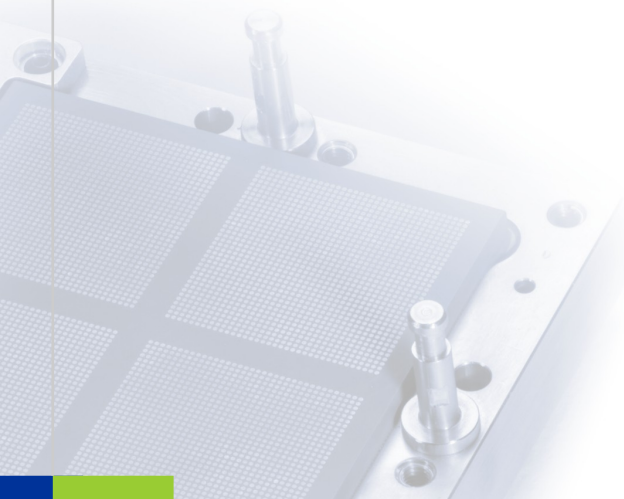


VBA (Visual Basic for Application)

Prozeduren



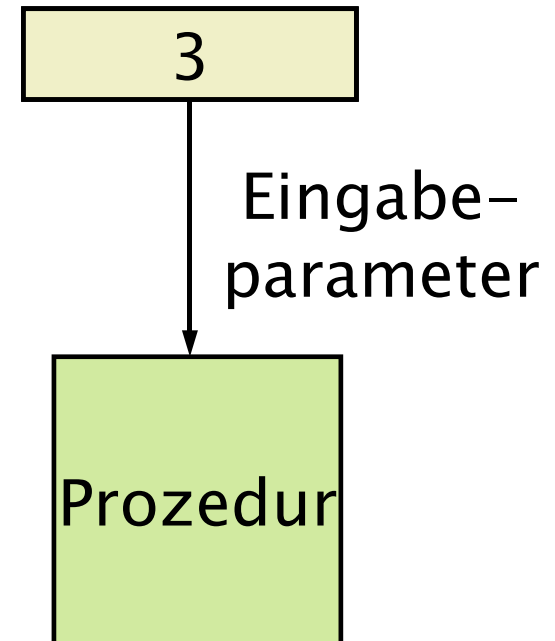
- ... sind kleine Unterprogramm (Subroutinen) mit denen Teilprobleme einer größeren Aufgabe gelöst werden können.
- ... fassen Anweisungen, die eine bestimmte Aufgabe lösen, zu einem Block zusammen.
- ... sind eine Abfolge von VBA-Befehlen zu einem bestimmten Thema
- ... werden zeilenweise abgearbeitet.

- Jede Prozedur hat einen eindeutigen Namen. Mit Hilfe dieses Namens wird eine Prozedur aufgerufen.
- Jede Prozedur ist unabhängig. Sie kann nur Werte von anderen Prozeduren übergeben bekommen und diese unabhängig weiterverarbeiten.
- ... können einen Wert an das aufrufende Programm zurückgeben. Diese Art von Prozeduren werden als Funktionen bezeichnet.

- Eine Ereignisprozedur reagiert auf das Verlassen eines Steuerelements. Die Daten werden eingelesen und an eine Funktion zur Auswertung weitergereicht.
- Die Funktion überprüft die Daten auf Plausibilität. Falls die Daten korrekt sind, wird true zurück geliefert und andernfalls false.
- Eine weitere Prozedur spezifiziert den Fehler und gibt eine Meldung an den Nutzer heraus.

- Die Aufgabenstellung wird in kleinere unabhängige Module eingeteilt und somit auch strukturiert
- Der Quellcode lässt sich besser lesen.
- Der Code einer Funktion kann in anderen Programmen wiederverwendet werden.
- Wiederholende Aufgaben werden in einer eigenständigen Prozedur eingeschlossen. Diese Prozedur kann von verschiedenen Stellen im Programm aufgerufen werden.
- Fehler lassen sich schneller finden, weil der Code nur an einer Stelle bearbeitet werden muss.
- Veränderungen lassen sich einfacher vornehmen und testen, weil nur Codefragmente betroffen sind.

- In einer Prozedur wird eine bestimmte Aufgabe gelöst. Der Nutzer der Prozedur muss nicht wissen, wie die Aufgabe gelöst wird. Dem Nutzer genügt es zu wissen, wie die Prozedur aufgerufen wird. Die Prozedur ist eine Blackbox.
- Der Prozedur können Werte übergeben werden. Dem Nutzer der Prozedur muss die Art / Typ der Werte bekannt sein. Einige Prozeduren besitzen keine Eingabeparameter.



```
Sub datenPlz()  
  Dim varPLZ As String  
  Dim fehler As Integer  
  
  varPLZ = plz.Text  
  
  fehler = pruefe(varPLZ)  
  
  If fehler <> 0 Then  
    Select Case fehler  
      Case fehlercode.laenge  
        MsgBox ("Fehlerhafte Länge")  
        plz.ForeColor = RGB(255, 0, 0)  
      Case fehlercode.notNumeric  
        MsgBox ("Falsche Zeichen")  
        plz.ForeColor = RGB(255, 0, 0)  
    End Select  
  Else  
    plz.ForeColor = RGB(0, 0, 0)  
  End If  
End Sub
```

```
Private Sub plz_BeforeUpdate(Cancel As Integer)  
  datenPlz  
End Sub
```

Bezeichnung plz



wird geändert und
das Steuerelement
verlassen

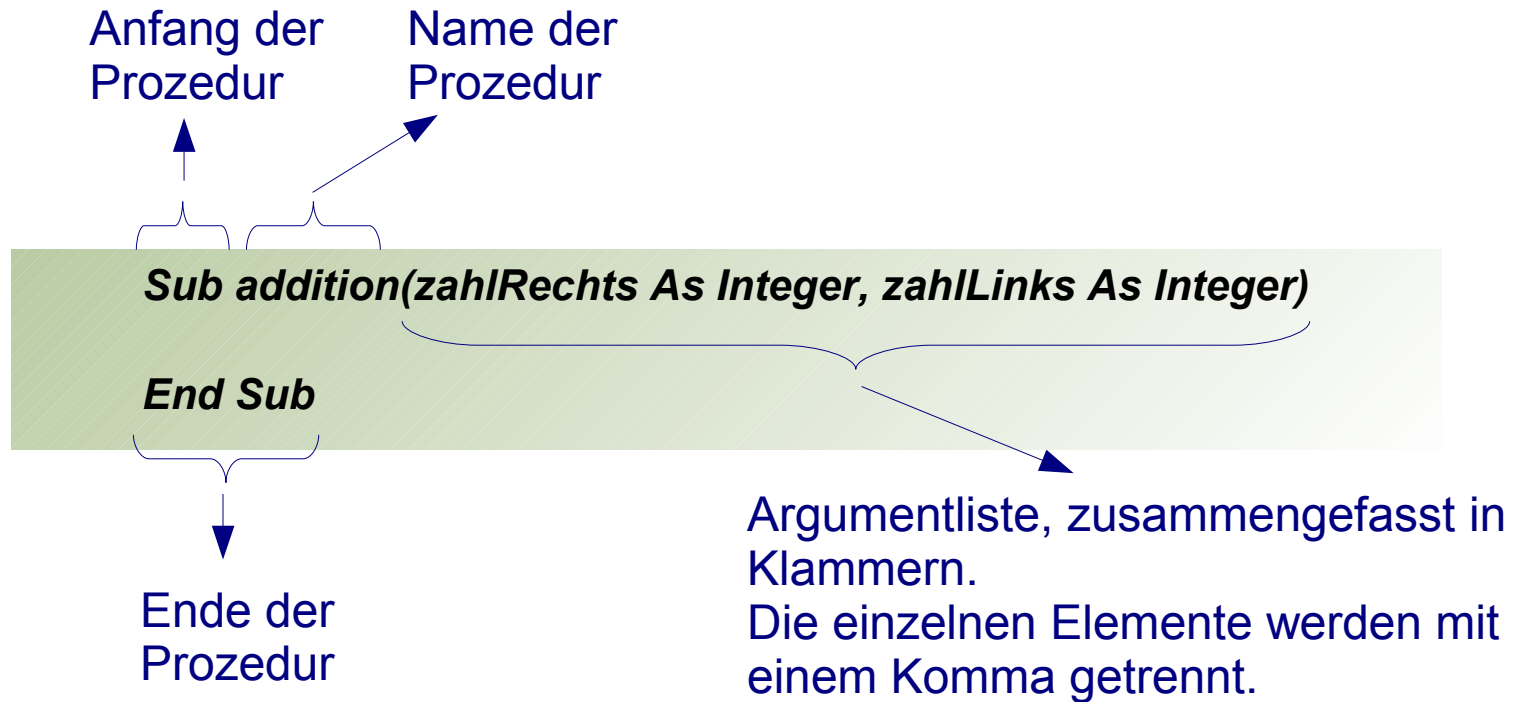
```
'Vor der Aktualisierung  
Private Sub plz_BeforeUpdate(Cancel As Integer)  
    datenPlz  
End Sub
```

