# Sprachelemente von VB: Operationen

Prof. Dr. Aris Christidis

- Zuweisung:

  **=**

- Grundrechenarten:

  **+**, **-**, *****, **/**,

- Erweiterte Grundrechenarten:

  **** (ganzzahlige Division), **Mod** (Div.-Rest), **^** (Potenzieren)

- Logische Verknüpfungen:

  **Not**, **And**, **Or**, **Xor**, **Imp** (Implication) , **Eqv** (Equivalence)

- Komparatoren:

  **=**, **<>**, **<=**, **<**, **>=**, **>**, **Is** (vergleicht zwei Objektreferenzen)

- Zeichenkettenoperatoren:

  **&** (Konkatenation), **like** (prüft Zeichenkette auf Muster – z.B.: **Dateiname Like ".exe"** )

- Funktionsaufrufe (später)

# Sprachelemente von VB: Operationen

Prof. Dr. Aris Christidis

**Bemerkungen den Folien dieses Abschnitts:**

- Unter einer **Operation** ist im hiesigen Kontext eine Aktion (Vorgang) zu verstehen, die eine gezielte Veränderung (Manipulation) von Daten durch eine Anweisung herbeiführt. Ihre Darstellung innerhalb einer Programmiersprache wird als **Operator**, die der Operation zugeführten Daten **Operand(en)** bezeichnet. Der erste hier erwähnte Operator (=) hat die niedrigste Priorität und wurde deshalb in der u.a. Prioritätsliste weggelassen.

  Die weniger häufig verwendeten der o.a. logischen Verknüpfungen haben jeweils folgende Wirkung:

- `Xor` ist wahr, wenn genau eine Bedingung wahr ist:

  0 0 | 0
  0 1 | 1
  1 0 | 1
  1 1 | 0

- `Imp` ist falsch, genau wenn Bedingung1 wahr und Bedingung2 falsch sind:

  0 0 | 1
  0 1 | 1
  1 0 | 0
  1 1 | 1

- `Eqv` ist wahr, wenn Bedingung1 und Bedingung2 gleich sind, sonst falsch:

  0 0 | 1
  0 1 | 0
  1 0 | 0
  1 1 | 1

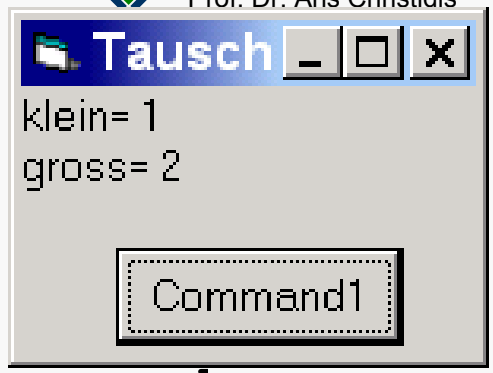
# Sprachelemente von VB: Operationen

Prof. Dr. Aris Christidis

Häufige zusammengesetzte Operationen:

- **Werte-Tausch unter Variablen:**

```
'(falsch:) x = y: y = x
Dim klein%, gross%, dummy%
klein = 2: gross = 1
dummy = klein: klein = gross: gross = dummy
Print "klein="; klein: Print "gross="; gross
```




- **Toggle (allg.:):**

```
Static malso%, z1%, z2%
'(meist:) malso = 1 - malso
If malso = 0 Then
z1 = 13: z2 = 8: malso = z1
End If
malso = z1 + z2 - malso
Print "malso="; malso
```


Toggle.exe

# Sprachelemente von VB: Operationen

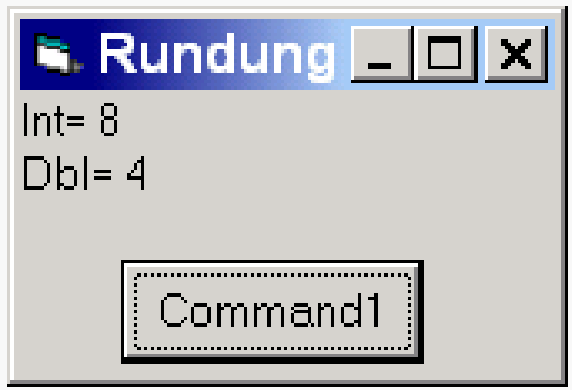
Prof. Dr. Aris Christidis

- Vorsicht vor Überlauf:

```
Dim Value As Long
Value = 100000
Value = Value * 200000 / 200000
'Zwischenergebnis>2.147.483.647
⇨sprengt Long-Rahmen
Value = 17 * 2000 / 2000
'Zwischenergebnis>32.767
⇨sprengt Integer-Rahmen!!
```


- Vorsicht vor impliziten Rundungen:

```
Dim intVal As Integer
Dim dblVal As Double
dblVal = 3.8
intVal = dblVal * 2
dblVal = intVal / 2
Print "Int="; intVal
Print "Dbl="; dblVal
```



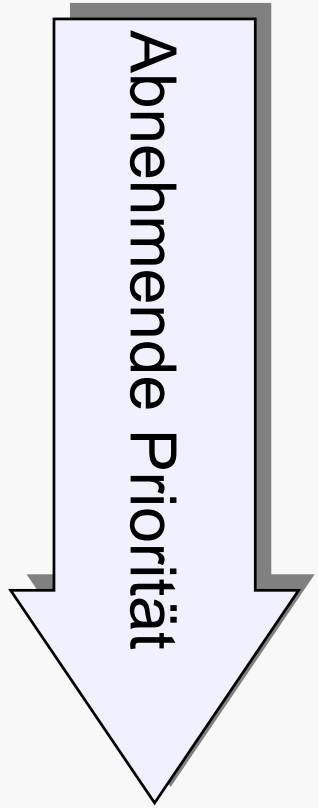
# Sprachelemente von VB: Operationen

Prof. Dr. Aris Christidis

**Prioritäten:**

Treten mehrere Operatoren in Ausdrücken auf, so werden sie in folgender Reihenfolge abgearbeitet:

- Funktionsaufrufe, Klammern
- `^`
- `-` (Vorzeichen)
- `*`, `/`
- `\` (ganzzahlige Division),
- `Mod` (Divisionsrest),
- `+`, `-`
- `=`, `<>`, `<` , `>`, `<=` , `>=` (logische Operatoren)
- `Not`
- `And`
- `Or`
- `Xor`
- `Eqv`
- `Imp`
- `=` (Zuweisung)



# Verzweigungen I

Prof. Dr. Aris Christidis

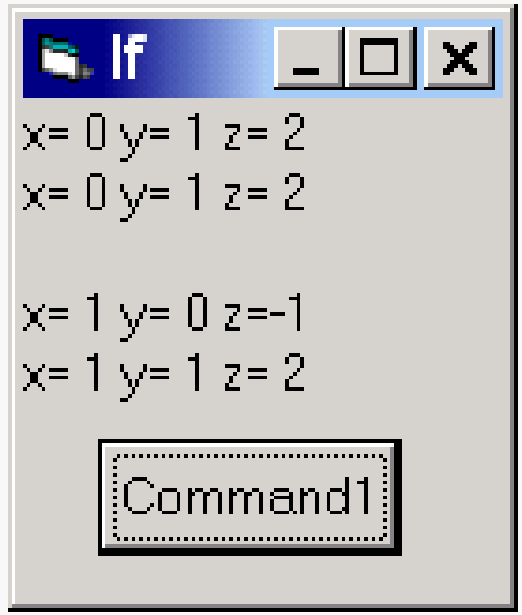
Ein- oder zweiseitige Auswahl:

einzeilig!

```
If BEDINGUNG Then ANWEISUNGEN [Else ANWEISUNGEN]
```

Beispiel:

```
Private Sub Command1_Click()
Static x%, y%, z%
If x = 1 Then y = 0: z = -1 Else
y = 1: z = 2
Print "x="; x; "y="; y; "z="; z
If x = 1 Then y = 0: z = -1 Else
y = 1: z = 2
Print "x="; x; "y="; y; "z="; z
x = 1 - x: Print
End Sub
```

