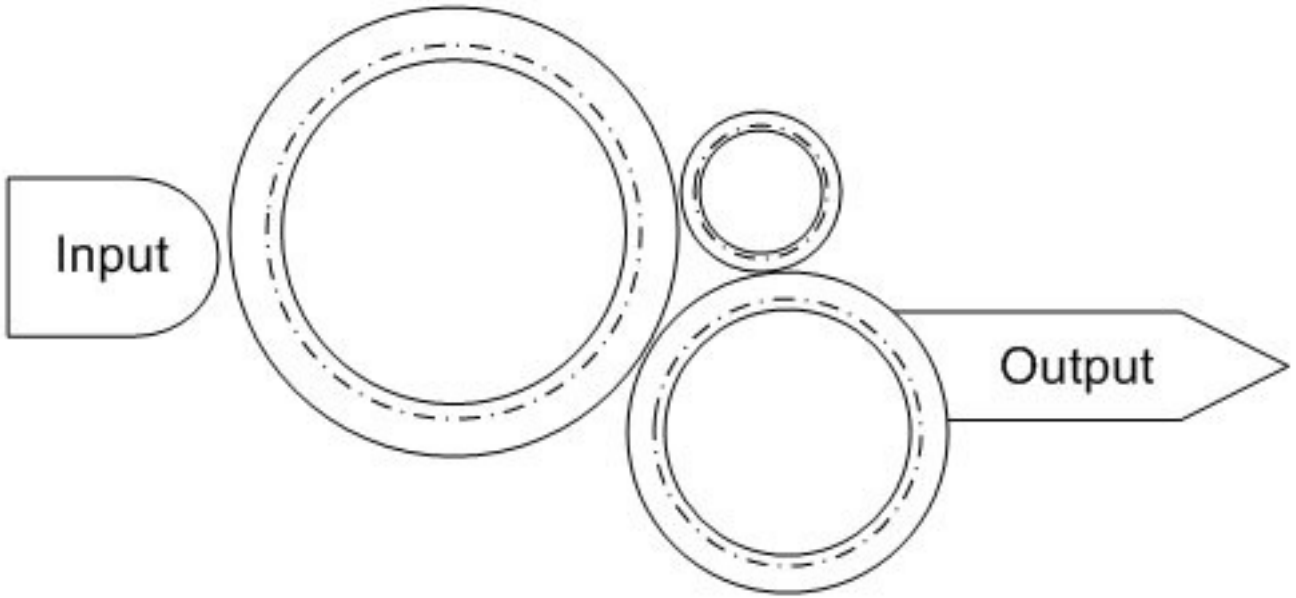
VB.NET - Prozeduren und Funktionen



Unterprogramme (Subroutinen)

- ... lösen Teilprobleme der Gesamtaufgabe.
- ... fassen Anweisungen zu einem Block zusammen, die ein bestimmtes Thema bearbeiten.
- ... sind eine Abfolge von VB-Befehlen und -Anweisungen, die zeilenweise abgearbeitet werden.
- ... fassen Code, der an verschiedenen Stellen benötigt wird, zusammen.
- ... werden Prozeduren genannt, falls sie keinen Wert an den Aufrufer zurückgeben.
- ... werden Funktionen genannt, falls sie einen Wert an den Aufrufer zurückgeben.

Merkmale

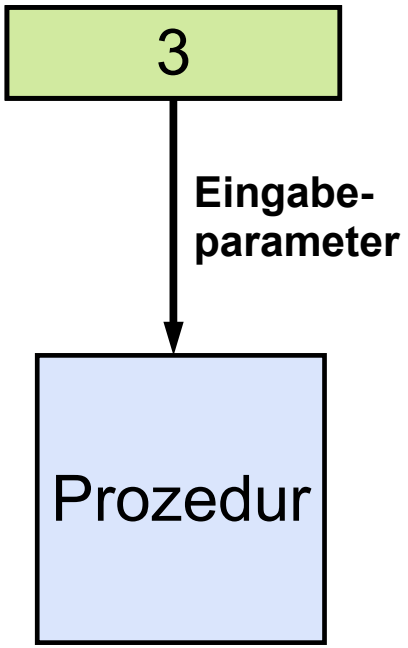
- Jedes Unterprogramm hat eine eindeutige Bezeichnung mit deren Hilfe es aufgerufen wird.
- Jedes Unterprogramm ist unabhängig von anderen Unterprogrammen. Ausnahme: Es ruft ein weiteres Unterprogramm auf und nutzt den Rückgabewert.
- Innerhalb eines Unterprogramms
  - ... kann ein weiteres Unterprogramm aufgerufen werden.
  - ... kann aber kein weiteres Unterprogramm deklariert werden.

Vorteile

- 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 gekapselt. Die gekapselte Prozedur kann von verschiedenen Stellen aufgerufen werden.
- Der Code muss nur an einer Stelle bearbeitet werden. Fehler lassen sich schneller finden.
- Veränderungen lassen sich einfacher vornehmen und testen. Es sind nur Codefragmente betroffen

Wie arbeitet eine Prozedur?

- 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 Parameter übergeben werden. Dem Nutzer der Prozedur muss die Art, der Typ der Parameter bekannt sein. Einige Prozeduren besitzen keine Eingabeparameter.