Aufruf einer Sub-
Routine

Hole Daten aus dem Steuerelement plz		
Starte Überprüfung auf Fehler		
Fehler vorhanden?		
Fehlercode		Schriftfarbe = Schwarz
Länge?	Nicht Numerisch?	
Fehlermeldung ausgeben	Fehlermeldung ausgeben	
Schriftfarbe = Rot	Schriftfarbe = Rot	

- Prozeduren selber bestehen aus
 - ... dem Prozedurkopf, der die Schnittstelle zum Aufruf darstellt.
 - ... dem Prozedurrumpf, der die Variablen und Anweisungen enthält, die zur Funktion gehören.
- Prozeduren selber dürfen nicht verschachtelt werden. Eine Prozedur kommt nicht innerhalb einer anderen Prozedur vor. Eine Prozedur kann nur innerhalb einer Prozedur aufgerufen werden.

```
Sub addition(zahlRechts As Integer, zahlLinks As Integer) } Prozedurkopf  
Dim summe As Integer  
  
summe = zahlLinks + zahlRechts } Prozedurrumpf  
  
End Sub
```



- ... beginnen immer mit einem Buchstaben.
- ... sind kürzer als 256 Zeichen.
- ... dürfen im Namen kein Leerzeichen, Punkt, Ausrufezeichen, Bindestrich, oder @, &, \$ oder # haben.
- ... sollten keine Umlaute, Satzzeichen oder Sonderzeichen enthalten.
- Es gibt keine Unterscheidung zwischen Groß- und Kleinschreibung.
- Schlüsselwörter oder vordefinierte Prozedurnamen dürfen nicht genutzt werden.
- ... dürfen nur einmal vorkommen.
- ... sollten die Aufgabe der Funktion widerspiegeln.

- Wenn die Aufgabe der Prozedur beschrieben werden soll, werden häufig Substantive genutzt.
 - Mittelwert(), Integral(), TuermeVonHanoi(), FehlerAusgeben(), setExponent(), Auswaehlen().
- Falls logische Operationen beschrieben werden, nutzen Prozedurnamen als erstes Wort Adjektive.
 - kleinsterWert(), groessterMesswert().

Sub bezeichnung (var1 As Datentyp, var2 As Datentyp)

- In den runden Klammern, die der Bezeichnung folgen, werden alle Argumente zusammengefasst.
- Argumente sind Variablen, die an die Prozedur übergeben werden.
- Die einzelnen Argumente werden in der Liste durch Kommata getrennt.
- Die Liste kann beliebig viele Argumente aufnehmen.
- Jedes Argument
 - ... hat einen eindeutigen Namen.
 - ... bekommt mit *As* einen Datentyp zugewiesen, um den Wert korrekt zu interpretieren.

addition
Call addition()

- Sie können die Prozedur direkt mit dem Namen aufrufen. Der Prozedur können keine Argumente übergeben werden. Das Setzen von Klammern erzeugt einen Fehler.
- Mit Hilfe des Schlüsselwortes *Call* (Ruf) kann eine Prozedur aufgerufen werden. Die Klammern für die Argumentliste können, müssen aber nicht gesetzt werden. Um Fehler zu vermeiden, sollte dieses Art des Aufrufs genutzt werden.

Sub addition(zahlRechts As Integer, zahlLinks As Integer)

Call addition(wert, zahl)

- Die Argumentliste und die zu übergebene Parameterliste haben die gleiche Anzahl von Elementen.
- Der Wert des ersten zu übergebenen Parameters wird an das erste Argument übergeben und so weiter.
- Der zu übergebene Parameter und das dazugehörige Argument haben den gleichen Datentyp.

```
Sub addition(ByRef zahlRechts As Integer, ByRef zahlLinks As Integer)
```

```
Dim summe As Integer
```

```
summe = zahlLinks + zahlRechts
```

```
End Sub
```

```
Sub main()
```

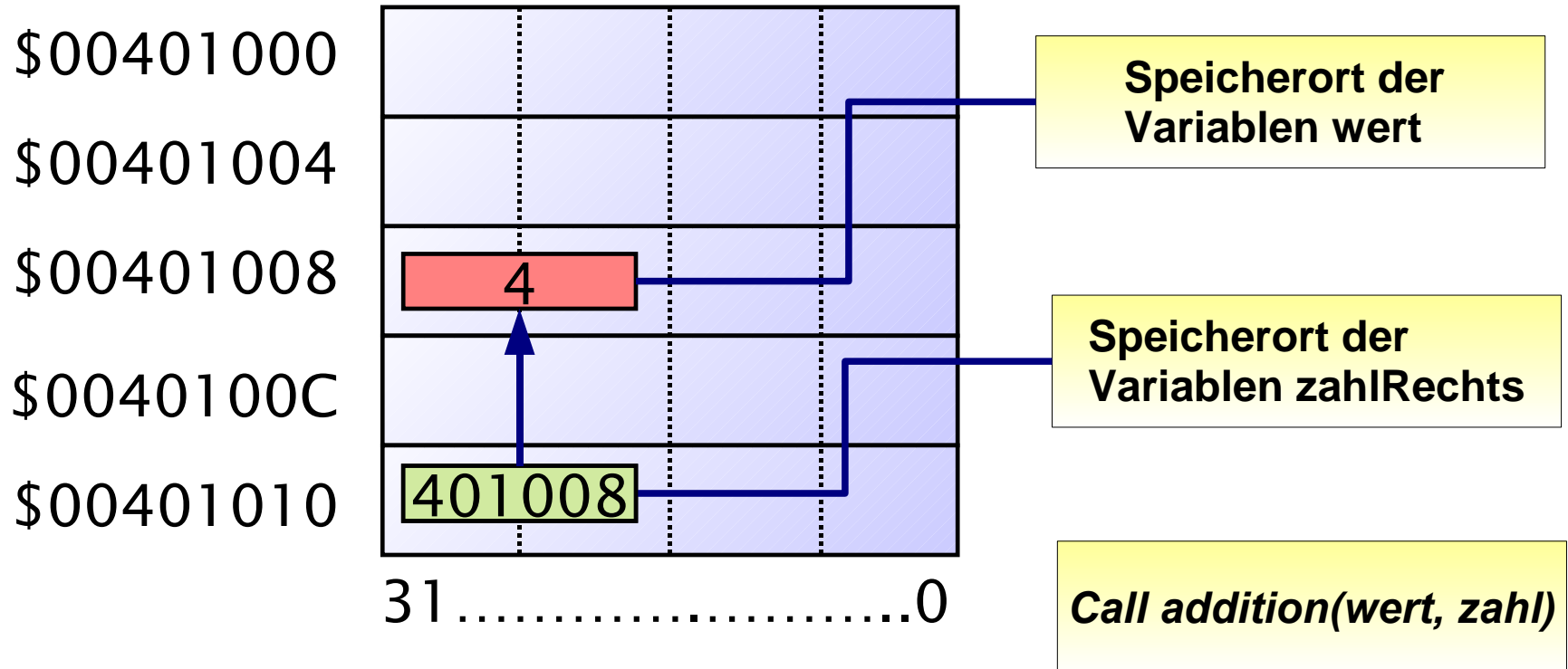
```
Const wert = 4
```

```
Const zahl = 5
```

```
Call addition(wert, zahl)
```

```
End Sub
```