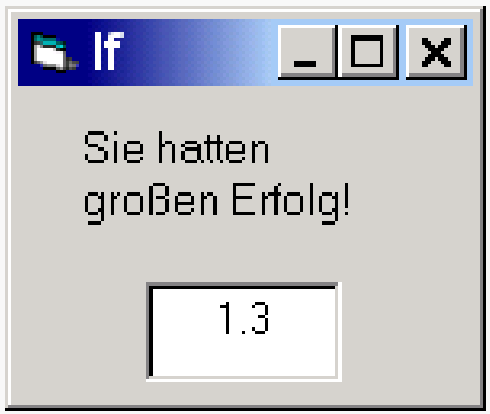


# Verzweigungen I

Prof. Dr. Aris Christidis

## Mehrstufige Auswahl:

```
If BEDINGUNG1 Then
ANWEISUNGEN
[ElseIf BEDINGUNG2 Then
ANWEISUNGEN
ElseIf BEDINGUNG3 Then
ANWEISUNGEN .....]
[Else ANWEISUNGEN]
End If
```



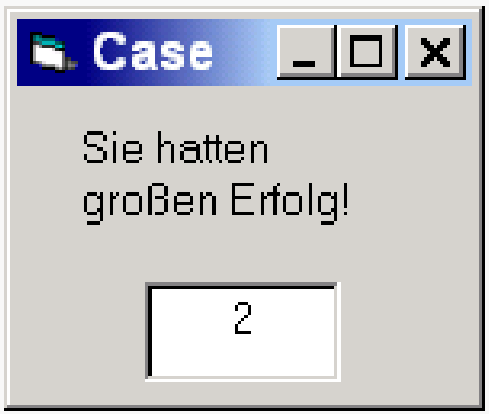

```
Private Sub Text1_Change()
Dim note!, urteil$
note = Val(Text1.Text)
urteil = ""

If note > 0 And note < 6 Then
urteil = "Sie hatten "

If note < 3 Then
urteil = urteil & "großen "

ElseIf note < 5 Then
urteil = urteil & "durchaus "
Else
urteil = urteil & "leider keinen "
End If

urteil = urteil & "Erfolg!"
End If
Label1.Caption = urteil
End Sub
```

# Verzweigungen II

Prof. Dr. Aris Christidis

Mehrseitige Auswahl:

```
Select Case AUSDRUCK
Case AUSDRUCK1
ANWEISUNGEN
[Case AUSDRUCK2
ANWEISUNGEN .....]
[Case Else
ANWEISUNGEN]
End Select
```




```
Private Sub Text1_Change()
Dim note%, urteil$ '(*)
note = Val(Text1.Text)
urteil = "Sie hatten "

Select Case note
Case 1 To 2: urteil = urteil & _
             "großen Erfolg!"
Case 3, 4: urteil = urteil & _
           "durchaus Erfolg!"
Case 5: urteil = urteil & _
        "leider keinen Erfolg!"
Case Else: urteil = ""
End Select

Label1.Caption = urteil
End Sub
```

(*) Es ist unüblich, aber möglich, den sog. Selektor (hier: d. Variable **note**) als **Single** zu deklarieren.

# Verzweigungen III

Prof. Dr. Aris Christidis

Mehrseitige Auswahl:

**Choose** (INDEX,AUSDRUCK1 [,AUSDRUCK2]...)


Beispiel:

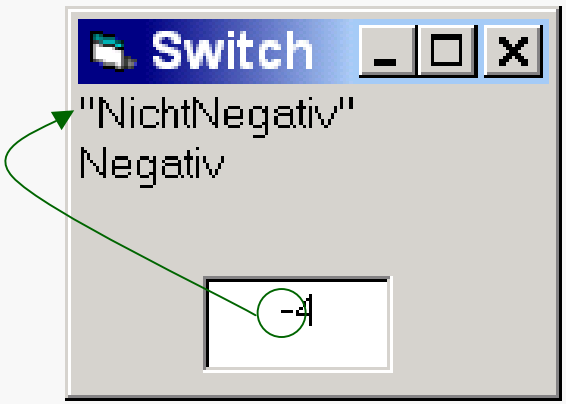
```
Private Sub Text1_Change()
Dim x%
Label1.Caption = ""
x = Val(Text1.Text)
If x >= 1 And x <= 3 Then
Label1.Caption = Choose(x, "Eins", "Zwo", "Zwei")
End If
End Sub
```

# Verzweigungen IV

Prof. Dr. Aris Christidis

Mehrseitige Auswahl:

```
Switch (AUSDRUCK1,VAR1 [AUSDRUCK2,VAR2])
```




Beispiel:

```
Private Sub Text1_Change()
Dim x%: x = Val(Text1.Text)
Print Switch(x < 0, "Negativ", _
    x >= 0, """NichtNegativ""", x > 0, "Positiv")
End Sub
```

wird nie erreicht!

# Verzweigungen V

Prof. Dr. Aris Christidis

Unbedingte Verzweigungen mit

**GoTo** (ohne Rückkehrmöglichkeit) und

**GoSub** (mit Rückkehrmöglichkeit) zu Zielmarken mit „:“

Beispiel:

```
Private Sub Command1_Click()
Dim Haben%, endlos As Boolean
endlos = False 'True '
MarktOhneEnde:
If Haben <= 0 Then GoSub Bank
Haben = Haben - 100
If endlos Then GoTo MarktOhneEnde
'damit kein Laufzeitfehler @ Return:
GoTo MarktMitEnde
Bank: Haben = Haben + 500: Return
MarktMitEnde: Beep
End Sub
```

endlos=True
➡ Absturz !

**Achtung:**

- Mit **GoTo** und **GoSub** können Programme sehr unübersichtlich werden („Spaghetti-Code“).
- Maßvolle (besser: keine) Verwendung nur, wo sie hilfreich sind (typisch: Fehlerbehandlung u.ä.)

# Schleifenanweisungen

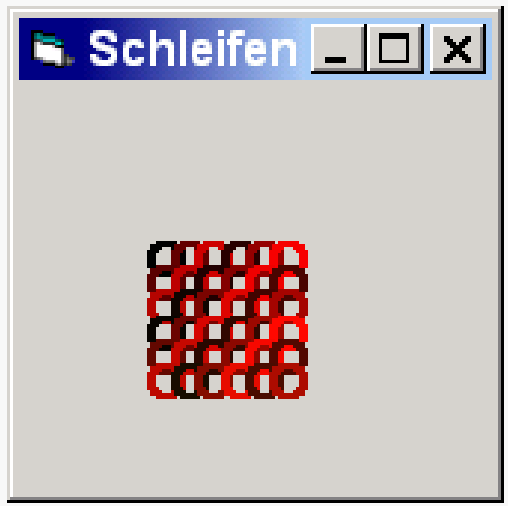
Prof. Dr. Aris Christidis

**Schleifenanweisungen** bieten eine kompakte Form für Programmteile, die mehrere Wiederholungen derselben Anweisungsfolgen benötigen.