Beispiel: Hintergrundfarbe eines Textfeldes setzen

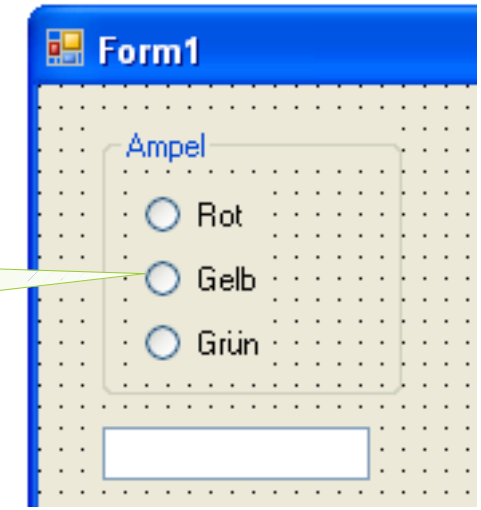
```
Sub whichFarbe(ByVal farbe As String)
    Select Case farbe

        Case "rot"
            txtFarbe.BackColor = Color.Red
        Case "gelb"
            txtFarbe.BackColor = Color.Yellow
        Case "gruen"
            txtFarbe.BackColor = Color.Green

    End Select
End Sub
```

Prozeduren

- ... beginnen mit Sub und enden mit End Sub.
- ... fassen Anweisungen, die zu einer Aktion gehören zusammen.
- ... haben einen eindeutigen, frei wählbaren Namen.
- ... geben keinen Wert zurück.
- ... können aber Werte übergeben bekommen. Innerhalb der runden Klammern stehen alle Parameter, die einer Prozedur übergeben werden. Zwischen dem Prozedurnamen und der Klammer befindet sich kein Leerzeichen!

Beispiel: Nachbildung einer Ampel

Für jedes Optionsfeld wird ein Event-Handler benötigt.




```
Private Sub optRot_CheckedChanged(
                ByVal sender As System.Object,
                ByVal e As System.EventArgs)
                Handles optRot.CheckedChanged

    If optRot.Checked = True Then
        Call whichFarbe("rot")
    End If
End Sub
```

Prozeduren aufrufen

Call whichFarbe("rot")

- Die Prozedur whichFarbe wird aufgerufen.
- Der Prozedur wird ein Parameter durch den Aufrufer übergeben. Der Parameter wird mit Hilfe der runden Klammern zusammengefasst.
- Die Parameter folgen unmittelbar dem Prozedurnamen.

Ereignisprozedur

```
Private Sub optRot_CheckedChanged( ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles optRot.CheckedChanged
```

- ... reagiert immer auf eine Benutzeraktion.
- ... ist privat. Sie kann nur innerhalb der Form genutzt werden.
- Der Name der Ereignisprozedur besteht aus
  - ... dem Namen des Steuerelements, welches die Aktion auslöst.
  - ... dem Namen des Ereignisses plus einem Präfix Unterstrich. In diesem Beispiel wird das Ereignis CheckedChanged behandelt. Der Status des Optionsfeldes ändert sich.
- Die Benutzeraktion wird mit Hilfe des Schlüsselwortes Handles eindeutig mit der angegebenen Prozedur verbunden.

Parameter der Prozedur

```
Private Sub optRot_CheckedChanged( ByVal sender As System.Object,
                ByVal e As System.EventArgs) Handles optRot.CheckedChanged
```

- sender enthält ein Verweis auf das auslösende Objekt. Das Objekt hat das Ereignis ausgelöst.
- e sind Parameter in Abhängigkeit des Ereignisses. Zum Beispiel kann in der entsprechenden Ereignisprozedur der Code der gedrückten Taste abgefragt werden.

Eigenschaften

If optRot.Checked = True ...

txtFarbe.BackColor = Color.Red

- Checked ist eine Eigenschaft eines Optionsfeldes. Die Eigenschaft bekommt als Wert den Status des Optionsfeldes übergeben. Der Wert True signalisiert ein aktives Optionsfeld.
- BackColor verändert die Hintergrundfarbe eines Fenster oder eines Steuerelements.
- Der Name des Objekts wird von der dazugehörigen Eigenschaft mit Hilfe eines Punktes getrennt.
- Variablen und Eigenschaften arbeiten gleich. Eigenschaften sind Variablen, die an ein bestimmtes Objekt gebunden sind. Eigenschaften beschreiben das Objekt.

