz- oder Dezimalzahlen.
- Falls Sie einen negativen Wert für die Schrittweite angeben, wird die Zählvariable um die Angabe vermindert. Falls Sie einen positiven Wert angeben haben, wird die Zählvariable um die Angabe erhöht.



```
Dim count As Integer  
Dim zaehler As Double;
```

```
For count = 0 To 9  
  Debug.Print count  
Next count
```

```
For count = 9 To 0  
  Debug.Print count  
Next count
```

```
For count = 9 To 0 Step - 1  
  Debug.Print count  
Next count
```

In dieser Zählschleife wird von 0 bis 9 hochgezählt. Die Schrittweite beträgt standardmäßig eins.

Diese Schleife wird nicht durchlaufen, weil der Startwert größer als der Endwert ist.

In dieser Zählschleife wird von 9 bis 0 runtergezählt. Hier ist eine Schrittweite von minus eins angegeben.

```
Dim count As Integer  
Dim zaehler As Double;
```

In dieser Zählschleife wird von 0 bis 6 hochgezählt.
Die Schrittweite beträgt standardmäßig eins.

```
For count = 0 To 5.5  
Debug.Print count  
Next count
```

In dieser Zählschleife wird von 0 bis 5 hochgezählt.
Die Schrittweite beträgt standardmäßig eins.

```
For zaehler = 0 To 5.5  
Debug.Print count  
Next count
```

```
For count = 0 To 5.5 Step 0.5  
Debug.Print count  
Next count
```

Die Schleife läuft endlos. Die Angabe der Schrittweite passt nicht zum Datentyp des Zählers.

```
For zaehler = 9 To 0 Step - 0.5  
Debug.Print count  
Next count
```

Hier wird der Zähler von 9 beginnend immer um 0.5 minimiert.

```
Dim wert As Integer  
Dim faktor As Double;  
Dim ergebnis As Double
```

```
For wert = 1 To 5  
  ergebnis = 0
```

```
  For faktor = 0.5 To 2 Step 0.5  
    ergebnis = faktor * wert
```

```
    If (faktor > 1.5) AND (wert = 2) Then  
      Exit For
```

```
    End If  
  Next faktor
```

```
  Debug.Print ergebnis  
Next wert
```

In diesem Beispiel werden zwei verschachtelte For-Next-Schleifen für eine Berechnung genutzt. Mit Hilfe von *Exit* wird die innere Schleife abgebrochen. Die Anweisungen der äußeren Schleife werden aber vollständig ausgeführt.

```
Do  
  Anweisung  
Loop
```

- Die Schleife beginnt mit *Do* und endet mit *Loop*.
- Momentan würde diese Schleife endlos laufen. Es fehlt eine Abbruchbedingung.

Do
Anweisung
Loop Until Bedingung

Do
Anweisung
Loop While Bedingung

- Die Abbruchbedingung steht im Fuß der Schleife. Das heißt, die Schleife wird mindestens einmal durchlaufen.
- *Until* sagt aus, dass die Schleife solange läuft bis die Bedingung erfüllt ist. Sobald die Bedingung erfüllt ist, wird die Schleife abgebrochen.
- *While* sagt aus, dass die Schleife solange läuft wie die Bedingung erfüllt ist. Sobald die Bedingung nicht erfüllt ist, wird die Schleife abgebrochen.

```
Dim count As Integer
```

```
count = 0
```

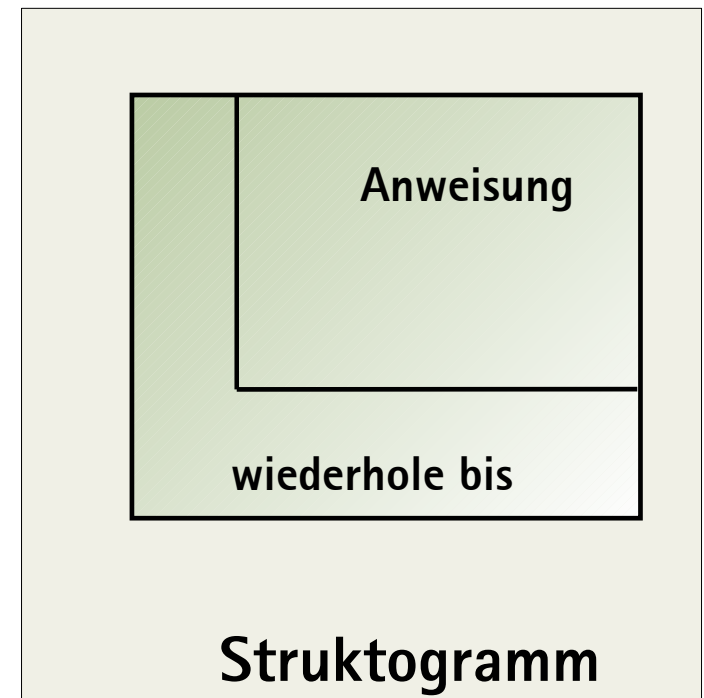
```
Do
```

```
count = count + 1
```

```
Debug.Print count
```

```
Loop Until count > 10
```