Sie sind wichtig, vor allem weil zum Zeitpunkt der Programmierung meist gar nicht bekannt ist, wieviele Wiederholungen der Anweisungsfolgen benötigt werden.

Beispiele:

- Berechnung der neuen Gehälter nach einer Tariferhöhung
- Filterung und Wiedergabe (Rendering) in der Bildbearbeitung
- Texturierung von Flächen in der Computergrafik




(s.u.)

# Schleifenanweisungen I

Prof. Dr. Aris Christidis

**Kopfgesteuert:**

```
For ZÄHLER=ANFANG To ENDE [Step SCHRITT]
ANWEISUNGEN
[Exit For]
ANWEISUNGEN
Next [ZÄHLER]
```

Beispiel: Dreimalige Ausgabe

```
Dim j1%
For j1 = 1 To 3 Step 1
Print "Hallo!"
Next j1
```

# Schleifenanweisungen II

Prof. Dr. Aris Christidis

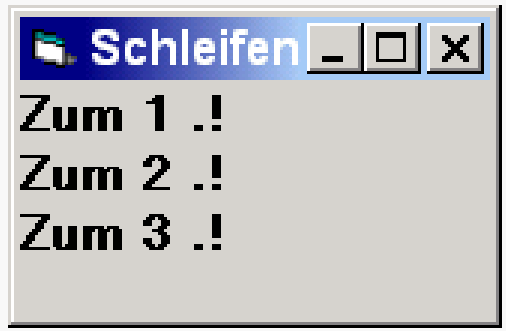
**Kopfgesteuert:**

```
Do [While|Until BEDINGUNG]
ANWEISUNGEN
[Exit Do]
ANWEISUNGEN
Loop
```

```
Dim j1%: j1 = 1
Do Until j1 > 3
Print "Zum"; j1; ".!„
j1 = j1 + 1
Loop
```

Beispiel: Dreimalige Ausgabe

```
Dim j1%: j1 = 1
Do While j1 <= 3
Print "Zum"; j1; ".!„
j1 = j1 + 1
Loop
```

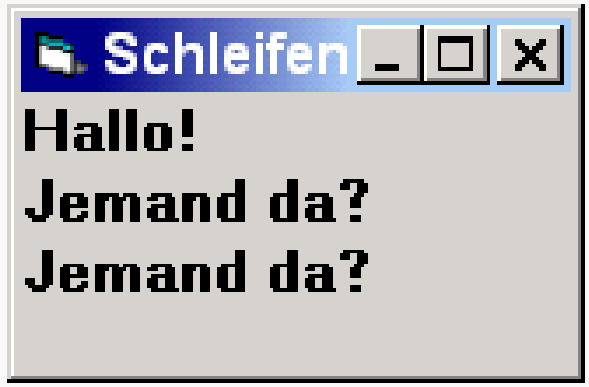


# Schleifenanweisungen III

Prof. Dr. Aris Christidis

**Kopfgesteuert:**

```
While BEDINGUNG
ANWEISUNGEN
Wend
```




Beispiel: Dreimalige Ausgabe

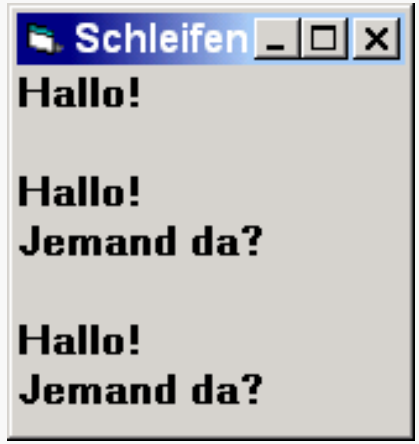
```
Dim j1%: j1 = 0
While j1 <= 2
If j1 = 0 Then Print "Hallo!" _
Else Print "Jemand da?„
j1 = j1 + 1
Wend
```

# Schleifenanweisungen IV

Prof. Dr. Aris Christidis

**Fußgesteuert:**

**Do**
ANWEISUNGEN
[**Exit Do**]
ANWEISUNGEN
**Loop** [**While**|**Until** BEDINGUNG]




```
Dim j1%: j1 = 1
Do
Print "Hallo!"
If j1 > 1 Then _
Print "Jemand da?"
j1 = j1 + 1: Print
Loop Until j1 > 3
```

Beispiel: Dreimalige Ausgabe

```
Dim j1%: j1 = 1
Do
Print "Hallo!"
If j1 > 1 Then Print "Jemand da?"
j1 = j1 + 1: Print
Loop While j1 <= 3
```

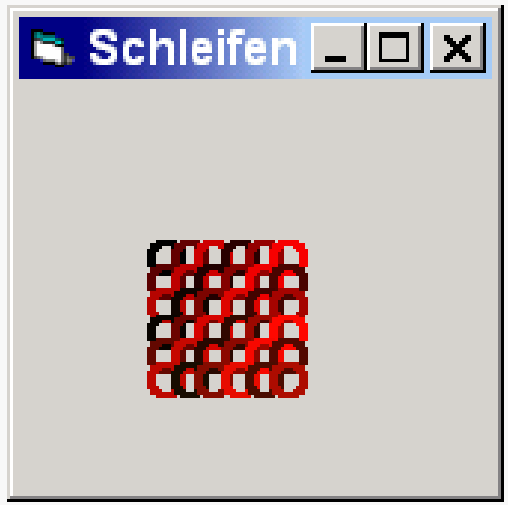
```
Dim j1%: j1 = 1
Do
Print "Hallo!"
If j1 > 1 Then _
Print "Jemand da?"
If j1 = 3 Then Exit Do
j1 = j1 + 1: Print
Loop
```

# Schleifenanweisungen

Prof. Dr. Aris Christidis

## Anwendungsbeispiel: Flächenbelegung

```
Private Sub Form_Click()
Dim jX%, jY%
Form1.ForeColor = 0
For jY = Form1.Height / 4 To Form1.Height / 2 Step 100
For jX = Form1.Width / 4 To Form1.Width / 2 Step 100
Form1.CurrentX = jX
Form1.CurrentY = jY
Print "o"
Form1.ForeColor = Form1.ForeColor + 100
Next jX
Next jY
End Sub
```




(Schleifen.exe)

- **Konstanten, Felder u. benutzerdefinierte Datentypen**
- **Funktionen und Prozeduren**
- **Dateien**

# Sprachelemente von VB: Konstanten

Prof. Dr. Aris Christidis

- ...sind Platzhalter für Werte, die während der Ausführung eines Programms benötigt (u. nicht mehr verändert) werden.
- ...bieten Abstraktioshilfe: Bezeichner statt Werte.
- ...bekommen nur bei der Deklaration einen Wert zugewiesen.
- ...haben Namen, Datentyp, Gültigkeitsbereich (wie Variablen).