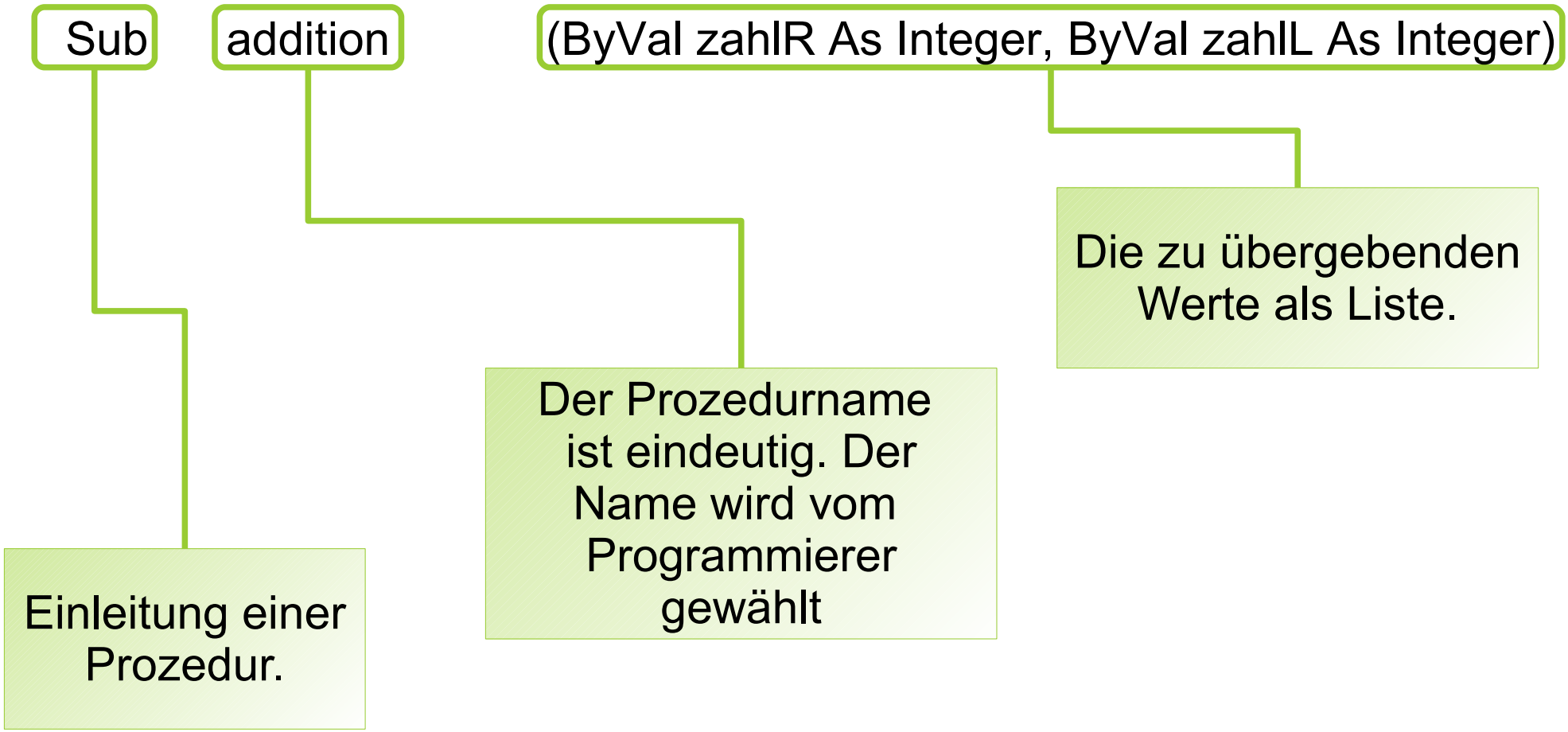
Prozeduren in VB erstellen

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

  Dim summe As Integer
  Dim ausgabe As String

  summe = zahlLinks + zahlRechts
  ausgabe = "Addition: " & zahlLinks & " + " & zahlRechts
  ausgabe = ausgabe & " = " & summe
  Console.WriteLine(ausgabe)