In dieser Schleife wird *count* bei jedem Durchlauf um eins hochgezählt. Die Schleife bricht ab, sobald *count* den Wert 11 besitzt.



```
Dim count As Integer
```

```
count = 0
```

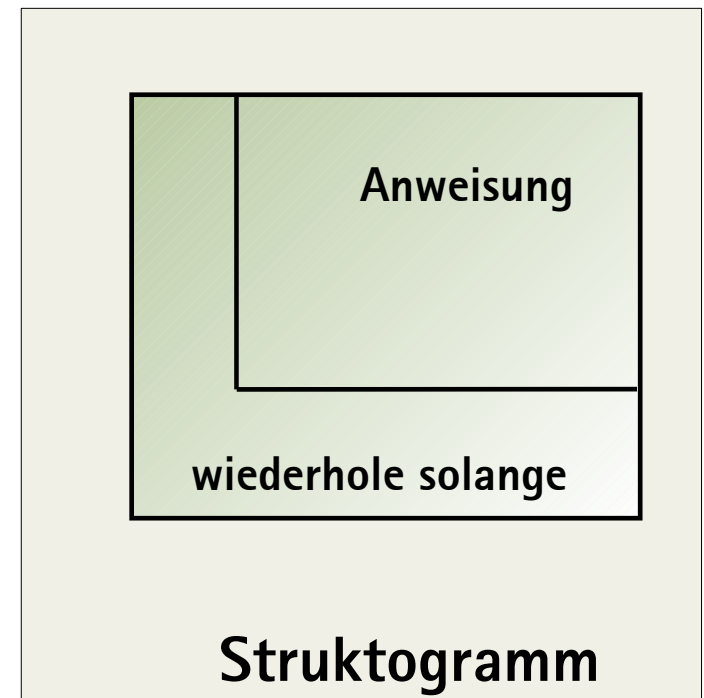
```
Do
```

```
count = count + 1
```

```
Debug.Print count
```

```
Loop While count > 10
```

Diese Schleife wird exakt einmal durchlaufen. Die Bedingung ist nicht erfüllt, weil *count* kleiner als 10 ist.



***Do Until Bedingung
Anweisung
Loop***

***Do While Bedingung
Anweisung
Loop***

- Die Abbruchbedingung steht im Kopf der Schleife. Am Anfang wird die Bedingung überprüft. Falls die Bedingung nicht erfüllt ist, wird die Schleife nicht durchlaufen.
- *Until* sagt aus, dass die Schleife solange läuft bis die Bedingung erfüllt ist. Sobald die Bedingung erfüllt ist, wird die Schleife abgebrochen.
- *While* sagt aus, dass die Schleife solange läuft wie die Bedingung erfüllt ist. Sobald die Bedingung nicht erfüllt ist, wird die Schleife abgebrochen.

```
Dim count As Integer
```

```
count = 0
```

```
Do Until count > 10
```

```
count = count + 1
```

```
Debug.Print count
```

```
Loop
```

In dieser Schleife wird *count* bei jedem Durchlauf um eins hochgezählt. Die Schleife bricht ab, sobald *count* den Wert 11 besitzt.

Wiederhole bis

Anweisung

Struktogramm

```
Dim count As Integer
```

```
count = 0
```

```
Do While count > 10
```

```
count = count + 1
```

```
Debug.Print count
```

```
Loop
```

Diese Schleife wird nicht durchlaufen. Die Bedingung ist nicht erfüllt, weil *count* kleiner als 10 ist.