- Jeder Variablenname kennzeichnet den Anfang eines bestimmten Speicherblocks. Die Größe des Speicherblocks ist abhängig vom ausgewählten Datentyp der Variablen.
- ... übergibt einen Verweis auf die Variable und damit auf den dazugehörigen Speicherblock.
- Der Wert der Variablen, der innerhalb dieses Speicherblocks steht, kann durch die aufgerufene Prozedur geändert werden. Dadurch ist es schwierig zu kontrollieren, wer wann wo eine Variable geändert hat.
- ... wird standardmäßig bei allen Variablen genutzt, die an eine Prozedur übergeben werden.



```
Sub addition(ByVal zahlRechts As Integer, ByVal zahlLinks As Integer)
```

```
Dim summe As Integer
```

```
summe = zahlLinks + zahlRechts
```

```
End Sub
```

```
Sub main()
```

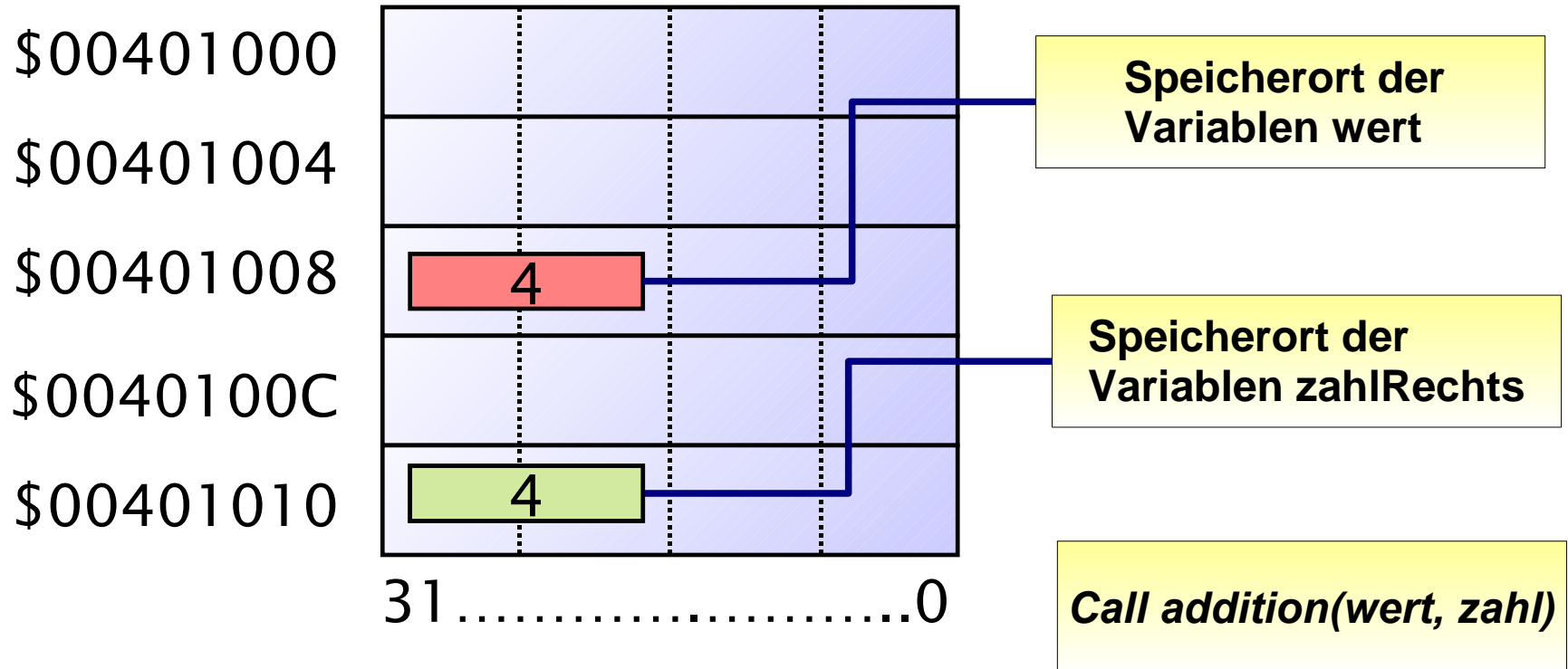
```
Const wert = 4
```

```
Const zahl = 5
```

```
Call addition(wert, zahl)
```

```
End Sub
```

- Der Wert einer Variablen wird an eine Prozedur übergeben.
- Es wird eine Kopie der Variablen erzeugt, die an die Prozedur übergeben wird.
- Die Prozedur hat nur Zugriff auf die Kopie, aber nicht auf das Original. Das heißt der Wert der Ursprungsvariablen kann nicht von der aufgerufenen Prozedur verändert werden.
- In dem vorhergehenden Beispiel
 - ... wird der Wert von der Variablen *wert* in die Variable *zahlRechts* kopiert. Ähnlich wie die Anweisung *zahlRechts = wert*.
 - ... wird der Wert von der Variablen *zahl* in die Variable *zahlLinks* kopiert. Ähnlich wie die Anweisung *zahlLinks = zahl*.



- ... können
 - ... innerhalb einer Prozedur definiert werden. Diese Variablen werden als lokal bezeichnet. Sie sind nur innerhalb dieser Prozedur gültig und bekannt.
 - ... am Anfang eines Moduls definiert werden. Diese Variablen werden als global bezeichnet. Diese Variablen sind im gesamten Modul bekannt.
 - ... als statische Variablen definiert werden. Diese Variablen behalten auch nach dem Verlassen der Prozedur ihren Wert.
- ... sollten so lokal wie möglich und so global wie nötig angelegt werden.