End Sub
```

Prozeduren bestehen aus

- ... dem Prozedurkopf Sub Prozedur(parameter01, parameter02).
  - Schnittstelle nach außen
  - Wie wird die Funktion aufgerufen?
- ... dem Prozedurrumpf.
  - Auszuführende Anweisungen.
- Jede Prozedur endet mit End Sub.

Prozedurkopf



Prozedurname

- ... 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.
- Schlüsselworter oder vordefinierte Prozedurnamen dürfen nicht genutzt werden.
- ... dürfen nur einmal vorkommen.
- ... sollten die Aufgabe der Prozedur widerspiegeln.

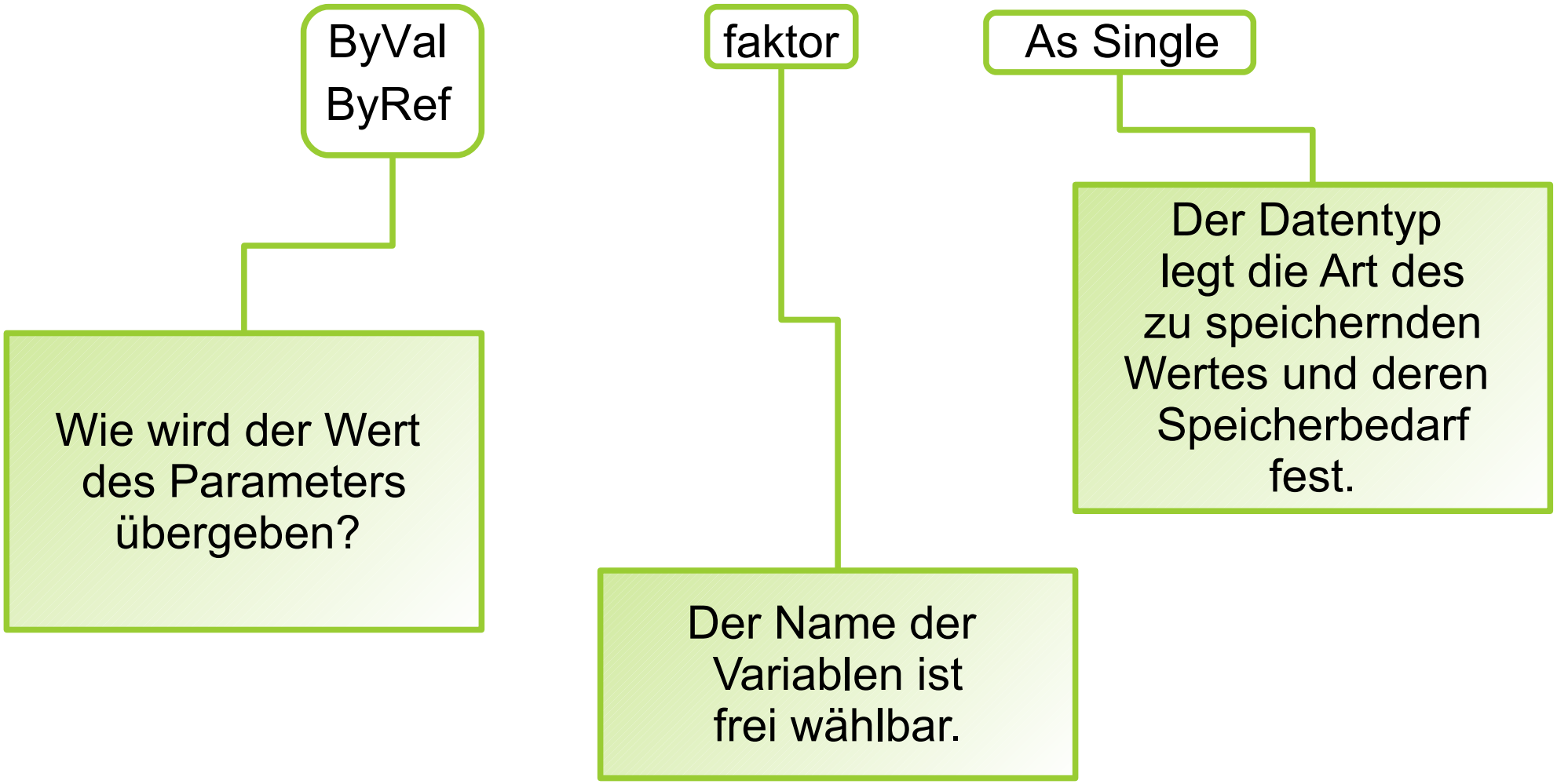
Beispiele

- Wenn die Aufgabe der Prozedur beschrieben werden soll, werden häufig Substantive genutzt. Welches Problem beschreibt die Prozedur?
  - Mittelwert(), Integral(), TuermeVonHanoi(), FehlerAusgeben().
- Wenn Berechnungen oder die Frage "Wie wird die Aufgabe erledigt?" beschrieben werden, werden häufig Verben genutzt.
  - intAddieren beschreibt eine Funktion die Ganzzahlen addiert.
  - Das Verb set... wird genutzt, wenn Eigenschaftswerte eines Objekts gesetzt werden.
  - Das Verb get... wird genutzt, wenn der Attributwert eines Objekts ausgelesen wird.
- Falls logische Operationen beschrieben werden, nutzen Prozedurnamen als erstes Wort Adjektive.
  - kleinsterWert(), groessterMesswert().

Parameterliste

- Die Liste wird mit Hilfe der runden Klammern begrenzt.
- Die Liste kann beliebig viele Parameter aufnehmen.
- Die einzelnen Listenelemente werden durch Kommata getrennt.
- Die Liste kann leer sein.
- Die Parameterliste folgt unmittelbar dem Prozedurnamen. Zwischen Liste und Prozedurnamen befindet sich kein Leerzeichen.

Parameter deklarieren



Prozeduren aufrufen

```
Sub Main()
   Dim nZahl01 As Integer = 5
   Dim nZahl02 As Integer = 6

   addition(nZahl01, nZahl02)