Wiederhole solange

Anweisung

Struktogramm

```
Dim wert As Integer  
Dim faktor As Double;  
Dim ergebnis As Double
```

```
wert = 0
```

```
Do
```

```
  ergebnis = 0
```

```
  faktor = 0
```

```
  Do While faktor < 2.0
```

```
    ergebnis = ergebnis + (faktor * wert)
```

```
    faktor = faktor + 0.5
```

```
  If (faktor > 1.5) AND (wert = 2) Then
```

```
    Exit Do
```

```
  End If
```

```
Next faktor
```

```
Debug.Print ergebnis
```

```
wert = wert + 1
```

```
Loop Until wert > 5
```

In diesem Beispiel werden zwei verschachtelte Do-Loop-Schleifen für eine Berechnung genutzt. Mit Hilfe von *Exit* wird die innere Schleife abgebrochen. Die Anweisungen der äußeren Schleife werden aber vollständig ausgeführt.

- Syntaxfehler entstehen beim Schreiben des Programmcodes.
- Logische Fehler treten auf, wenn der Entwickler ein Denkfehler bei der Umsetzung der Aufgabe in ein Programm macht. Das Programm wird fehlerfrei ausgeführt, aber das Ergebnis ist nicht korrekt.
- Laufzeitfehler treten während der Ausführung des Programms auf. Zum Beispiel eine CD, von der Daten gelesen werden sollen, wurde nicht in das Laufwerk gelegt.

- Die Syntax einer Programmiersprache ist die Gesamtheit der Regeln für die Bildung von Anweisungen aus Operatoren und Operanden sowie die Nutzung von Funktionen.
- ... sind Tippfehler bei der Eingabe von Variablennamen oder Schlüsselwörtern.
- ... entstehen, wenn Argumente an Funktionen falsch übergeben werden.

- Mehrfach verwendete Bezeichner.
- Zeichensetzungsfehler wie zum Beispiel ein fehlendes Komma als Trennzeichen zwischen Listenelementen oder Parametern.
- Fehlende Schlüsselwörter in einer Schleife oder bedingten Anweisung.
- Unzulässige Mischung von numerischen und nicht-numerischen Operanden.

- So bald die Zeile gewechselt wird und der Code verändert wurde, wird ein Syntaxfehler farbig dargestellt.
- Aktivieren Sie die Option Automatische Syntaxüberprüfung auf der Registerkarte Editor im Optionen-Fenster (Extras - Optionen).
- Die Anweisung *Option Explicit* vermeidet eine doppelte Vergabe von Variablennamen.

- ... sind Bugs, die nach dem Start eines Programms auftreten können.
- ... betreffen immer die Programmlogik.
- ... können Programme zu einem unerwünschten Verhalten oder einen Programmabsturz zwingen.
- ... entstehen, wenn Ausdrücke oder Anweisungen vom Programm nicht korrekt ausgewertet werden.

- Division durch Null.
- Überlauf (zu großer oder zu kleiner Wert für den angegebenen Datentyp).
- Falsche Abbruchbedingung für eine Schleife.
- Verwendung von ungültigen Operatoren
- Ein- und Ausgabefehler.
- Tippfehler wie "1o" statt 10.

```
Const zahlA = 4  
Const zahlB = 0  
Dim ergebnis As Integer  
  
ergebnis = zahlA / zahlB
```

- Das Objekt *Err* bekommt alle Fehler eines VBA-Programms übergeben und kann ausgelesen werden.
 - *Err.Number* gibt eine Fehlernummer aus.
 - *Err.Description* gibt einen Fehlertext aus.
 - *Err.Helpfile* gibt einen Hilfetext zu einem Fehler aus.
 - *Err.Clear* setzt alle Eigenschaften auf die Standardwerte zurück. Es werden alle vorhandenen Fehlermeldungen gelöscht.

- Die Anweisung *On Error* regelt das Verhalten des Programms bei einem Laufzeitfehler.
- *On Error Resume 0* oder *On Error Resume* wiederholt die fehlerbehaftete Zeile.
- *On Error Resume Next* überspringt die fehlerbehaftete Zeile und arbeitet mit der nächsten Programmzeile weiter. Nachteil: Fehlerbehaftete Variablen werden weiter durch das Programm geschleppt.
- *On Error Goto Marke* springt, sobald ein Fehler auftritt, zu einer bestimmten Position im Programm. An dieser Position wird der Fehler behandelt.
- *On Error Goto 0* deaktiviert alle selbst definierten Fehlerbehandlungsroutinen. Es werden die Standard-Fehlerdialoge der Access-Datenbank genutzt.

```
Const zahlA = 4  
Const zahlB = 0  
Dim ergebnis As Integer
```

Alle vorhandenen
Fehlermeldungen werden
gelöscht.

```
Err.Clear
```

```
On Error Goto FEHLER
```