- Alle lokalen Variablen in dem neben stehenden Beispiel sind mit *kursiver blauer* Schrift geschrieben.
- Lokale Variablen werden innerhalb einer Prozedur definiert.
- Sobald eine Prozedur betreten wird, wird die Variable immer wieder neu angelegt. Wird der Anweisungsblock verlassen, wird die Variable aus dem Speicher gelöscht. Das heißt, eine Variable lebt nur so lange wie die Prozedur abgearbeitet wird.
- Die Variable ist nur innerhalb einer Prozedur bekannt.

```
Sub GetFaktorMal (ByVal zahl As Integer)
```

```
Const faktor = 0.3
```

```
Dim result As Double
```

```
result = zahl * faktor
```

```
End Sub
```

```
Sub main()
```

```
Const wert = 4
```

```
Dim result As Integer
```

```
Call GetFaktorMal(wert)
```

```
result = wert + wert
```

```
End Sub
```

- In jeder Prozedur kommt ein Variablenname exakt einmal vor.
- Jeder Variablenname ist ein Stellvertreter für eine Speicherzelle im Arbeitsspeicher. Dieser Speicher wird bei der Deklaration einer Variablen angefordert. Sobald die Prozedur verlassen wird, wird der Speicherbereich automatisch frei gegeben.
- Aus diesen Grund können in verschiedenen Prozeduren identische Namen genutzt werden. Zur besseren Lesbarkeit und Verständlichkeit können unterschiedliche Namen verwendet werden. Beispiel:
 - mainFaktor in dem Hauptprogramm und potenzierenFaktor in der Funktion potenzieren.
 - preisBrutto und preisNetto für die Kennzeichnung von Beträgen mit und ohne Steuern.

- ... werden außerhalb einer Prozedur definiert. Meist werden Sie am Anfang eines Moduls definiert.
- ... gelten für das gesamte Modul.
- ... können von allen Prozeduren in einem Modul genutzt werden.
- Eine lokale Variable, die den selben Namen wie eine globale Variable hat, hat Vorrang vor der globalen Variablen.
- Überlegen Sie sich genau, welche Variablen global gehalten werden sollten. Variablen, die selten oder nur in bestimmten Funktionen benötigt werden, sollten lokal gehalten werden.

```
Const faktor = 0.3
```

```
Sub GetFaktorMal (ByVal zahl As Integer)
```

```
Dim result As Double
```

```
result = zahl * faktor
```

```
End Sub
```

```
Sub main()
```

```
Const wert = 4
```

```
Dim result As Integer
```

```
Call GetFaktorMal(wert)
```

```
result = wert + wert
```

```
End Sub
```

Hier wird für ein Modul eine globale Variable definiert. Die Variable kann für alle Prozeduren genutzt werden.

- Bei einem Aufruf einer Prozedur werden für alle lokalen Variablen neue Speicherzellen reserviert. Nach dem Ende der Prozedur wird der Speicher wieder freigegeben. Die lokale Variable ist innerhalb der Prozedur existent und sichtbar.
- Globale Variablen werden am Anfang eines Moduls deklariert. Für globale Variablen wird am Anfang eines Moduls Speicher bereitgestellt und am Ende des Moduls freigegeben. Sie sind während der Laufzeit des Moduls für alle Prozeduren existent.
- Wenn eine lokale Variable in einer Prozedur und eine globale Variable identische Namen besitzen, überdeckt die lokale Variable die globale Variable in dieser Prozedur. Die globale Variable ist in dieser Prozedur nicht sichtbar.

- ... beginnen mit dem Schlüsselwort *Static* statt *Dim*.
- ... bezeichnen immer dieselbe Speicherzelle, egal wie oft eine Prozedur aufgerufen wird.
- ... sind nur innerhalb der Prozedur sichtbar, in der sie deklariert werden.
- ... behalten auch nach dem Verlassen der Prozedur den zugewiesenen Wert.
- ... können global oder lokal sein.

```
Sub PlusEins()  
  Static count As Integer  
  
  count = count + 1  
  
End Sub  
  
Sub main()  
  Call PlusEins()  
  Call PlusEins()  
  Call PlusEins()  
End Sub
```

Modul

Dim var As...
Static var As ...

Prozedur

Dim var As ...

Prozedur

Static var As ...

Prozedur

Dim var As ...

Prozedur

Dim var As ...
Dim var As

```
Sub main()  
  Dim messwerte() As Double  
  
  SetMesswert(messwerte())  
End Sub
```

In der Argumentliste sind
Felder immer vom Datentyp
Variant.

Ist der übergebene
Parameter ein Array
und vom gewünschten
Typ?

```
Sub SetMesswert(ByVal messung As Variant)  
  Const wert = 4  
  Dim max As Integer  
  Dim errCode As Integer  
  
  errCode = 0  
  If Not(IsArray(messung)) Then  
    errCode = fehlerCode.NotArray  
    GoTo fehler  
  
  End If  
  If (TypeName(messung) <> "Double()") Then  
    errCode = fehlerCode.NotType  
    GoTo fehler  
  
  End If  
End Sub
```

Das letzte Argument ist optional. Für das Argument muss kein Parameter übergeben werden. Alle nachfolgenden Argumente müssen auch optional sein

```
Sub GetError(fehler As Integer, Optional messFeld As Variant)
```

```
    Select Case fehler
```

```
        Case fehlercode.NotArray
```

```
            errorMsg = "Kein Array vorhanden"
```

```
            Exit Sub
```

```
        Case fehlercode.NotType
```

```
            errorMsg = "Falscher Datentyp vorhanden"
```

```
            Exit Sub
```

```
        Case 9
```

```
            errorMsg = "Array nicht dimensioniert"
```

```
            Redim messFeld(0 To 0)
```

```
            Exit Sub
```

```
    End Select
```

```
End Sub
```

```
Sub SetMesswert(ByVal messung As Variant)
```

```
...
```

```
m_subWrite:
```

```
  Err.Clear
```

```
  On Error GoTo m_fehler
```

```
  if (Ubound(messung) >= 0) Then
```

```
    max = Ubound(messung)
```

```
    messung(max) = wert
```

```
    Redim Preserve messung(max + 1)
```

```
  End If
```

```
m_fehler:
```

```
  If (errCode > 0) Then
```

```
    Call GetError(errCode)
```

```
  Else
```

```
    Call GetError(Err.Number, messung)
```

```
    Goto m_subWrite
```

```
  End If
```

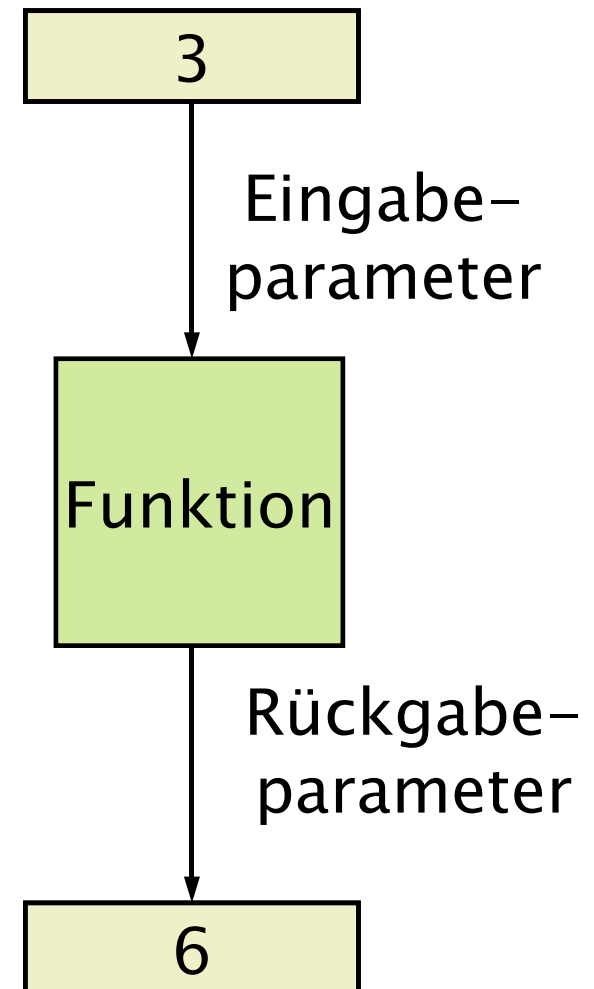
```
End Sub
```

Hier wird die Prozedur ohne optionalen Parameter aufgerufen.

