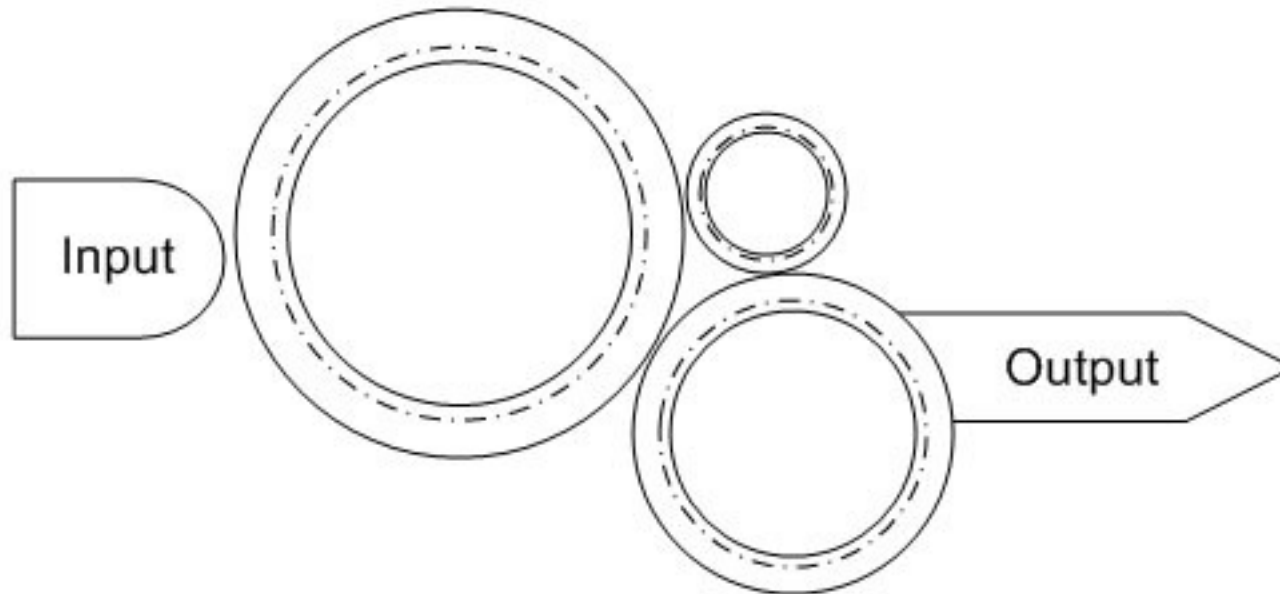


Access 2010 – Programmierung

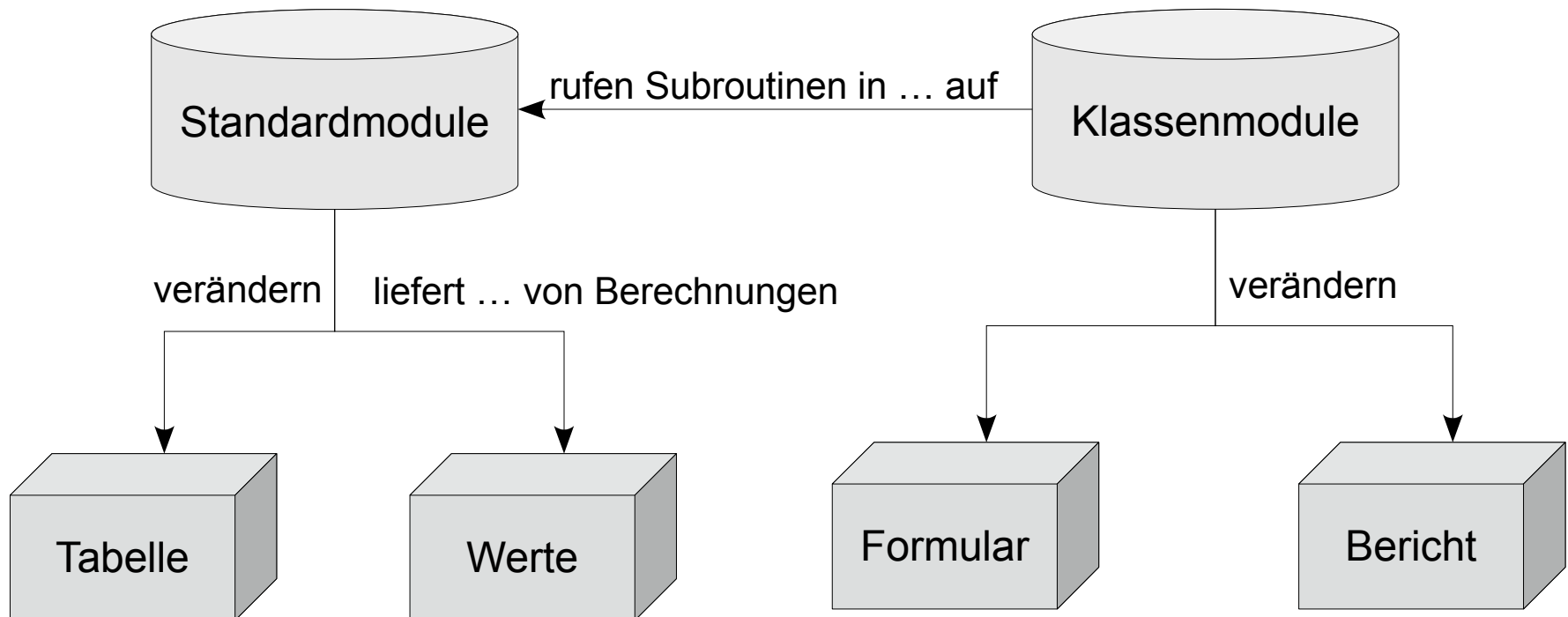
Funktionen in Access



Module ...