Falls ein Fehler in den
nachfolgenden Anweisungen
auftritt, wird zu der Marke
FEHLER gesprungen.

```
ergebnis = zahlA / zahlB
```

```
Exit Sub
```

Hier wird die Prozedur
verlassen. Andernfalls werden
alle Anweisungen, die zur
Sprungmarke gehören,
abgearbeitet.

```
FEHLER:  
MsgBox(Err.Description)
```

```
Const zahlA = 4  
Const zahlB = 0  
Dim ergebnis As Integer
```

```
Err.Clear
```

```
On Error Goto FEHLER
```

```
ergebnis = zahlA / zahlB
```

```
Exit Sub
```

```
FEHLER:  
MsgBox(Err.Description)
```

Hier steht der Name der Sprungmarke. Der Bezeichnung folgt immer ein Doppelpunkt. Alle nachfolgenden Anweisungen werden ausgeführt unabhängig davon ob sie zur Sprungmarke gehören oder nicht.

Falls ein Fehler auftritt, wird die dazugehörige Fehlermeldung ausgegeben.

- Logische Fehler entstehen
 - ... beim Design eines Programms oder
 - ... bei der Definition von Anforderungen an das Programm.
- Logische Fehler können
 - ... durch fehlendes Fachwissen oder
 - ... Missverständnissen zwischen Nutzern und Entwicklern entstehen.
- Logische Fehler können nicht mit Hilfe von Programmstrukturen abgefangen werden.

- ... eine falsche Anzahl von Schleifendurchläufen.
- ... durch das Erzeugen von Endlosschleifen.
- ... falsch formulierte Bedingungen in Anweisungen und Schleifen.
- ... eine falsche oder nicht vorhandene Klammerung von komplexen Ausdrücken.
- ... falsch initialisierte Variablen.

- Das Programm muss Zeile für Zeile durchlaufen werden. Sehr zeitaufwendig.
- VBA bietet die Möglichkeit
 - ... das Programm im Einzelschrittmodus zu durchlaufen.
 - ... Zwischenergebnisse in das Direktfenster zu drucken.
 - ... Variablen zu überwachen.

- Im Einzelschrittmodus wird das Programm Zeile für Zeile abgearbeitet.
- ... wird mit Hilfe des Menüs Debuggen – Einzelschritt oder <F8> gestartet.
- Jede Zeile muss mit der Taste <F8> quittiert werden.
- Die aktuelle Zeile wird gelb unterlegt und mit Hilfe eines gelben Pfeils am linken Rand gekennzeichnet. Der Pfeil kann mit Hilfe von Drag & Drop verschoben werden, um zum Beispiel vorhergehende Anweisungen erneut auszuführen.
- Wenn der Mauszeiger über eine Variable oder Ausdruck liegt, wird der aktuelle Wert in einem gelben Erklärfenster angezeigt.

```
Sub openKunde()  
    Dim heute As Date  
    On Error GoTo Err_NotOpen  
    DoCmd.OpenForm "frmKunde", acNormal, , , acFormReadOnly, acDialog  
    Exit Sub  
Err_NotOpen:  
    MsgBox Err.Description  
End Sub
```

- Das Programm kann mit <F5> oder im Einzelschrittmodus gestartet werden.
- Falls eine Programmzeile mit einem Haltepunkt versehen ist, wird das Programm dort unterbrochen.
- Das Programm kann anschließend wieder
 - ... mit <F5> vollständig bis zum Ende abgearbeitet werden.
 - ... im Einzelschrittmodus bis zum Ende oder des nächsten Haltepunktes durchlaufen werden.
- Eine Programmzeile mit einem Haltepunkt wird braun hinterlegt und mit einem braunen Punkt am linken Rand gekennzeichnet.
- Auf Deklarationsanweisungen, leeren Zeilen, Sprunganweisungen oder Kommentaren kann kein Haltepunkt gesetzt werden!