Hier wird die Prozedur mit allen möglichen Parametern aufgerufen.

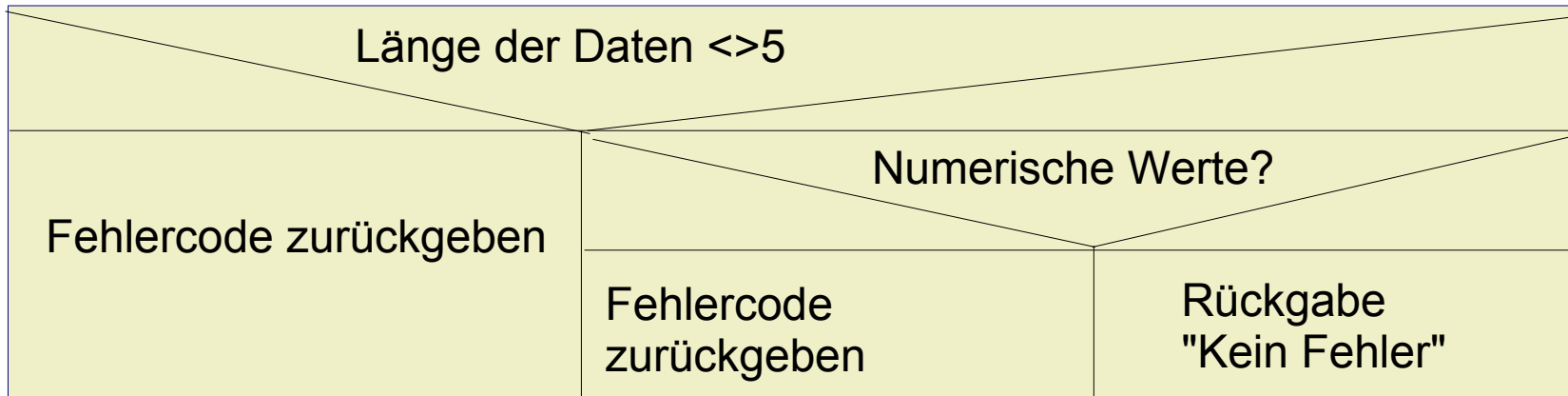
- ... sind kleine Unterprogramm (Subroutinen) mit denen Teilprobleme einer größeren Aufgabe gelöst werden können.
- ... fassen Anweisungen, die eine bestimmte Aufgabe lösen, zu einem Block zusammen.
- ... sind eine Abfolge von VBA-Befehlen zu einem bestimmten Thema
- ... werden zeilenweise abgearbeitet.
- ... geben einen Wert an den Aufrufer zurück. Für die Rückgabe können alle Standard-Datentypen genutzt werden.

- In einer Funktion wird eine bestimmte Aufgabe gelöst. Der Nutzer der Funktion muss nicht wissen, wie die Aufgabe gelöst wird. Dem Nutzer genügt es zu wissen, wie die Funktion aufgerufen wird. Die Funktion ist eine Blackbox.
- Der Funktion können Werte übergeben werden. Dem Nutzer der Funktion muss die Art / Typ der Werte bekannt sein. Einige Funktionen besitzen keine Eingabeparameter.
- Die Lösung der Aufgabe kann an den Aufrufer zurückgegeben werden.



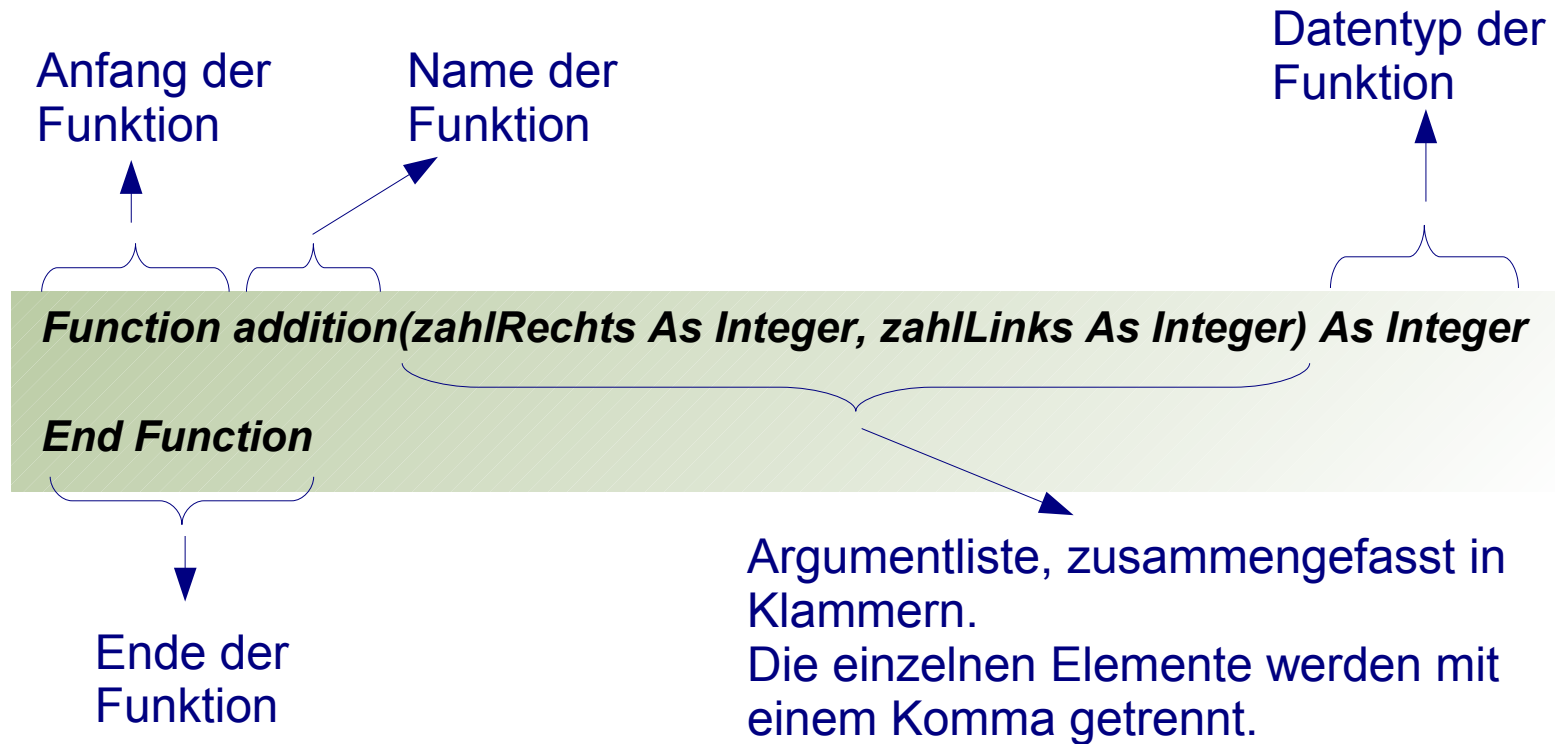
```
Sub datenPlz()  
  Dim varPLZ As String  
  Dim fehler As Integer  
  
  varPLZ = plz.Text  
  
  fehler = pruefe(varPLZ)  
  
  If fehler <> 0 Then  
    Select Case fehler  
      Case fehlercode.laenge  
        MsgBox ("Fehlerhafte Länge")  
        plz.ForeColor = RGB(255, 0, 0)  
      Case fehlercode.notNumeric  
        MsgBox ("Falsche Zeichen")  
        plz.ForeColor = RGB(255, 0, 0)  
    End Select  
  Else  
    plz.ForeColor = RGB(0, 0, 0)  
  End If  
End Sub
```

```
Function pruefe(daten As String) As Integer  
  If Len(daten) <> 5 Then  
    pruefe = fehlercode.laenge  
    Exit Function  
  End If  
  
  If Not IsNumeric(daten) Then  
    pruefe = fehlercode.notNumeric  
    Exit Function  
  End If  
  
  pruefe = 0  
End Function
```