Beispiele:

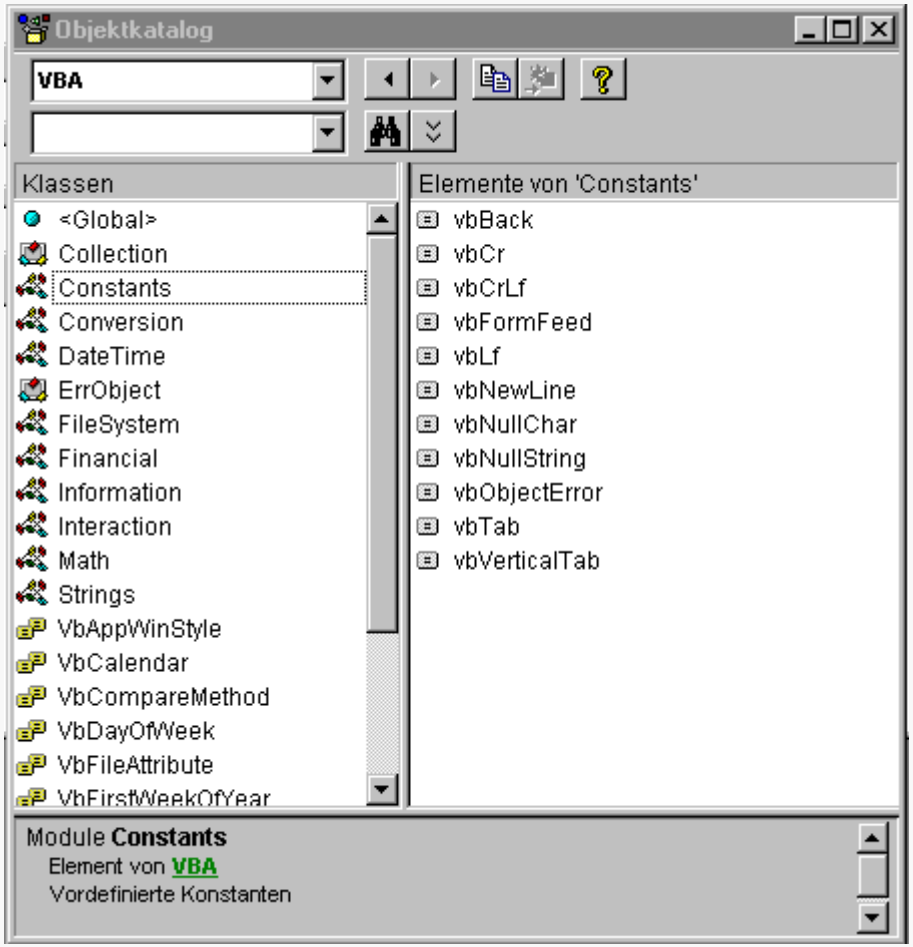
```
Const GRUSS$ = "Hallo!", Epsilon#=1E-6
Const pi# = 4 * Atn(1#)  'Fehlermeldung:
                         'Konstanter Ausdruck erforderlich
```

# Sprachelemente von VB: Konstanten

Prof. Dr. Aris Christidis

## Vordefinierte Konstanten:

Aufruf einer Liste vordefinierter Konstanten mit [F2]




- String-Konstanten:
  - Zeilenumbruch
  - ...
- Grafische Konstanten
  - Linienarten
  - Linienfarben
  - Linienbreiten
  - Farben
  - ...
- ...

➡ In Programmen keine absoluten Werte, sondern Konstanten benutzen!

# Sprachelemente von VB: Konstanten



**Aufzählungen** („*Enumerations*“): „Geordnete“ Konstanten

```
[Public | Private] Enum TYPNAME
    BEZEICHNER1 [= KONSTANTER_AUSDRUCK1] '(def.:0)
    BEZEICHNER2 [= KONSTANTER_AUSDRUCK2] '(def.:1)
    ...
End Enum
```

Vorteile und Besonderheiten:

- **`Enum`** gewährleistet konsistenten Einsatz der Bezeichner – so nach Änderung ihrer Folge oder Hinzufügen weiterer Werte
- Die **`Enum`**-Anweisung definiert jeweils einen neuen Datentyp; bei Deklaration von Variablen dieses Typs werden gültige Werte in Combobox angezeigt.
- Standardmäßig sind **`Enum`**-Datentypen öffentlich (**`Public`**) – mit Zusatz **`Private`** auf das Modul beschränkbar.
- Deklaration nur außerhalb aller Unterprogramme möglich (sonst Fehlermeldung: „Innerhalb einer Prozedur ungültig“).

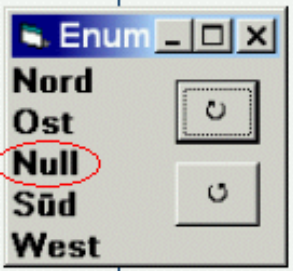
# Sprachelemente von VB: Konstanten

Prof. Dr. Aris Christidis

Beispiel:

```
Enum Himmelsrichtg 'Typ-Def.
KEINE = -1   'oft für "Nicht-.."
NORD         '=0(=KEINE+1)
OST = -True  '=1(Umwdl.Integer)
SUED = 2.7   '=3(Rundg.Integer)
WEST         '=4(=SUED+1)
SONST = -2   'unüblich, aber OK
End Enum
```

```
Private Sub cmdImU_Click()
Dim Richtg%: Cls 'ClearScreen
      For Richtg = NORD To WEST
Print Switch _
(Richtg = NORD, "Nord", _
 Richtg = OST, "Ost", _
 Richtg = SUED, "Süd", _
 Richtg = WEST, "West")
Next Richtg
End Sub
```




```
Private Sub cmdGgnU_Click()
'Typ-Definition @ Enum (s.o.)
Dim Richtg As Himmelsrichtg
Richtg = WEST: Cls
 Do While Richtg >= NORD
If Richtg = NORD Then _
   Print "Nord" Else
If Richtg = OST Then _
   Print "Ost" Else
If Richtg = SUED Then _
   Print "Süd" Else
If Richtg = WEST Then _
   Print "West"
Richtg = Richtg - 1
 Loop
End Sub
```



(Enum.exe)

# Felder



- Feld-Deklaration:

```
Dim VARNAME([[UNTERE_GRENZE To] OBERE_GRENZE _
      [,[UNTERE_GRENZE  To]  OBERE_GRENZE]])  _
      [As [New] DATENTYP]
```

- Ein Feld (indizierte Variable, engl. *array*) eignet sich zur Aufnahme “gleichartiger” Daten unter einem Bezeichner.
- Ist keine untere Grenze angegeben, so ist der Standardwert 0.
- Beispiele:

```
Dim vektor!(1 To 3), mat!(1 To 3, 1 To 3) 'Single
Dim Person$(2)                            'String
Const Vorname% = 0, Nachname% = 1
Person(Vorname) = "Aris"
Person(Nachname) = "Christidis"
```

- Häufige Fehlerquelle:
  Wieviele Elemente hat `Dim Feld(100)`?