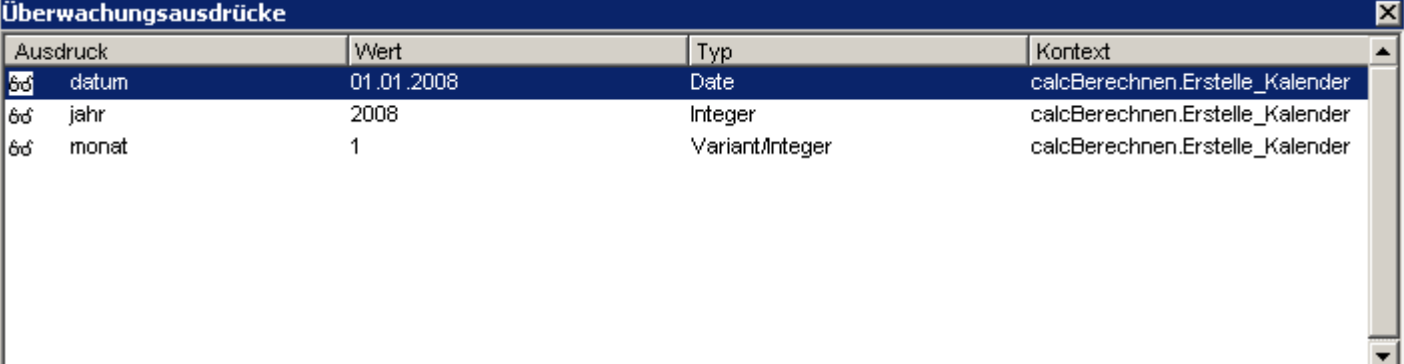
```
count = 1
For monat = FirstMonth To LastMonth
    spalte = 2 * count
    zeile = 3
    datum = DateSerial(jahr, monat, 1)
    index = Weekday(datum, vbMonday)
    LastDay = Day(DateSerial(jahr, monat + 1, 1) - 1)
    comment = ""
```

- Haltepunkte setzen:
 - Der Mauszeiger befindet sich in der Zeile, in der ein Haltepunkt gewünscht ist.
 - Der Haltepunkt wird dann mit <F9> oder Debuggen – Haltepunkte ein / aus gesetzt.
- Haltepunkte löschen:
 - Der Mauszeiger befindet sich in einer Zeile mit Haltepunkt.
 - Der Haltepunkt wird dann mit <F9> oder Debuggen – Haltepunkte ein / aus gelöscht.
 - Andere Möglichkeit: Mausklick auf den braunen Punkt am linken Rand.
- Alle Haltepunkte löschen:
 - <STRG>+<UMSCHALT>+<F9> löscht alle Haltepunkte in einem Programm.

- Das Programm wird im Einzelschrittmodus durchlaufen.
- Markieren Sie die gewünschte Variable.
- Klicken Sie auf Debuggen – Aktueller Wert anzeigen oder <UMSCHALT>+<F9>.
- Im sich öffnenden Dialogfenster
 - ... wird die Funktion angezeigt, in der die Variable vorkommt.
 - ... wird der Name der Variablen angezeigt.
 - ... wird der aktuelle Wert der Variablen angezeigt.
 - ... kann mit Hilfe der Schaltfläche Hinzufügen die Variable in das Überwachungsfenster übernommen werden.



- ... wird mit Hilfe von Ansicht – Überwachungsfenster oder automatisch beim Hinzufügen einer Variablen geöffnet.
- Pro Zeile wird ein Ausdruck, eine Variable oder eine Funktion angezeigt.



Ausdruck	Wert	Typ	Kontext
öö datum	01.01.2008	Date	calcBerechnen.Erstelle_Kalender
öö jahr	2008	Integer	calcBerechnen.Erstelle_Kalender
öö monat	1	Variant/Integer	calcBerechnen.Erstelle_Kalender

Ausdruck	Wert	Typ	Kontext
datum	01.01.2008	Date	calcBerechnen.Erstelle_Kalender
jahr	2008	Integer	calcBerechnen.Erstelle_Kalender
monat	1	Variant/Integer	calcBerechnen.Erstelle_Kalender

Bezeichnung
der Variablen,
Funktion oder
Ausdruck.

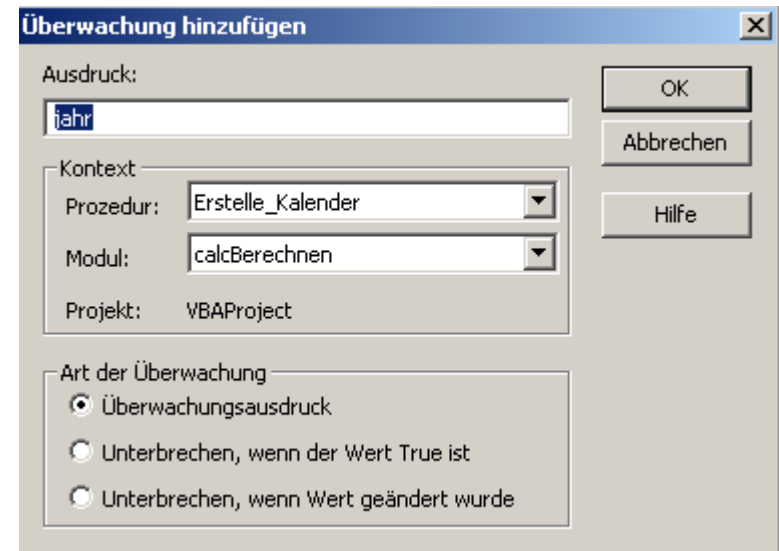
Aktueller Wert
des Ausdrucks.
Rückgabewert
der Funktion.