- Funktionen selber bestehen aus
 - ... dem Funktionskopf, der die Schnittstelle zum Aufruf darstellt.
 - ... dem Funktionsrumpf, der die Variablen und Anweisungen enthält, die zur Funktion gehören.
- Funktionen selber dürfen nicht verschachtelt werden. Eine Funktion kommt nicht innerhalb einer anderen Funktion oder Prozedur vor. Eine Funktion kann nur innerhalb einer Prozedur oder Funktion aufgerufen werden.

```
Function addition(zahlRechts As Integer, zahlLinks As Integer) As Integer } Funktions-  
    Dim summe As Integer } kopf  
  
    summe = zahlLinks + zahlRechts } Funktionsrumpf  
  
    addition = summe  
End Function
```



Function bezeichnung(var1 As Typ, var2 As Typ) As Typ

- Am Ende des Funktionsrumpfs wird mit Hilfe von *As* ein Datentyp für die Funktion festgelegt.
- Der Datentyp legt den Bauplan für den Wert fest, den die Funktion zurückgibt.
- Der Datentyp der Funktion legt automatisch den Datentyp des Rückgabewertes fest.
- Es kann jeder in VBA bekannter Datentyp für die Rückgabe genutzt werden.
- Eine Funktion kann immer nur ein Wert zurückgeben.

```
summe = addition(wert1, wert2)  
summe = addition()
```

- Eine Funktion wird direkt mit dem Namen aufrufen.
- In den runden Klammern stehen die an die Funktion zu übergebenen Parameter. Wenn die Klammern leer sind, werden keine Parameter übergeben.
- Mit Hilfe des Gleichheitszeichen wird der Rückgabewert der Funktion der Variablen *summe* zugewiesen.
 - Die Variable muss den gleichen Datentyp haben wie die Funktion. Anderenfalls muss der Rückgabewert konvertiert werden.
 - Diese Variable kann innerhalb der aufrufenden Funktion oder Prozedur verändert oder in einem Ausdruck genutzt werden.

```
Function addition(zahlRechts As Integer, zahlLinks As Integer) As Integer
    Dim summe As Integer

    summe = zahlLinks + zahlRechts

    addition = summe ' Rückgabewert
End Function
```

- Dem Funktionsnamen wird mit Hilfe des Gleichheitszeichen ein Wert zugewiesen.
- Die Funktion wird aber nicht automatisch mit der Zuweisung beendet.
 - Mit Hilfe der Anweisung *End Function* wird die Funktion beendet.
 - Mit der Anweisung *Exit Function* wird die Funktion vorzeitig verlassen.
- Der Wert, der dem Funktionsnamen zugewiesen wird, muss den gleichen Datentyp besitzen wie die Funktion.

- ... oder integrierte Funktionen.
- ... sind für
 - ... mathematische Berechnungen wie zum Beispiel Winkelberechnungen oder Berechnungen im Bereich Finanzmathematik vorhanden.
 - ... sind für das Arbeiten mit Datums- und Zeitwerten vorhanden.
 - ... sind für das Arbeiten mit Strings vorhanden.
 - ... sind für das Arbeiten mit Dateien vorhanden.
- ... werden in den Hilfeseiten unter ihrer englischen Bezeichnung beschrieben.
- ... werden genauso wie selbst definierte Funktionen aufgerufen.

```
tage = DateDiff("d", Now(), #12/31/2008#)  
tage = DateDiff("d", Now(), "31.12.")
```

- Die Funktion *Now()* liefert das aktuelle Systemdatum zurück.
- Mit Hilfe der Funktion *DateDiff()* werden zwei Datumswerte voneinander abgezogen.
 - Der erste Parameter der Funktion gibt die Einheit des Zeitraums zwischen zwei Datumswerten an. Hier wird die Differenz in Tagen zurückgegeben.
 - Die nächsten zwei Parameter sind die Datumswerte, die voneinander abgezogen werden. Folgende Anweisung ist gleichbedeutend $\text{Parameter3} - \text{Parameter2}$.
- Sie können die Datumswerte
 - ... mit Hilfe einer Funktion berechnen.
 - ... als Konstanten im Datumsformat oder Stringformat angegeben. Falls kein Jahr angegeben wird, wird automatisch das aktuelle Jahr genutzt.
 - ... als Variablen vom Datentyp *Date* angegeben.

```
Dim datum As Date
```

```
datum = DateAdd("m", 2, "1.11")
```

```
datum = DateAdd("m", -2, "1.11")
```

- Mit Hilfe von *DateAdd()* wird ein bestimmter Zeitraum zu einem Datum dazu addiert.
 - Der erste Parameter der Funktion gibt die Einheit des Zeitraums an, die zu dem angegebenen Datum hinzu addiert werden soll. Hier wird eine bestimmte Anzahl von Monaten addiert.
 - Der nächste Parameter gibt an, wie viele Einheiten zu dem Datum hinzu addiert werden sollen. Hier werden einmal zwei Monate hinzu addiert und im zweiten Beispiel zwei Monate abgezogen.
 - Der dritte Parameter ist ein Datumswert, welches um den angegebenen Zeitraum erhöht oder vermindert wird.

Dim which As Integer

which = DatePart("q", "1.11")

which = DatePart("yyyy", "1.11")

- Mit Hilfe von *DatePart()* wird ein bestimmtes Element des Datumswertes heraus gefiltert
 - Der erste Parameter der Funktion gibt die Einheit des Elements an, welches heraus gefiltert werden soll. Mit Hilfe des Parameters *q* wird das Quartal berechnet. Mit Hilfe von *"yyyy"* wird das Jahr zurück gegeben.
 - Der nächste Parameter ist ein Datumswert, welches in seine Elemente zerlegt wird.

```
Dim text As String  
Dim laenge As Integer
```

```
text = "Hello World"  
laenge = Len(text)
```

- Mit Hilfe von *Len()* wird die Anzahl der Zeichen in einem String zurückgeliefert.
- Die vordefinierte Funktion liefert die Länge eines Strings zurück.
- Falls der String leer ist, wird eine Länge von Null zurückgeliefert.