# Felder

Prof. Dr. Aris Christidis

Beispiel: Vektor-Addition:

```
Private Sub Command1_Click()
Dim j%, vec1!(1 To 3), vec2!(1 To 3)'j:Integer;vec*:Single

vec1(1) = 1#: vec1(2) = 2#: vec1(3) = 3#
vec2(1) = 0.1: vec2(2) = 0.2: vec2(3) = 0.3

For j = 1 To 3
vec1(j) = vec1(j) + vec2(j)
Print j; ":"; vec1(j)
Next j
End Sub
```


06Feld

# Felder



- Syntax der Deklaration dynamischer Feldvariablen :

  ```
  Dim DYNAMISCHE_FELDVARIABLE()
  ```

  (statt **Dim** sind auch die Formen **Public**, **Private**, **Static** möglich)

- Vor Wertzuweisung muß die Dimension feststehen:

  ```
  ReDim VARIABLE(NEUE_GRENZE[N]) 'löscht Inhalt
  ReDim Preserve VARIABLE(NEUE_OBERGRENZE)'erhält I.
  ```

  Behält die Werte an ihren ursprünglichen Indizes (sofern vorhanden, falls das Feld kleiner wurde – d.h.: keine Umordnung)

- **Redim**-Anweisungen dürfen beliebig oft im Code stehen (Zeit!)

- Mit **Preserve** ist nur die Obergrenze der letzten Dimension veränderbar! (Ohne: bel. Änderungen, auch neue Dimensionen)

- Abfrage der Feld-Grenzen durch die Funktionen:

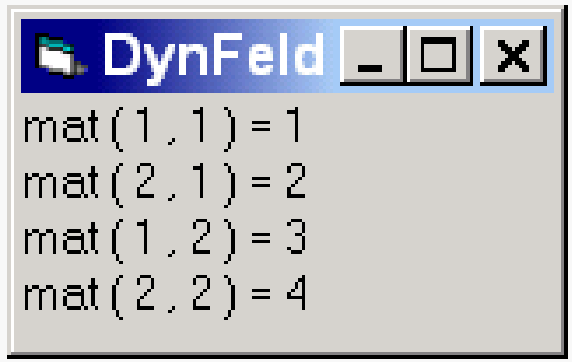
  ```
  Lbound(VARIABLE[, Dimension])  'untere Grenze
  Ubound(VARIABLE[, Dimension])  'obere  Grenze
  ```

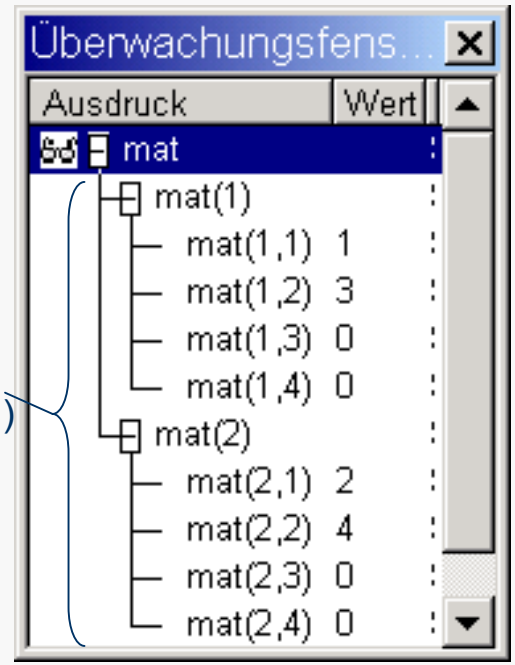
# Felder

Prof. Dr. Aris Christidis

Beispiel: Belegung einer dynamisch deklarierten Matrix:

```
Private Sub Form_Click()
Dim j%, jz%, js%, mat!() 'Int.,Sing.
ReDim mat(1 To 2, 1 To 2)
Cls 'ClearScreen

For j = 1 To 4
js = (j - 1) \ 2 + 1
jz = j - 2 * (js - 1)
mat(jz, js) = j
Print "mat ("; jz; ","; js; ") ="; _
      mat(jz, js)
Next j
ReDim Preserve _
mat(1 To 2, 1 To 2 * UBound(mat, 2))
' (Ausbau...)
End Sub
```




(zeilenweise Speicherung)

07DynFeld

# Benutzerdefinierte Datentypen

Prof. Dr. Aris Christidis

- Syntax Typdefinition:

```
[Public|Private] Type TYPNAME
  ELEMENTNAME As DATENTYP
  ...
End Type
```

vgl. `typedef struct` (C/C++)
`record` (Pascal)

- Beispiel:

```
Type Personenname
  Vorname As String * 20
  Nachname As String * 20
End Type
Type Personendaten
  Name As Personenname
  Personalnr As Integer
  Apparat As String
End Type
Public Person(1 to 100) As Personendaten
```

Definition zweier Datentypen

Strukturen mit Elementen

Deklaration einer Strukturvariablen

# Benutzerdefinierte Datentypen

Prof. Dr. Aris Christidis

Syntax für den Zugriff auf Strukturvariablen:

- Über den “Qualifizierer” ‘.’:

```
STRUKTURNAME.ELEMENTNAME = WERT
```

Beispiel:

```
Person(1).Name.Vorname = "Aris"
Person(1).Name.Nachname = "Christidis"
```

- Mit der **With**-Anweisung:

```
With STRUKTURNAME
  .ELEMENTNAME1 = WERT
  .ELEMENTNAME2 = WERT
  '(...)
End With
```

Beispiel:

```
With Person(1).Name
  .Vorname = "Aris"
  .Nachname = "Christidis"
End With
```

# Benutzerdefinierte Datentypen



Beispiel:

```
Private Type PNam
    Vorname As String * 10
    Nachname As String * 15
End Type
Private Type PDat
    Name As PNam
    Denke As Variant
End Type
Private Sub cmdSucht_Click()
Dim Person(1 To 100) As PDat
Dim Wanted As PDat, j%
Const Massnahme$ = "fassen!"
For j = 1 To 100:Select Case j
Case 1: '...
Case 3: With Person(j).Name
    .Vorname = "Aris"
    .Nachname = "Christidis"
        End With
Case 100: '...
End Select: Next j '=>
```

```
Wanted.Name.Vorname = "Aris"

For j = 1 To 100
If Person(j).Name.Vorname = _
     Wanted.Name.Vorname Then
Wanted = Person(j) 'setzen
Exit For
End If: Next j

If Wanted.Name.Nachname <> "" Then
Print "Erster Fund:"
With Wanted.Name
Print .Vorname; .Nachname; _
                  Massnahme
End With: End If
End Sub
```


# Funktionen und Prozeduren



- Häufig verwendete, in sich geschlossene Folgen von Anweisungen werden oft als separate Einheiten, sog. **Unterprogramme** (engl. *subprograms*) realisiert und unter einem Namen abgelegt.

  Unterprogramme können von mehreren Stellen beliebiger Programme oder Unterprogramme aufgerufen werden (vgl. VB-Anweisungen).

  Den Aufruf eines Unterprogramms durch das Unterprogramm selbst nenn man eine Rekursion.

- Nach Aufruf und Abarbeitung eines Unterprogramms wird die Ausführung des aufrufenden Programms mit der auf den Aufruf folgenden Operation oder Anweisung fortgesetzt.

# Funktionen und Prozeduren



VB kennt zwei Arten von Unterprogrammen:

- Eine Funktion erzeugt ein Ergebnis (z.B. den Grenzwert einer Folge), das einer Variablen zugewiesen oder in einem Ausdruck verwendet werden kann.