Datentyp des
Ausdrucks.

In welcher Prozedur
befindet sich der
Ausdruck?

- Markieren Sie den zu überwachenden Ausdruck im Programmcode und ziehen diesen mit der Maus in das geöffnete Überwachungsfenster.
- Markieren Sie eine Zeile im Überwachungsfenster mit der Maus und klicken Sie auf <ENTF>, um einen Ausdruck aus der Überwachung zu entfernen.
- Falls das Programm im Einzelschrittmodus abgearbeitet wird, markieren Sie den Wert einer Variablen im Überwachungsfenster. Der markierte Wert kann mit Hilfe der Tastatur durch einen neuen ersetzt werden.

- Mit Hilfe von Debuggen – Überwachung bearbeiten oder <STRG>+<W> wird ein Fenster geöffnet, in dem die Art der Überwachung eingestellt werden kann.
- Überwachungsausdruck wird bei der Überwachung von mehreren Variablen im Einzelschrittmodus genutzt.
- Unterbrechen, wenn der Wert True ist unterbricht das Programm, so bald deren Wert wahr wird.
- Unterbrechen, wenn der Wert geändert wurde unterbricht das Programm, so bald sich der Wert der überwachten Variablen verändert.



- Öffnen Sie das Direktfenster mit Hilfe des Menüs Ansicht – Direktfenster.
- Im Direktfenster kann der Ablauf eines Programms mit Hilfe der Anweisung *Debug.Print* in einem gewissen Umfang protokolliert werden.
- Falls das Programm im Einzelschrittmodus durchlaufen wird, können Variablen und deren Werte überprüft werden.
- Mit Hilfe von <STRG>+<A> wird der gesamte Inhalt des Direktfensters markiert.
- Mit Hilfe der Taste <ENTF> kann der markierte Inhalt gelöscht werden.

- Die Anweisung *Debug.Print variable* im Programmcode schreibt den Wert einer Variablen in das Direktfenster.
- Sie können auch direkt im Direktfenster arbeiten.
 - Klicken Sie mit dem Mauszeiger auf eine freie Stelle im Fenster.
 - Die Eingabe *?Variable* gibt den Wert einer Variablen aus, sobald Sie die Anweisung mit <ENTER> abgeschlossen haben.
 - Die Anweisung *variable = 20* verändert den Wert der Variablen.
 - Die Anweisung *?Left(zeichen, 4)* führt die Funktion *Left* aus und das Ergebnis wird in der nächsten Zeile ausgegeben. Im Direktfenster können Sie vordefinierte Funktionen ausprobieren, um ihre Arbeitsweise zu verstehen.

- ... sind zusammengesetzte Datentypen
- ... bestehen aus mehreren Elementen, die vom gleichen Datentyp sind.
- Eine Gruppe von Variablen, die den gleichen Datentyp besitzen, werden gemeinsam verwaltet.
- ... sind aufeinander gestapelte Behälter gleicher Größe, aber unterschiedlichen Inhalts.
- Konstante Felder können in VBA nicht implementiert werden.
- In Feldern werden zum Beispiel die Namen der Monate oder eine bestimmte Kategorie von Objekten gespeichert.

Dim array([Min To] max) As Datentyp

Dim tag As String

Dim monat(1 To 12) As String

Dim messwerte(12) As Double

- Ein Array beginnt genauso wie ein Variable mit dem Schlüsselwort *Dim*.
- Ein Array wird mit Hilfe von *As* ein Datentyp zugewiesen. Jedes Element in diesem Array hat den gleichen Bauplan.
- Jedes Array hat einen eindeutigen Namen mit dem es aufgerufen wird.
- Im Anschluss an die Bezeichnung folgen runden Klammern. In den runden Klammern
 - ... steht einer Obergrenze. Die Anzahl der Elemente berechnet sich aus der Standarduntergrenze und der Obergrenze.
 - ... steht eine Untergrenze und Obergrenze. Die Anzahl der Elemente berechnet sich aus den Angaben.