End Sub
```

Die Prozedur wird mit Hilfe des Namens aufgerufen. In runden Klammern folgen die zu übergebenden Parameter.

Möglichkeiten für den Prozeduraufruf

addition(nZahl01, nZahl02)

- Die Prozedur wird mit dem Namen aufgerufen.
- Die zu übergebenden Parameter werden in den runden Klammern zusammengefasst.
- Die runden Klammern werden automatisch gesetzt, wenn die Parameter ohne Klammern eingegeben werden.

Call addition(nZahl01, nZahl02)

- Der Aufruf wird mit Hilfe des Schlüsselwortes Call gekennzeichnet.
- Die Liste der Parameter muss geklammert werden.

Parameter zu ordnen

Aufruf: addition(nZahl01, nZahl02)

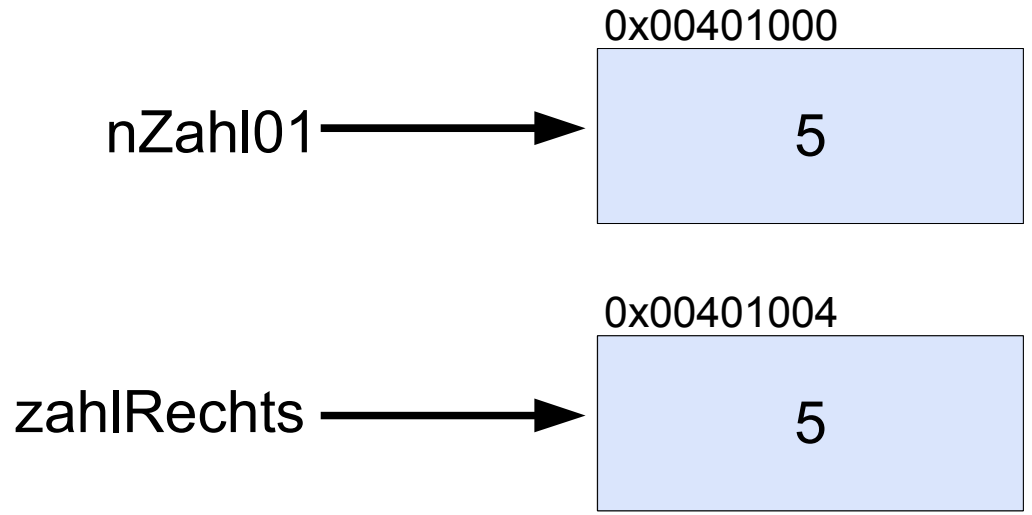
Prozedur: Sub addition (ByVal zahlR As Integer, ByVal zahlL As Integer)

- Der erste Parameter im Aufruf wird dem ersten Parameter im Prozedurkopf zugeordnet und so weiter.
- Der Parameter im Prozedurkopf und die Variable im Aufruf haben den gleichen Datentyp.
- Aufruf und Prozedurkopf haben die gleiche Anzahl von Parameter.
- Die einzelnen Elemente der Liste werden durch Kommata getrennt. Die Liste wird mit runden Klammern zusammengefasst.

Parameterübergabe "Call By Value"

ByVal variable As Integer

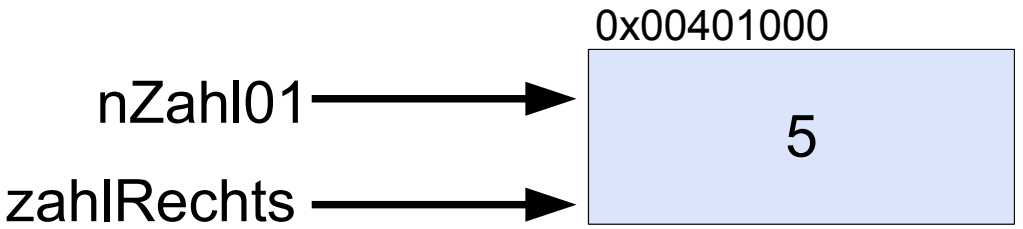
- Der Wert des Aufrufers wird in die Variable kopiert.
- Die Funktion hat keinen Zugriff auf die Variable des Aufrufers.
- Die Variable in der Parameterliste und die Variable des Aufrufers haben den gleichen Wert, liegen aber in unterschiedlichen Schubladen (Speicherstellen).
- Schreibschutz für die zu übergebene Variable.



Parameterübergabe "Call By Reference"

ByRef variable As Integer

- Die Variable selber wird übergeben.
- Es wird ein Verweis auf einen Wert übergeben.
- Die Variable in der Parameterliste und die Variable des Aufrufers zeigen auf die gleiche Schublade (Speicherstellen).
- Der Wert der übergebenden Variablen kann durch die Funktion verändert werden.



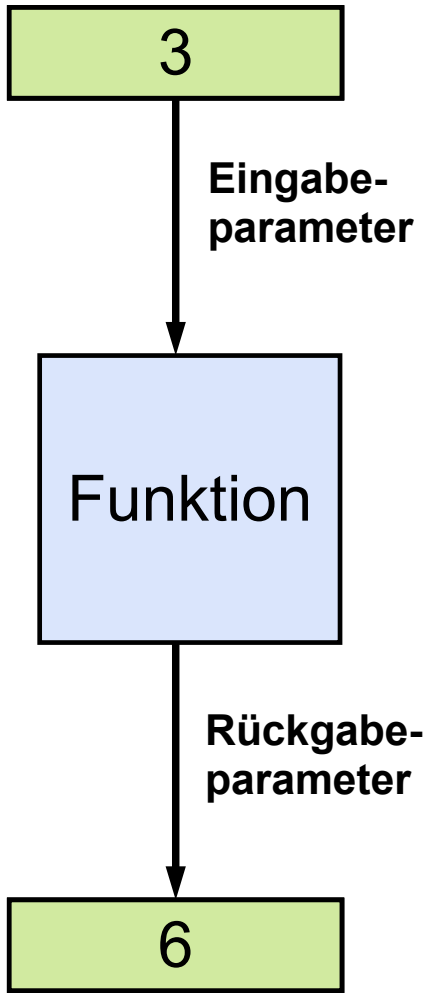
Sub Main

- ... stellt das Hauptprogramm für ein Konsolen-Programm dar.
- ... muss in jedem VB-Programm vorhanden sein.
- ... wird immer als erstes in einem Konsolen-Programm aufgerufen. Über diese Funktion können andere Funktionen aufgerufen werden.
- ... kann vom Betriebssystem Parameter übergeben bekommen.

```
Sub Main()
   Dim nZahl01 As Integer = 5
   Dim nZahl02 As Integer = 6

   addition(nZahl01, nZahl02)