  Funktionen können vordefiniert (Bestandteil der Sprache) oder benutzerdefiniert (vom Benutzer programmiert) sein.

- Eine Prozedur ist d. Zusammenstellung von Anweisungen (z.B.: Ermittlung der Lösungen eines GI.-Systems und deren Zuweisung an ein Feld); sie ist somit universeller, aber nicht immer intuitiv einsetzbar.

  Prozeduren können vordefiniert oder benutzerdefiniert sein; vordefinierte P. werden aufgerufen, wenn sog. Ereignisse (vorgesehene Situationen in Soft- u./o. Hardware) eintreten.

# Funktionen und Prozeduren



**Vordefinierte Funktionen:**

- Rechenfunktionen
- Stringfunktionen
- Umwandlungsfunktionen
- Datumsfunktionen
- Statusfunktionen
- Sonstige Funktionen

# Funktionen und Prozeduren



## Rechenfunktionen:

| | |
|---|---|
| **`Abs`** | Absolutwert (Zahlenwert ohne Vorzeichen) |
| **`Atn`** | Arcustangens - Winkel zwischen -π/2 und π/2 |
| **`Cos`** | Cosinus |
| **`Exp`** | Potenz von e (maximal: 709782712893) |
| **`Fix`** | Abschneiden des Nachkommateils (4,6 zu 4; -4,6 zu -4) |
| **`Int`** | nächstkleinere ganze Zahl (4,6 wird 4, -4,6 wird -5) |
| **`Log`** | natürlicher Logarithmus (zur Basis e) |
| **`Rnd`** | Pseudozufallszahl zwischen 0 und 1 |
| | (Vorsicht: **`Rnd`** erzeugt bei jedem Programmlauf die gleiche Sequenz - daher mit **`Randomize`** Timer initialisieren!) |
| **`Sgn`** | Vorzeichen: -1 bei negativen Zahlen, 0 bei 0, 1 pei positiven Z. |
| **`Sin`** | Sinus |
| **`Sqr`** | Quadratwurzel (nur positive Argumente erlaubt) |
| **`Tan`** | Tangens |

# Funktionen und Prozeduren



## Rechenfunktionen – Bemerkungen:

- Zum Quadrieren Produkt vorziehen (^-Operator langsam):

```
x = x * x          'statt x ^ 2
```

- Winkelfunktionen im Bogenmaß:

```
pi = 4 * Atn(1) 'liefert 3,14159265358979

Dim phi#, d2r#  'Double (phi in Grad gedacht)
d2r = Atn(1) / 45
phi = phi * d2r 'phi ab hier im Bogenmaß
```

- Funktionen wirken oft ähnlich – z.B. **Fix** (Abschneiden des Nachkommateils) u. **Int** (nächstkleinere ganze Zahl):

```
x = Fix(x)         '=Sgn(x)*Int(Abs(x))
```

- Ganze Zufallszahlen zwischen beliebigen Ober-/Untergrenzen:

```
Dim z%, Oben%, Unten% 'Integer
Randomize             'Timer-Initialisierung
Oben = 6 : Unten = 1  'Spielwürfel
z = Int((Oben - Unten + 1) * Rnd + Unten)
```

# Funktionen und Prozeduren



## Stringfunktionen:

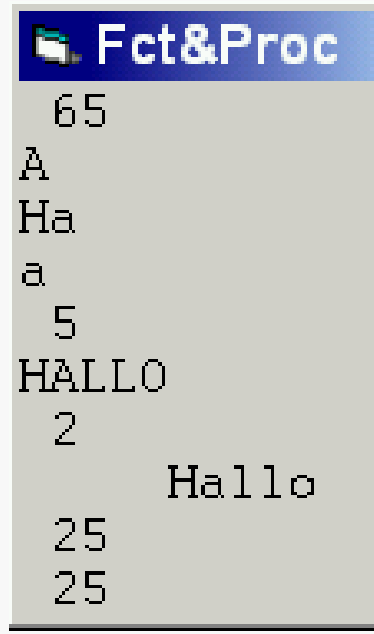
| | |
|---|---|
| **`Chr`** | Byte-Umwandlung in 1-Zeichen-String – z.B.: **`Chr(65)="A"`** |
| **`Instr`** | Gibt Position eines Teilstrings in einem String zurück |
| **`Left`** | Trennt Zeichen links von einem String ab |
| **`Len`** | Ermittelt Länge eines Strings |
| **`LTrim`** | Schneidet führende Leerstellen von einem String ab |
| **`Mid`** | Ermittelt einen String in der Mitte eines anderen Strings |
| **`Right`** | Trennt Zeichen rechts von einem String ab |
| **`RTrim`** | Schneidet nachfolgende Leerstellen von einem String ab |
| **`Str`** | Wandelt einen Ausdruck in einen String um |
| **`StrComp`** | Vergleicht zwei Strings |
| **`StrConv`** | Konvertiert einen String z.B. von Klein- in Großbuchstaben oder von ANSI-Code in UNICODE |
| **`String`** | Erstellt einen String mit einer Anzahl gleicher Zeichen |
| **`Trim`** | Schneidet führende und nachfolgende Leerzeichen von String ab |

# Funktionen und Prozeduren

Prof. Dr. Aris Christidis

**Stringfunktionen – Beispiele:**

```
Print Asc("A")           ' ergibt 65
Print Chr$(65)           ' ergibt "A"
Const Text$ = "Hallo"
Print Left$(Text, 2)     ' ergibt "Ha"
Print Mid$(Text, 2, 1)   ' ergibt "a"
Print Len(Text)          ' ergibt 5
Print UCase$(Text)       ' ergibt "HALLO"
Print InStr(Text, "a")   ' ergibt 2
Print Space(5) & Text    ' ergibt "     Hallo"
Print Val("25")          ' ergibt 25
Print Str$(25)           ' ergibt " 25"
                         ' (mit Leer- o.Vorzeichen)
```




Funktionennamen mit '**$**' ergeben Strings, sonst Variant!
(z.B.: **Mid$** / **Mid** – Variant ist speicher- und zeitintensiv)

# Funktionen und Prozeduren

Prof. Dr. Aris Christidis

## Stringfunktionen

Beispiel:

Extraktion eines Dateinamens, Austausch der Extention:


```
Private Sub Text1_Change()
Dim filnam$             'String
filnam = Text1.Text 'z.B.:"C:\dir\file.txt"
Print filnam
  Do While InStr(filnam, "\") <> 0
filnam = Right(filnam, Len(filnam) - InStr(filnam, "\"))
  Loop
Print filnam
filnam = Left(filnam, InStr(filnam, ".")) & "bla"
Print filnam
End Sub
```