Dim tag(1 To 7) As String



Dim tag(6) As String

- Falls keine Untergrenze angegeben ist, wird eine Untergrenze von Null genutzt. Das heißt, das erste Element hat den Index 0 und das letzte Element als Index die angegebene Obergrenze.
- Am Anfang eines Moduls kann die Anweisung *Option Base 1* geschrieben werden. Die Untergrenze und damit der Index des ersten Elements wird auf eins gesetzt. Einige, in VBA integrierte Funktionen, beachten aber diese Anweisung nicht.
- Die Standard-Untergrenze wird außer Kraft gesetzt, wenn in der Deklaration ein Wert für die Untergrenze angegeben wird.



tag(3) = Sonntag

Dim tag(1 To 7) As String

Jedes Element hat ein Index.
Der Index des ersten Elements wird durch die Untergrenze festgelegt. Das nachfolgende Element hat einen Index + 1.
Der Index des letzten Element wird durch die Obergrenze festgelegt.
Der Index wird in runden Klammern direkt im Anschluss an die Bezeichnung angegeben.
Ein Index wird immer als Ganzzahl angegeben.

```
For count = LBound(tag) To UBound(tag)  
    Debug.Print tag(count)  
Next Count
```

```
Dim element As Variant  
For Each element In tag  
    Debug.Print element  
Next element
```

- Ein Array kann vollständig mit Hilfe einer for-next-Schleife durchlaufen werden.
 - Der Index beginnt bei der mit Hilfe von *LBound()* ermittelten Untergrenze.
 - Sobald die Obergrenze (*UBound()*) erreicht ist, wird die Schleife abgebrochen.
- Ein Array kann mit Hilfe einer for each-Schleife durchlaufen werden.
 - Die Variable *element* ist ein Symbol für jedes Listenelement und muss als *Variant* deklariert werden.

```
Dim tag(0 To 6) As String  
Dim anzahl As Integer
```

```
anzahl = UBound(tag) – LBound(tag) + 1
```

- Für die Berechnung gibt es keine vordefinierte Funktion.
- Es kann nur folgende Berechnung genutzt werden:

$$\begin{array}{rclcl} \text{UBound(Tag)} - \text{LBound(tag)} + 1 & & & & \\ 6 & - & 0 & + & 1 & = & 7 \\ \text{Obergrenze} & - & \text{Untergrenze} & + & 1 & & \end{array}$$

```
Dim feld(4) As String  
Dim feldFilter() As String  
Dim element As Variant
```

```
feld(0) = "Northeim"  
feld(1) = "Göttingen"  
feld(2) = "Hannover"  
feld(3) = "Hildesheim"
```

```
feldFilter = Filter(feld, "Hannover", True)
```

```
For Each element In feldFilter  
    Debug.Print element  
Next element
```

Hier wird ein Array nach einem bestimmten Wort durchsucht. Der Funktion *filter()* wird als Parameter ein Feld übergeben, welches durchsucht wird. Als zweiter Parameter wird ein Suchwort übergeben. In dem Suchwort können Platzhalter nicht genutzt werden.

True sagt aus, dass alle Elemente, die dem Suchwort entsprechen, in das Array *feldfilter* geschrieben werden. Bei einem Wert *False* würden alle Elemente übernommen, die nicht dem Suchwort entsprechen.

- ... werden für die Darstellung von Matrizen oder Koordinatensystem genutzt.
- ... können eine Tabelle, die aus Zeilen und Spalten besteht, abbilden.
- In VBA können maximal 60 Dimensionen genutzt werden.

- Deklaration von mehrdimensionalen Arrays:

Dim arrayname (min To max, ... , min To max) As Datentyp

- Deklaration eines zweidimensionalen Arrays:

Dim matrix(0 To 4, 0 To 5) As Integer

- Die erste Dimension hat eine Untergrenze von null und eine Obergrenze von vier.
- Die zweite Dimension hat eine Untergrenze von null und eine Obergrenze von fünf.
- Wenn man ein zweidimensionales Array als Tabelle abbildet, gibt die erste Dimension die Spalten und die zweite Dimension die Zeilen an.

- Deklaration eines dreidimensionalen Arrays:

Dim matrix(0 To 4, 0 To 5, 0 To 2) As Integer

- Die erste Dimension entspricht der x-Achse in einem Koordinatensystem.
- Die zweite Dimension entspricht der y-Achse in einem Koordinatensystem.
- Die dritte Dimension entspricht der z-Achse in einem Koordinatensystem.