- *Left()* liefert eine bestimmte Anzahl von Zeichen, beginnend am Anfang des Strings.
- *Right()* liefert eine bestimmte Anzahl von Zeichen, beginnend am Ende des Strings.
- *Mid()* liefert ein Teilstring, beginnend ab einer Startposition (2. Parameter) von einer bestimmten Länge (3. Parameter). Falls keine Startposition angegeben wird, wird eine Startposition 1, beginnend am Anfang des Strings angenommen.

```
Dim text As String  
Dim tmp As String  
Dim laenge As Integer
```

```
text = "Hello World"
```

```
tmp = Left(text, 3)
```

```
tmp = Right(text, 3)
```

```
tmp = Mid(text, 1, 5)
```

```
tmp = Mid(text, 7, 5)
```

```
Dim text As String  
Dim pos As Integer  
  
text = "Eisbären leben in der Arktis"  
pos = Instr(4, text, "leben")
```

- Mit Hilfe der Funktion *Instr()* kann innerhalb eines Strings nach einem bestimmten Muster gesucht werden.
 - Der erste Parameter ist optional. Hier kann der Beginn der Suche spezifiziert werden. In diesem Beispiel beginnt die Suche am vierten Zeichen.
 - Als zweiter Parameter wird der zu durchsuchende Text übergeben.
 - Als dritter Parameter wird das Suchmuster angegeben. Hier wird nach dem Begriff "leben" im Satz "Eisbären leben in der Arktis" gesucht.
- Die Funktion liefert eine Position als Integer-Wert zurück. Es wird der Beginn des Suchmusters im zu durchsuchenden Text zurückgeliefert. Anschließend bricht die Funktion ab. Falls das Muster nicht gefunden wird, wird Null zurückgegeben.

```
Const komma = ","  
Dim text As String  
Dim tmpText As String  
Dim pos As Integer  
Dim tmpPos As Integer  
  
text = "Eisbär, Braunbär, Nasenbär, Grizzlybär"  
tmpPos = 1  
pos = Instr(text, komma)  
  
If (pos > 0) Then  
    Do  
        tmpZeichen = Mid(text, tmpPos, (pos - tmpPos))  
        tmpPos = pos + 1  
        pos = Instr(tmpPos, text, komma)  
    Loop While (pos > 0)  
  
    tmpZeichen = Right(text, Len(text) - tmpPos)  
  
End If
```

```
Const komma = ","  
Dim text As String  
Dim tmpText As String  
Dim pos As Integer  
Dim tmpPos As Integer
```

```
text = "Eisbär, Braunbär, Nasenbär, G..."  
tmpPos = 1  
pos = Instr(text, komma)
```

```
If (pos > 0) Then  
  Do
```

```
    tmpZeichen = Mid(text, tmpPos, (pos - tmpPos))  
    tmpPos = pos + 1  
    pos = Instr(tmpPos, text, komma)
```

```
  Loop While (pos > 0)
```

```
  tmpZeichen = Right(text, Len(text) - tmpPos)
```

```
End If
```

Ist ein Komma in dem Text vorhanden?
Wenn ja, wird diese Position gespeichert und die Anweisungen in der if-Anweisung bearbeitet.

Die Zeichenkette wird anhand der Kommata in die einzelnen Listenelemente zerlegt.

Jedes Element geht von dem davor gefundenen Komma (tmpPos) bis zu dem nächsten Komma (pos).

```
text = "Eisbär, Braunbär, Grizzlybär"  
tmpPos = 1  
pos = Instr(text, komma)
```

```
If (pos > 0) Then  
  Do
```

```
    tmpZeichen = Mid(text, tmpPos, (pos - tmpPos))  
    tmpPos = pos + 1  
    pos = Instr(tmpPos, text, komma)
```

```
  Loop While (pos > 0)
```

```
  tmpZeichen = Right(text, Len(text) - tmpPos)
```

```
End If
```

Anschließend wird die aktuelle Position zwischengespeichert und das nächste Komma gesucht.

```
Const komma = ","  
Dim text As String  
Dim tmpText As String  
Dim pos As Integer  
Dim tmpPos
```

Das letzte Element wird zwischengespeichert. Das letzte Element geht vom letzten Komma bis zum Ende der Zeichenkette. Was passiert, wenn das letzte Zeichen in der Zeichenkette ein Komma ist?

```
    tmpZeichen = Mid(text, tmpPos, (pos - tmpPos))  
    tmpPos = pos + 1  
    pos = Instr(tmpPos, text, komma)  
Loop While (pos > 0)  
  
    tmpZeichen = Right(text, Len(text) - tmpPos)  
  
End If
```

```
Const laenge = 3  
Dim text As String  
Dim tmpText As String  
Dim feldText() As String  
Dim element As Variant
```

Für die Funktion `split()` muss ein Array ohne Dimension definiert werden.

```
text = "Eisbär, Braunbär, Nasenbär, Grizzlybär"
```

```
feldText = Split(text, ", ", laenge)
```

`Split()` wird der zu trennende Text übergeben sowie ein Trennzeichen. Optional kann noch die Anzahl der Elemente des Arrays angegeben werden.

```
For Each element in feldText  
    Debug.Print element  
Next element
```

```
tmpText = Join(feldText, ", ")  
Debug.Print tmpText
```

`Join()` wird ein Array angegeben, deren Elemente mit Hilfe des Trennzeichens zu einem String zusammengefügt werden.

```
Dim text As String  
Dim tmpText As String  
text = "Eisbär, Braunbär, Nasenbär, Grizzlybär"  
tmpText = Replace(text, "ä", "ae")
```

- Mit Hilfe der Funktion *Replace()* können Teile des Strings oder die Zeichenkette vollständig ersetzt werden.
- Folgende Parameter werden übergeben:
 - Als erstes Parameter wird ein String übergeben.
 - Als zweiter Parameter wird der zu ersetzende Text übergeben.
 - Als dritter Parameter wird der Text übergeben, der den zweiten Parameter ersetzt. Hier werden alle "ä" im Text gegen ein "ae" getauscht.
- Optional können noch folgende Parameter angegeben werden:
 - Als vierter Parameter kann eine Startposition für die Suchen & Ersetzen-Funktion vorgegeben werden.
 - Als fünfter Parameter kann die Anzahl der zu ersetzenden Zeichen eingegeben werden.
 - Als sechster Parameter kann für die Suche eine Vergleichsart bestimmt werden.

```
Dim text As String  
Dim tmpText As String
```

```
text = "Eisbär, Braunbär, Nasenbär, Grizzlybär"  
tmpText = LCase(text)  
tmpText = UCase(text)
```

- Mit Hilfe der Funktion *UCase()* werden alle Kleinbuchstaben in Großbuchstaben umgewandelt.
- Mit Hilfe der Funktion *LCase()* werden alle Großbuchstaben in Kleinbuchstaben umgewandelt.

```
Dim text As String  
Dim tmpText As String  
  
text = " Eisbär, Braunbär, Nasenbär, Grizzlybär "  
tmpText = LTrim(text)  
tmpText = Rtrim(text)  
tmpText = Trim(text)
```

- Mit Hilfe der Funktion *LTrim()* werden alle Leerzeichen am Beginn der Zeichenkette entfernt.
- Mit Hilfe der Funktion *RTrim()* werden alle Leerzeichen am Ende der Zeichenkette entfernt.
- Mit Hilfe der Funktion *Trim()* werden alle Leerzeichen am Beginn und Ende der Zeichenkette entfernt.