End Sub
```

Wie arbeitet eine Funktion?

- 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 Parameter übergeben werden. Dem Nutzer der Funktion muss die Art oder der Typ der Parameter bekannt sein. Einige Funktionen besitzen keine Eingabeparameter.
- Die Lösung der Aufgabe kann an den Aufrufer zurückgegeben werden.



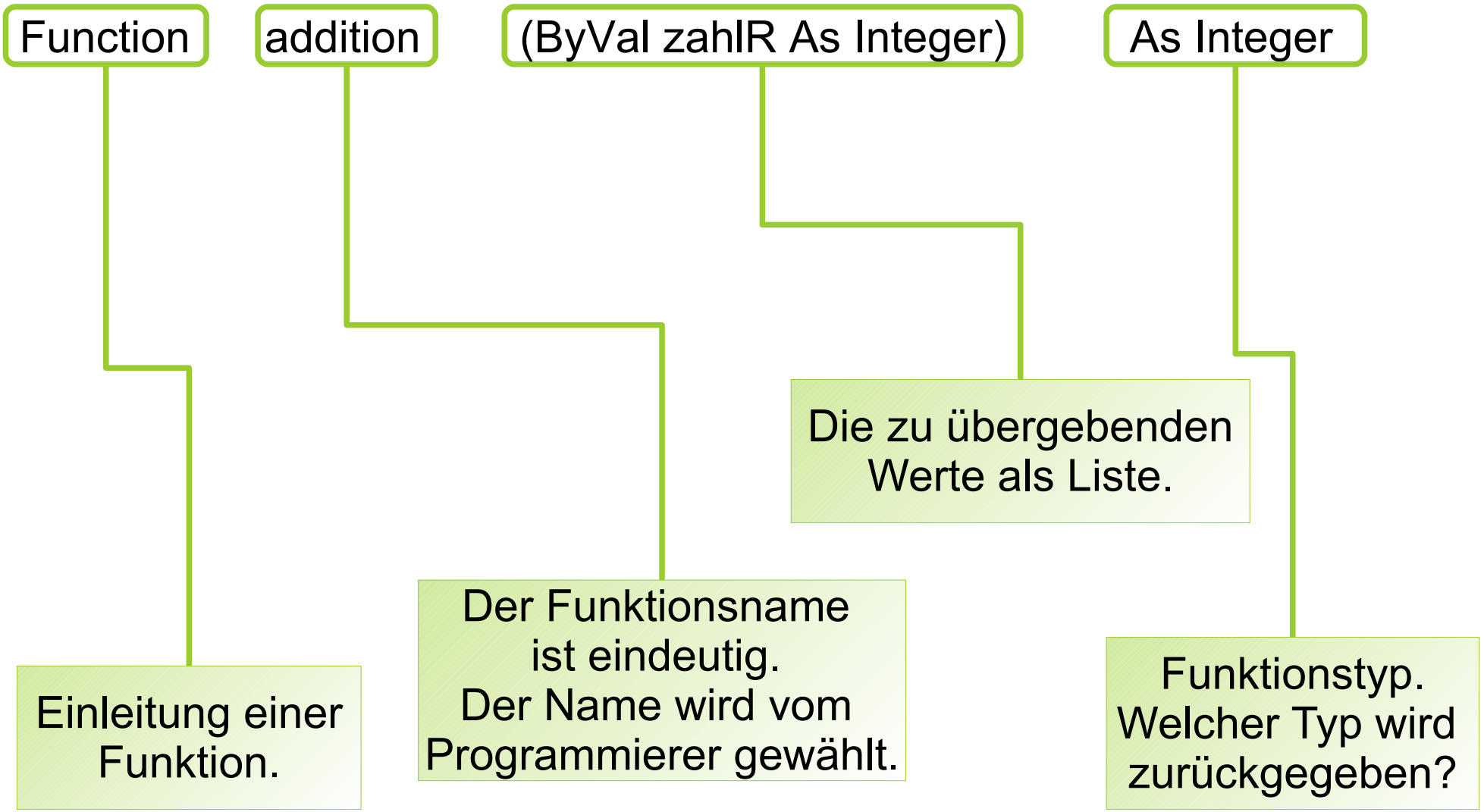
Funktionen in VB erstellen

```
Function Addition(ByVal zahlR As Integer, ByVal zahlL As Integer) As Integer

    Dim summe As Integer
    Dim ausgabe As String

    summe = zahlLinks + zahlRechts
    ausgabe = "Addition: " & zahlLinks & " + " & zahlRechts
    ausgabe = ausgabe & " = " & summe

    Return ausgabe

End Function
```

Funktionen bestehen aus

- ... dem Funktionskopf Function Funkt (para01, par02) As Integer.
  - Schnittstelle nach außen
  - Wie wird die Funktion aufgerufen?
  - Art des Rückgabewertes. Funktionstyp.
- ... dem Funktionsrumpf.
  - Auszuführende Anweisungen.
  - Eine Anweisung, die einen Wert an den Aufrufer zurückgibt.
- Jede Prozedur endet mit End Function.

Funktionskopf



Rückgabewert

Return wert

- Jede Funktion kann exakt einen Wert an den Aufrufer zurückgeben.
- ... hat den gleichen Datentyp wie die Funktion.
- ... kann in der Funktion berechnet werden.
- ... kann an jeder beliebiger Stelle in einer Funktion auftreten. Häufig steht die Anweisung am Ende einer Funktion.
- ... kann in Abhängigkeit einer Bedingung zurückgegeben werden.
- Mit Hilfe der Anweisung wird die Funktion verlassen.

Überladene Funktionen

- ... sind mehrfach definierte Funktionen, die alle die gleiche Aufgabe erfüllen, aber verschiedene Datentypen oder Anzahl von Parameter benötigen.
- ... haben alle den gleichen Funktionsnamen.
- ... unterscheiden sich in der Parameterliste.
- ... können sich im Rückgabewert unterscheiden, müssen aber nicht. Der Rückgabewert ist kein Kriterium zur Unterscheidung der Funktion.
- VB ordnet automatisch dem Aufruf die richtige Funktion zu.

Kriterium zur Unterscheitung

- Die Anzahl der Parameter ist unterschiedlich.
- Die Datentypen der Parameter unterscheiden sich.
- Beide Unterscheidungsmerkmale können in einer Mischform auftreten.

Beispiel