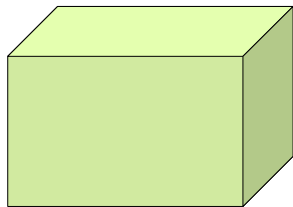
0-Dimension:
Ein Punkt beschreibt eine Koordinate.



1-Dimension:
Ein Linie wird durch einen Startpunkt beschrieben
und den Abstand von diesen Startpunkt.
Es werden unendlich viele Punkte benötigt.



2-Dimension:
Ein Rechteck (Fläche) wird durch seine Breite,
und Länge beschrieben.
Es werden x-, und y-Koordinaten benötigt.



3-Dimension:
Ein Quader (Körper) wird durch seine Breite,
Länge und Höhe beschrieben.
Es werden x-, y- und z-Koordinaten benötigt.

```
Dim rechteck(1 To 4, 1 To 5) As Double
```

```
Dim breite As Integer
```

```
Dim laenge As Integer
```

```
For breite = LBound(rechteck, 1) To UBound(rechteck, 1)
```

```
    For laenge = LBound(rechteck, 2) To UBound(rechteck, 2)
```

```
        rechteck(breite, laenge) = (breite * laenge) + 0.1
```

```
    Next laenge
```

```
Next breite
```

- Mit Hilfe einer verschachtelten for-Schleife wird das zweidimensionale Array gefüllt.
- In den runden Klammern hinter dem Feldnamen stehen für jede Dimension ein Index, getrennt durch ein Komma. Die erste Dimension hat den Index 1, usw.
- In den Funktion LBound() und UBound() muss die Dimension angegeben werden. Andernfalls werden die Grenzen der ersten Dimension abgefragt.

- ... haben keine festgelegten Grenzen für die verschiedenen Dimensionen.
- Die Anzahl der Fehler ist nicht statisch, sondern wird nach Bedarf erhöht oder vermindert.
- Deklaration: *Dim feldName() As Datentyp*
 - Die runden Klammern bleiben leer.
 - Die Anzahl der Elemente wird zur Laufzeit je nach Bedarf festgelegt.

```
Dim messwerte() As Double  
Dim max As Integer
```

```
ReDim messwerte(0 To 0)
```

```
max = UBound(messwerte)  
ReDim messwerte(0 To (max + 1))
```

- Mit Hilfe von *ReDim* wird zur Laufzeit die Dimension eines Feldes festgelegt. In der ersten Zeile wird mit Hilfe von *ReDim* ein eindimensionales Feld mit einem Element erstellt.
- Mit Hilfe der dritten Anweisung wird die Anzahl der Elemente des Feldes auf zwei erhöht. Das Feld wird gelöscht und neu dimensioniert. Der Inhalt der Felder wird automatisch gelöscht.
- Der Datentyp des Feldes kann nicht verändert werden.

```
Dim messwerte() As Double  
Dim max As Integer
```

```
ReDim messwerte(0 To 0)
```

```
max = UBound(messwerte)
```

```
ReDim Preserve messwerte(0 To (max + 1))
```

- Mit Hilfe von *ReDim* wird zur Laufzeit die Dimension eines Feldes festgelegt. In der ersten Zeile wird mit Hilfe von *ReDim* ein eindimensionales Feld mit einem Element erstellt.
- Mit Hilfe der dritten Anweisung wird die Anzahl der Elemente des Feldes auf zwei erhöht. Durch die Anweisung *ReDim Preserve* wird das Feld um ein Element erweitert. Der Feldinhalt des ersten Elements wird nicht gelöscht.
- Falls die Anzahl der Felder vermindert wird, geht die, in den Feldern, gespeicherte Information verloren. Es werden Felder und deren Inhalt gelöscht.

```
Const anzahlTage = 31  
Const min = 0
```

```
Dim sonnenSchein() As Single  
Dim tage As Integer  
Dim monat As Integer  
Dim anzahlMonat As Integer  
Dim wert As Single
```

```
For tage = 0 To anzahlTage  
  For monat = 0 To anzahlMonat  
    Redim Preserve sonnenSchein(monat, min To anzahlTage)  
    sonnenSchein(tage, monat) = wert  
  Next monat  
Next tage
```

Bei einem mehrdimensionalen Feld darf immer nur die letzte Dimension dynamisch verändert werden.