# Funktionen und Prozeduren

Prof. Dr. Aris Christidis

ASCII: 7 Bit / Zeichen
ANSI: 8 Bit / Zeichen

## Umwandlungsfunktionen:

| | |
|---|---|
| **`Asc`** | Ermittelt ANSI-Code eines String-Zeichens |
| **`CBool`** | Wandelt numerischen Ausdruck in Typ Boolean um |
| **`CByte`** | Wandelt numerischen Ausdruck in Zahl vom Typ Byte um |
| **`CCur`** | Wandelt numerischen Ausdruck in Zahl vom Typ Currency um |
| **`CDate`** | Wandelt numerischen Ausdruck in Zahl vom Typ Date um |
| **`CDec`** | Wandelt numerischen Ausdruck in Zahl vom Typ Decimal um |
| **`CDbl`** | Wandelt numerischen Ausdruck in Zahl vom Typ Double um |
| **`CErr`** | Wandelt numerischen Ausdruck in Zahl vom Typ Error um |
| **`CInt`** | Wandelt numerischen Ausdruck in Zahl vom Typ Integer um |
| **`CLng`** | Wandelt numerischen Ausdruck in Zahl vom Typ Long um |
| **`CSng`** | Wandelt numerischen Ausdruck in Zahl vom Typ Single um |
| **`Cvar`** | Wandelt numerischen Ausdruck in Zahl vom Typ Variant um |
| **`Hex`** | Wandelt ganzzahligen Ausdruck in eine Hexadezimalzahl um |
| **`Oct`** | Wandelt ganzzahligen Ausdruck in eine Oktalzahl um |
| **`Val`** | Berechnet numerischen Wert eines Strings |

# Funktionen und Prozeduren



## Datumsfunktionen:

| | |
|---|---|
| `CVDate` | Wandelt Datumsstring in einen Datumstyp um |
| `Date` | Liefert das Systemdatum oder stellt es ein |
| `DateSerial` | Wandelt drei Werte (Jahr, Monat, Tag) in Datumstyp um |
| `DateValue` | Wandelt Datumsstring in einen Datumstyp um |
| `DateAdd` | Berechnet ein Datum als Offset von einem anderen |
| `DateDiff` | Berechnet die Differenz zwischen zwei Daten in Tagen |
| `DatePort` | Berechnet aus einem Datum z.B. die KW oder das Quartal |
| `Day` | Extrahiert aus einem Datum die Tageszahl |
| `Hour` | Extrahiert aus einem Datum die Stundenzahl |
| `Minute` | Extrahiert aus einem Datum die Minutenzahl |
| `Month` | Extrahiert aus einem Datum die Monatszahl |
| `Now` | Liefert das Systemdatum und die Systemzeit |
| `Second` | Extrahiert aus einem Datum die Sekundenzahl |
| `Time` | Liefert die Systemzeit oder stellt sie ein |
| `Timer` | Liefert Sekunden seit Mitternacht |
| `Weekday` | Extrahiert aus einem Datum die Wochentag-Zahl (1 bis 7) |
| `Year` | Extrahiert aus einem Datum die Jahreszahl |

# Funktionen und Prozeduren



## Statusfunktionen:

| | |
|---|---|
| **`IsArray`** | Stellt fest, ob ein Ausdruck den Untertyp Array besitzt |
| **`IsDate`** | Stellt fest, ob ein Ausdruck den Untertyp Date besitzt |
| **`IsVariant`** | Stellt fest, ob ein Ausdruck den Typ Variant besitzt |
| **`IsEmpty`** | Stellt fest ob eine Variable vom Typ Variant initialisiert wurde |
| **`IsObject`** | Stellt fest, ob ein Ausdruck eine Refernz auf ein Objekt ist |
| **`IsError`** | Stellt fest, ob ein Ausdruck den Typ Error besitzt |
| **`IsMissing`** | Stellt fest, ob ein optionaler Parameterwert belegt ist |
| **`IsNull`** | Stellt fest, ob ein Ausdruck den Wert NULL besitzt |
| **`IsNumeric`** | Stellt fest, ob ein Ausdruck numerisch ist |
| **`TypeName`** | Gibt den Typ eines Ausdrucks als Zeichenkette zurück |
| **`VarType`** | Gibt den Typ eines Ausdrucks als Zahl zurück |

# Funktionen und Prozeduren



## Sonstige Funktionen:

| | |
|---|---|
| **`Environ`** | Gibt den Wert einer DOS-Environment-Variablen zurück |
| **`Dir`** | gibt den Dateinamen mit übergebenem Muster zurück oder schaltet zum nächsten ( mit dir() ) |
| **`FileAttr`** | Gibt Zugriffsattribute einer geöffneten Datei zurück |
| **`FileDateTime`** | Gibt Datum der letzten Änderung einer Datei zurück |
| **`FileLen`** | Gibt die Länge einer Datei in Byte zurück |
| **`Loc`** | Gibt Position des Schreib-/Lesekopfes in Datei zurück |
| **`LOF`** | Gibt die Länge einer geöffneten Datei in Byte zurück |

..., z.B. finanzmathematische Funktionen

# Funktionen und Prozeduren



Benutzerdefinierte Unterprogramme werden typischerweise eingerichtet für Programm-Teile, die

- universell anwendbar sind (z.B. Lösung v. Gl.-Systemen),
- vereinheitlicht bleiben müssen (z.B. Logo einer Institution),
- aufwendig i.d. Programmierung sind (z.B. Grafik-Funktionen),
- so häufig und umfangreich auftreten, daß sie die Übersichtlichkeit des Programms stören (z.B. Fehlermeldungen),
- sich innerhalb eines unübersichtlichen Programms als (möglichst sinnvolle) Einheiten auslagern lassen (z.B. Menüs).

Sie werden meist zu Gruppen mit verwandten Themen in Dateien, (VB:) in sog. **Modulen**, zusammengefaßt.

# Funktionen und Prozeduren



Bei verstärkter Einrichtung von Unterprogrammen spricht man von **modularer Programmiertechnik**.

Vorteile dieser Technik für die Sw-Entwicklung sind

- die größere **Übersichtlichkeit** der Programme,
- die zuverlässigere Projekt-**Planbarkeit** (inhaltlich/zeitlich),
- die leichtere **Testbarkeit**,
- die sicherere **Integrierbarkeit**,
- die bessere **Wiederverwendbarkeit**.

# Funktionen und Prozeduren

Prof. Dr. Aris Christidis

Gemeinsamkeiten der Unterprogramme (UPe):

- **Definition** (einfachste Form):

```
Function | Sub NAME([PARAM1,...])
ANWEISUNGEN
End Function | Sub
```

Formalparameter

**Aufruf** (einfachste Form):

Aktualparameter

```
NAME(PARAM1,...)          'zu: Function NAME
Call NAME(PARAM1,...)     'zu: Sub NAME
```

bzw.

- Die Übergabe von Argumenten (Werten) aus dem aufrufenden Programm an ein aufgerufenes UP erfolgt über gemeinsame Speicherbereiche (globale Variablen) oder über Parameterlisten.

- Die Verwendung einer Parameterliste bewirkt, daß beim Aufruf die lokalen Variablen des UPs (Formalparameter) mit den entsprechenden Variablen des aufrufenden Programms (Aktual-parametern) assoziiert werden. (Das kann bedeuten, daß sie nach Abarbeitung des UPs einen anderen Inhalt haben – s.u.)