```
Function addition
        (ByVal zahlR As Integer, ByVal zahlL As Integer)
        As Integer
  ...
End Function

Function addition
        (ByVal zahlR As Integer, ByVal zahlL As Integer, ByVal zahlMitte As Integer)
        As Integer
  ...
End Function

Function addition
        (ByVal zahlR As Double, ByVal zahlL As Double, ByVal zahlM As Double)
         As Double
  ...
End Function
```

Module

- ... beginnt mit Module Bezeichnung und endet mit End Module.
- ... fasst Prozeduren, die eine gemeinsame Aufgabe bearbeiten, zusammen.
- ... kapselt Code zu einem bestimmten Thema.
- ... sind die Kapitel in einem Buch. Die Funktionen in einem Modul symbolisieren die verschiedenen Abschnitte.
- ... sind eine Sammlung von Deklarationen, Anweisungen und Prozeduren, die im Rahmen eines bestimmten Projekts gespeichert werden.
- ... ermöglichen eine strukturierte Programmierung.
- In einer Datei kann mehr als ein Modul vorhanden sein.

Mit Modulen arbeiten

- *Projekt – Modul einfügen* fügt dem aktiven Projekt ein neues Modul hinzu.
- Im Codefenster kann direkt der Modulname durch Überschreiben verändert werden.
- Öffnen Sie *Projekt – [projekt]-Eigenschaften*. Es werden nicht alle Optionen angezeigt. Klicken Sie auf die Registerkarte *Anwendung.* Mit Hilfe des Kombinationsfeld wählen Sie ein Startmodul aus.

Variablen und Blöcke

```
Module Berechnung
    Dim nSumme As Integer

    Function addition(ByVal zahlR As Integer, ByVal zahlL As Integer) As Integer
        nSumme = zahlLinks + zahlRechts
        Return nSumme
    End Function

    Sub Main()
        Dim nZahl01 As Integer = 5
        Dim nZahl02 As Integer = 6

        nSumme = addition(nZahl01, nZahl02)
    End Sub
End Module
```

In jedem Block gibt es jede Variable nur einmal. Der Variablenname ist eindeutig.

Gültigkeit

- Die Variable ist innerhalb der Prozedur oder Funktion definiert. Die Variable wird als Parameter genutzt.
  - Lokale Variablen.
  - Die Variablen sind nur innerhalb der Prozedur oder Funktion gültig.
- Die Variable ist innerhalb eines try-Blocks, bedingte Anweisung oder Schleife definiert.
  - Lokale Variablen.
  - Die Variablen sind nur innerhalb des Blocks gültig, in dem sie deklariert ist.
- Die Variable ist am Anfang des Moduls definiert.
  - Globale Variablen.
  - Die Variablen sind im gesamten Modul gültig.
  - Globale Variablen sollten nur in Ausnahmefällen genutzt werden.
- Nach Verlassen des Blocks wird die Variable zerstört.

Lokale Variablen

```
Module Berechnung
    Dim nSumme As Integer

    Function addition(ByVal zahlR As Integer, ByVal zahlL As Integer) As Integer
        nSumme = zahlLinks + zahlRechts
        Return nSumme
    End Function

    Sub Main()
        Dim nZahl01 As Integer = 5
        Dim nZahl02 As Integer = 6

        nSumme = addition(nZahl01, nZahl02)
    End Sub
End Module
```

Diese Variablen sind nur zwischen den Anweisungen Sub Main() und End Sub bekannt. Die Variablen können innerhalb dieses Blockes genutzt werden.

Lokale Variablen

```
Module Berechnung
    Dim nSumme As Integer

    Function addition(ByVal zahlR As Integer, ByVal zahlL As Integer) As Integer
        nSumme = zahlLinks + zahlRechts
        Return nSumme
    End Function

    Sub Main()
        Dim nZahl01 As Integer = 5
        Dim nZahl02 As Integer = 6

        nSumme = addition(nZahl01, nZahl02)
    End Sub
End Module
```

Variablen in der Parameterliste sind lokal.

Globale Variablen

```
Module Berechnung
    Dim nSumme As Integer

    Function addition(ByVal zahlR As Integer, ByVal zahlL As Integer
        nSumme = zahlLinks + zahlRechts
        Return nSumme
    End Function

    Sub Main()
            Dim nZahl01 As Integer = 5
            Dim nZahl02 As Integer = 6

            nSumme = addition(nZahl01, nZahl02)
    End Sub
End Module
```

Diese Variable ist im Module Berechnung bekannt. Die Variable kann auch innerhalb der Funktionen genutzt werden. Sie ist im gesamten Modul gültig.

Sichtbarkeit

```
Dim nSumme As Integer

Sub addition(ByVal zahlRechts As Integer, ByVal zahlLinks As Integer)
    nSumme = zahlLinks + zahlRechts
    Console.WriteLine(nSumme)
End Sub

Sub Main()
    Dim nZahl01 As Integer = 5
    Dim nZahl02 As Integer = 6
    Dim nZahl03 As Integer = 7
    Dim nSumme As Integer

    nSumme = 20
    addition(nZahl01, nZahl02)
End Sub
```

Die lokale Variable nSumme überlagert die globale Variable nSumme. Die globale Variable ist in der Prozedur nicht sichtbar, aber gültig.

Zugriff auf die Variablen

- Private oder Dim
  - Die Variable ist nur lokal gültig.
  - Sie ist in ihren Block gekapselt.
- Public
  - Die Variablen sind öffentlich.
  - Jeder kann auf die Variable zugreifen.
  - ... darf nur auf Modulebenen genutzt werden.
  - Die Variable kann im gesamten Projekt genutzt werden.

Statische Variablen