Beispiel "Formatierungen"

```
Dim text As String
```

```
Dim tmpText As String
```

```
text = Date
```

```
tmpText = Format(text, "Long Date")           ' Dienstag, 9. April 2007
```

```
text = Time
```

```
tmpText = Format(text, "Short Time")         ' 09:37
```

```
text = "12345,64454"
```

```
tmpText = Format(text, "Currency")          ' 12.345,64 €
```

```
text = 0
```

```
tmpText = Format(text, "Yes/No")            ' Nein
```

```
text = #12/07/2007#
```

```
tmpText = Format(text, "dd.mmmm.yyy")       ' 07. Dezember. 2007
```

```
text = 1230.3
```

```
tmpText = Format(text, "##,##0.00")         ' 1.230.30
```

- Mit Hilfe der Funktion `Format()` können Daten für die Ausgabe formatiert werden.
- Hier können vordefinierte Formate oder benutzerdefinierte Formate genutzt werden.
- Der Funktion werden folgende Parameter übergeben:
 - Als erster Parameter werden die zu formatierten Daten übergeben.
 - Als zweiter Parameter wird ein Format als String übergeben. Die verschiedenen Möglichkeiten sind in der Hilfe aufgeführt.

```
Private Sub plz_BeforeUpdate (Cancel As Integer)  
End Sub
```

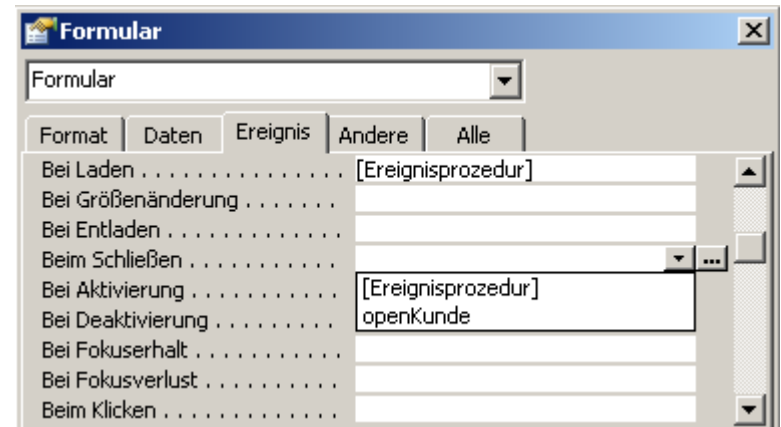
```
Private Sub Form_Load()  
End Sub
```

- An ein Fenster sind Funktionen gekoppelt, die ausgelöst werden, wenn der Anwender zum Beispiel in das Fenster klickt oder ein Fenster in der Größe ändert.
- Ein Mausklick etc. wird als Ereignis bezeichnet.
- Für Ereignisse werden in VBA die englischsprachigen Begriffe genutzt.

Private Sub plz_BeforeUpdate (Cancel As Integer)

- Eine Ereignisprozedur ist immer mit dem Zugriffsrecht *Private* ausgestattet. Die Prozedur kann nur innerhalb ihres Formular- oder Berichtsmoduls aufgerufen werden.
- Mit Hilfe von *Sub* wird die Prozedur eingeleitet.
- Der Name der Prozedur setzt sich immer aus dem Namen des Objekts und dem Ereignis zusammen. Hier wird zum Beispiel das Ereignis "Vor der Aktualisierung" des Objekts *plz* abgefangen.
- Die Anzahl der zu übergebenden Argumente ist immer von der Art des Ereignisses abhängig.

- Markieren Sie das gewünschte Objekt (Steuerelement, Formular oder Bericht) in der Entwurfsansicht.
- Öffnen Sie das dazugehörige Eigenschaftfenster.
- Wechseln Sie auf die Registerkarte Ereignis. Hier werden alle Ereignis aufgelistet, auf die das ausgewählte Objekt reagieren kann.
- Setzen Sie den Mauszeiger in das Kombinationsfeld rechts neben der Bezeichnung.
- Öffnen Sie die Liste mit dem schwarzen Pfeil nach unten und wählen den Eintrag [Ereignisprozedur] aus. Der VBA-Editor wird geöffnet und die Prozedur automatisch angelegt.



- Ein Formular- oder Berichtsmodul ist nur vorhanden, wenn mindestens eine Ereignisprozedur über das Eigenschaftfenster angelegt wurde.
- Wählen Sie im Projekt-Explorer ein Formular- oder Berichtsmodul aus.
- Wählen Sie aus dem Kombinationsfeld am linken, oberen Rand des Code-Fensters ein Objekt aus.
- Wählen Sie anschließend aus dem Kombinationsfeld am rechten, oberen Rand des Code-Fensters das gewünschte Ereignis aus. Die Prozedur wird automatisch in das Code-Fenster eingefügt.

Projekt

Fomularmodul

control_VorAktualisierung

form_open

PruefeDaten

Standardmodul

berechne_Daten

GetError

FormatData

- ... fassen Prozeduren, die ein Thema bearbeiten, zusammen.
- ... sind die Kapitel in einem Buch und die Prozeduren die Unterkapitel.
- ... sind eine Sammlung von Deklarationen, Anweisungen und Prozeduren, die im Rahmen eines bestimmten Projekts gespeichert werden.
- ... ermöglichen eine strukturierte Programmierung.
- ... steht mindestens eine Prozedur.

■ Standard-Module

- ... enthalten Code, der nicht an ein Formular oder Bericht gebunden ist.
- ... enthalten häufig Code, der innerhalb des gesamten Projekts von vielen Elementen genutzt wird.
- ... werden über das Objekt Module und dem Symbol Neu im Datenbankfenster erstellt.

■ Microsoft Office Access Klassenmodule

- ... sind mit einem Objekt aus der Microsoft Office Welt verbunden.
- ... sind mit einem bestimmten Formular oder Bericht verbunden.
Klassenmodule für Formulare oder Berichte werden automatisch angelegt, wenn Ereignisprozeduren erstellt werden. Das heißt, das Objekt oder ein darin enthaltenes Element reagiert zum Beispiel auf einen Mausklick.
- Mit Hilfe des Menüs Einfügen – Klassenmodule im VBA-Editor können Sie eigene Objekte definieren.

```
Public Sub GetErrorMsg(fehler As Integer)  
    Dim msg As String  
End Sub
```

```
Private Sub BerechneDatum(fzeitraum As Integer)  
    Dim msg As String  
End Sub
```

■ Das Schlüsselwort *Private*

- ... ist gleichbedeutend mit *Dim*.
- ... darf aber nicht innerhalb von Prozeduren genutzt werden.
- ... kapselt Deklarationen und Anweisungen in einem Modul. Es ist kein Zugriff von außen möglich.

■ Das Schlüsselwort *Public*

- ... darf aber nicht innerhalb von Prozeduren genutzt werden.
- ... stellt Deklarationen und Anweisungen eines Modul anderen Modulen in einem Projekt zur Verfügung. Ein Zugriff von außen ist möglich.
- ... wird genutzt, wenn kein Zugriffsrecht gesetzt ist.