# Funktionen und Prozeduren



- Syntax einer **Funktionsdefinition** (einfache Form):

```
[Public | Private] Function _
  NAME([PARAM1 [As DATENTYP],...]) [As DATENTYP]
ANWEISUNGEN
[Exit Function]
ANWEISUNGEN
End Function
```

- Der Rückgabewert wird einer lokalen, implizit deklarierten Variablen (quasi: dem Funktionsnamen) zugewiesen.
- Übergebene Parameter (Argumente) sind lokale Variablen.
  (D.h. hier: ihr Wert kann sich in der Funktion verändern.)
- Aufruf von Funktionen ohne Parameter: `NAME`
  Aufruf von Funktionen mit Parametern: `NAME(PARAM1,...)`

# Funktionen und Prozeduren

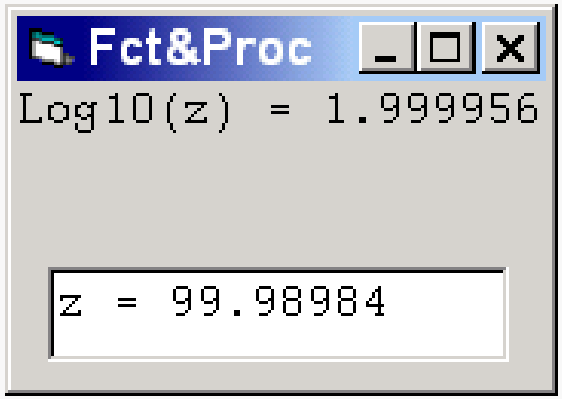
Prof. Dr. Aris Christidis

Beispiel für Funktionen:

```
Private Sub Form_Click()
Dim z!: Cls
z = Zufall    'Aufruf ohne Parameter
Text1.Text = "z =" & Str(z)
z = Log10(z) 'Aufruf mit Parametern
Print "Log10(z) =" & Str(z)
End Sub
```

```
Function Zufall() As Single
Randomize              'Initialisierg
Zufall = 100 * Rnd '<100
End Function
```

```
Function Log10!(x!)          'Single
Log10 = Log(x) / Log(10)
End Function
```




(FctRndLog.exe)

# Funktionen und Prozeduren



- Syntax einer **Prozedurdefinition** (einfache Form):

```
[Public | Private] Sub NAME([PARAM1,...])
ANWEISUNGEN
[Exit Sub]
ANWEISUNGEN
End Sub
```

- Übergebene Parameter (Argumente) sind lokale Variablen.
  (D.h. hier: ihr Wert kann sich in der Prozedur verändern.)

- Aufruf von Prozeduren ohne Parameter:
  `NAME`
  Aufruf von Prozeduren mit Parametern:
  `Call NAME(PARAM1, ...)` oder:
  `NAME PARAM1, ...`

# Funktionen und Prozeduren

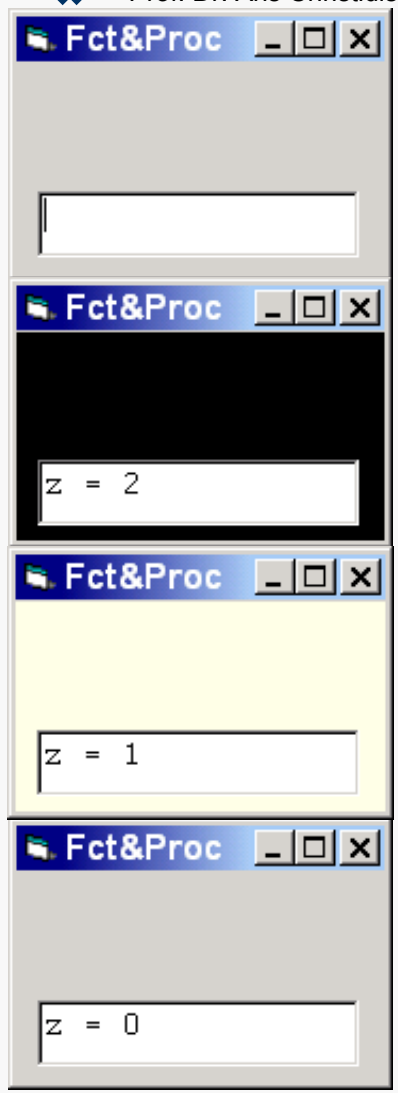


Beispiel für Prozeduren:

```
Private Sub Form_Click()
Dim z%: z = 0 'überflüssig
Call farbe(z) 'Aufruf mit Parametern
farbe z       'auch möglich
Text1.Text = "z =" & Str(z)
End Sub
```

```
Sub farbe(x%)
Static intra%: intra = (intra + 1) Mod 3
Select Case intra
 Case 1: Form1.BackColor = &H80000018
 Case 2: Form1.BackColor = &H80000008
 Case Else: loesch 'Aufruf o. Parameter
End Select
x = intra
End Sub
```

```
Sub loesch()
Form1.BackColor = &H8000000F
End Sub
```




(SubFarb.exe)

# Funktionen und Prozeduren

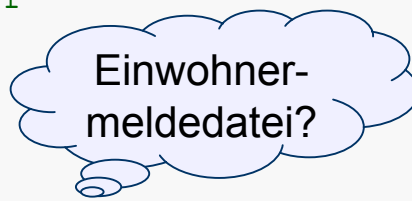


Besonderheiten bei der Parameterübergabe:

- Parameterübergabe „**By Reference**“ (`[ByRef]`):

  Übergabe der Adresse des jeweiligen Parameters; Veränderung des Originals in der Prozedur (default) – z.B.:

```
Sub Korrektur (Text As String) 'Korrektur-Hinweise
'(...)
Call Korrektur (Text)                'Text-Manipulation
```




- Parameterübergabe „**By Value**“ (**`ByVal`**):

  Übergabe des Wertes des jeweiligen Parameters (Kopie); Veränderung des Originals in der Prozedur unmöglich-z.B.:

```
Sub Korrektur (ByVal Text As String) 'Korr.-Hinweise
'(...)
Call Korrektur (Text)                'Text bleibt gleich
```

# Funktionen und Prozeduren



Besonderheiten bei der Parameterübergabe (Forts.):

- **Benannte Argumente**:

  Die Reihenfolge übergebener Argumente braucht im Aufruf von Prozeduren oder Funktionen nicht eingehalten zu werden, wenn die Aktualparameter „*benannt*“ werden, d.h., wenn sie mit dem Operator **:=** den entsprechenden Formalparametern zugeordnet werden.

  Beispiel:

```
Sub farbe(x%, y%)
'(...)
End Sub
```

  Aufruf ohne / mit „*Benennung*“ der Variablen:

```
Dim z%
Call farbe(z, 1)          'Standard
Call farbe(y:=1, x:=z)    'auch möglich
```

# Funktionen und Prozeduren

Prof. Dr. Aris Christidis

Besonderheiten bei der Parameterübergabe (Forts.):

- **Optionale Parameter**:

  Parameter können beim Aufruf weggelassen werden, wenn sie bei der Definition des UPs als „*optional*“ deklariert wurden.

  Argument-Entsprechungen sind dann mit **:=** zu kennzeichnen!

  Beispiel:

```
Sub farbe(Optional x%, Optional y%) 'y:für Erweiterg
'(...)
End Sub
```

  Aufruf mit obligatorischer „*Benennung*“ verwendeter Variablen:

```
Dim z%: farbe x:=z        'Beispiele zulässiger Aufrufe
farbe
Call farbe
Call farbe(y:=1, x:=z) 'Reihenfolge egal (wg. :=)
```

- Mit der Funktion **IsMissing** können Argumente auf Existenz überprüft werden, damit sie ggf. belegt werden – z.B.:

```
If IsMissing(y) Then y = 0
```