```
Sub plusEins()
    Static nSumme As Integer = 0

    nSumme = nSumme + 1
    Console.WriteLine(nSumme)
End Sub


Sub Main()
    plusEins()
    plusEins()
    plusEins()
End Sub
```

Diese Variable wird nicht am Ende des Blocks zerstört. Ihre Lebensdauer wird über das Ende des dazugehörigen Blocks verlängert. Statische Variablen können nicht innerhalb eines Moduls definiert werden.

Namensräume (Namespace)

- ... fassen Klassen nach Aufgabengruppen zusammen.
- ... können andere Namensräume enthalten.
- ... können als Baumstruktur abgebildet werden.
- ... vermeiden Konflikte zwischen gleichen Bezeichnungen.
- ... können mit Hilfe von *Ansicht – Objektbrowser* angezeigt werden.
- ... werden mit Hilfe von Imports am Anfang einer Datei eingebunden.

Beispiele für Namensräume

- Imports System enthält Basisklassen für die Datentypen, Exceptions, Events etc.
- Imports System.Console enthält Klassen für das Arbeiten mit der MS-DOS-Eingabeaufforderung.
- Imports System.Math enthält Konstanten für die Zahl PI und Funktionen wie Cos, Round etc.
- Imports System.String enthält Klassen für das Arbeiten und Manipulieren von Strings.

Strings formatieren

```
Imports System
Imports System.String
Module StringArbeiten

    Sub Main()
        Dim strZahl As String
        Dim nZahl As Integer

        strZahl = "4.3"
        nZahl = 3
        FormatCurrency(nZahl.ToString)   '3,00 €

        If (IsNumeric(strZahl) = True) Then
            FormatNumber(strZahl, 3)          '43,000
            Format(strZahl, "#00.000")        '4,3
        End If
End Module
```

Strings verbinden

```
Dim strLinks As String = "Eisbären"
Dim strRechts As String = " essen Fisch."
Dim strText As String
Dim laenge As Integer

strText = String.Concat(strLinks, strRechts) ' Strings verbinden
strText = strLinks + strRechts
strText = strLinks & strRechts

laenge = strText.Length                      ' Anzahl der Zeichen
```

Hinweise

- Die Anweisung "a" + 5 verursacht einen Fehler. Das Zeichen a kann nicht in einen Zahlenwert umgewandelt werden.
- Wenn String mit Hilfe des kaufmännischen Und verknüpft werden, werden Nicht-String-Variablen in String-Variablen konvertiert.
- Null-basierte Strings: variable = NULL. Der String hat keinen Wert. Wenn beide Strings Null sind, ist der verbundene String Null.
- Strings der Länge 0: variable = "". Wenn einer der Strings Null ist, wird der Null-basierte String in ein String der Länge 0 konvertiert.

Mit Strings arbeiten

```
Dim strLinks As String = "Eisbären"
Dim strRechts As String = " essen Fisch."
Dim strText As String
Dim pos As Integer
Dim buchstabe As Char
Dim wort() As String

strText = String.Concat(strLinks, strRechts)      ' Strings verbinden

wort = strText.Split(" ")                         ' Trennung
buchstabe = strText.Chars(laenge – 1)             ' Buchstabe zurückgeben
pos = strText.IndexOf(strLinks)                   ' Start der Zeichenfolge ermitteln
strText = strText.Substring(pos, strLinks.Length – 1)    'Substring speichern
strText.Replace("ä", "ae")                        'Zeichenfolge ersetzten
```

Objekt DateTime nutzen

```
Dim datum As DateTime
Dim datumHeute As DateTime
Dim datumZeit As DateTime

datum = New DateTime(2009, 12, 31)      ' (Jahr, Monat, Tag)
datumHeute = DateTime.Today             ' Heute
datumZeit = DateTime.Now                ' Heute + aktuelle Uhrzeit
```

- Ein neues Datum vom Objekt DateTime muss mit dem Schlüsselwort New erzeugt werden.
- DateTime ist ein Objekt, welches das Jahr, der Monat und der Tag als Parameter übergeben werden.

Bestandteile eines Datums

```
Dim datum As DateTime
Dim teil As Long

datum = New DateTime(2009, 12, 31)

teil = datum.Year
teil = datum.Month
teil = datum.Day
```

Datumswerte subtrahieren und addieren

```
Dim datum As DateTime
Dim newDatum As DateTime
Dim zeitspanne As TimeSpan
Dim teil As Long

datum = New DateTime(2009, 12, 31)

newDatum = datum.AddDays(3)                ' plus Anzahl Tage
newDatum = datum.AddMonths(3)              ' plus Anzahl Monat
zeitspanne = datum.Subtract(datumHeute)    ' datum - datumHeute
teil = zeitspanne.TotalDays                ' Anzahl Tage
```

Datumswerte vom Datentyp Date nutzen

```
Dim datum As Date
Dim heute As Date
Dim anzahlTage As Long

datum = #12/3/2009#
heute = #12/26/2009#

'heute - datum = Hier: Ergebnis in Tagen
anzahlTage = DateDiff(DateInterval.Day, datum, heute)

'heute + anzahlTage (Hier: Addition von Tagen)
datum = DateAdd(DateInterval.Day, anzahlTage, heute)
```