- kapseln Code zu einem Thema.
- sind Container für Code. In dem Container wird eine bestimmte Aufgabe des Gesamtprojekts gelöst.
- fassen Programmiercode und Deklarationen zu einem Thema zusammen.
- werden automatisch oder manuell vom Entwickler angelegt.
- werden im Projekt-Explorer des VBA-Editors angezeigt.

... in Access



Deklarationen, die für das gesamte Modul gelten ...

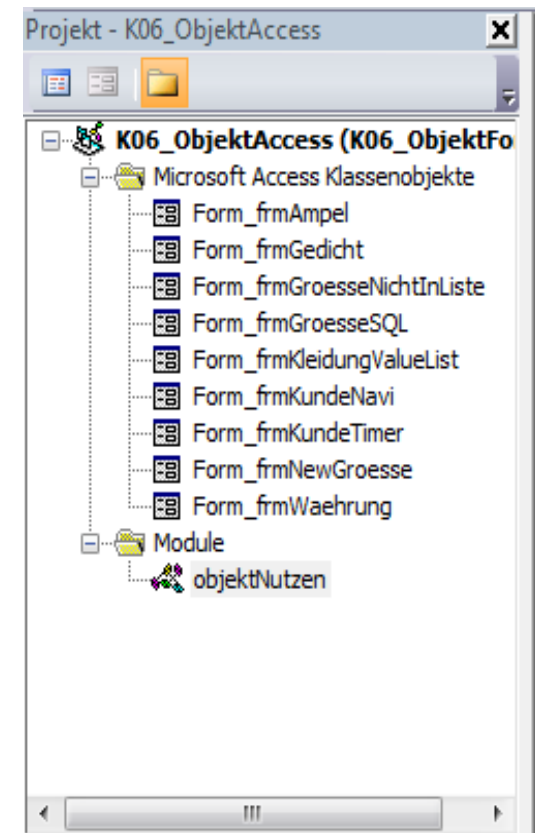
- stehen am Anfang des Moduls.
- « Option Compare Database » legt das Standardverfahren für Textvergleiche fest. Standardmäßig wird das Vergleichsverfahren der Datenbank genutzt.
- « Option Explicit ». Jede Variable muss vor ihrer Nutzung deklariert werden. Falls eine Variable nicht deklariert ist, wird eine Fehlermeldung ausgegeben.

Standardmodule ...

- sind Container für Deklarationen und Code, die an kein Klassenobjekt gebunden sind.
- enthalten Code, der eigenständig ablaufen kann.
- enthalten häufig Code, der Daten aus Tabellen verarbeitet.
- definieren keine neuen Objekte.
- beziehen sich nicht auf vorhandene Objekte.
- haben die Dateiendung „.bas“.

im Projekt-Explorer des VBA-Editors ...

- enthalten Standardmodule, die vom Entwickler selber geschrieben werden.
- Die Namen der Module werden vom Entwickler vergeben und sollten für sich selber sprechen.

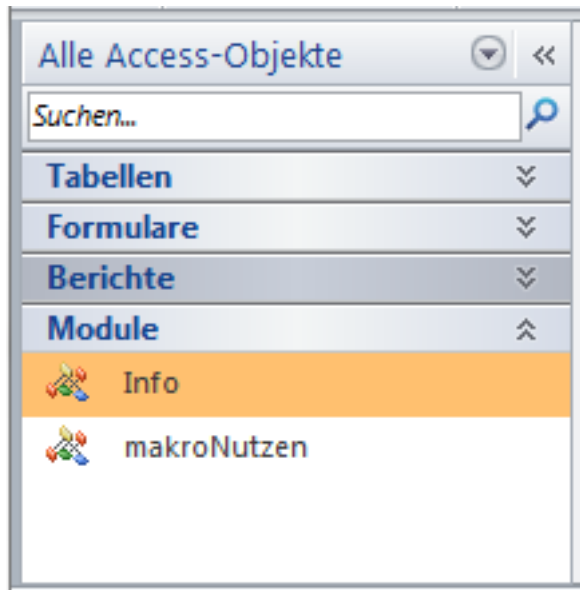


... erstellen

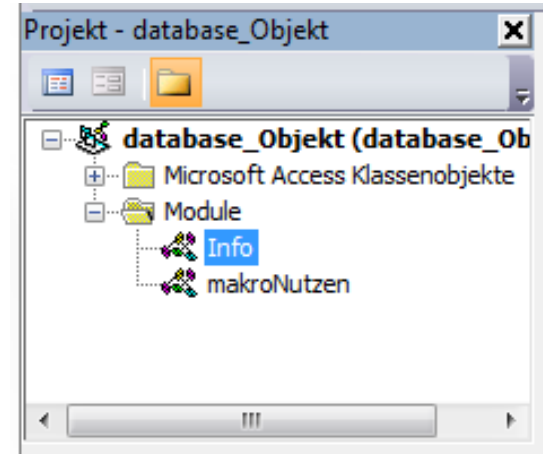
- Im Datenbank-Fenster:
 - Das Menüband *Erstellen* ist aktiv.
 - Mit einem Klick auf das Symbol *Modul* in dem Bereich Makros und Code wird ein eine leeres Modul im VBA-Editor angezeigt. Falls das Symbol nicht angezeigt wird, wird mit einem Klick auf den schwarzen Pfeil nach unten das Menü erweitert.
- Im VBA-Editor:
 - *Einfügen – Modul*.
 - Es wird ein leeres Modul im VBA-Editor angezeigt.

Anzeige des Standard-Modulnamens ...

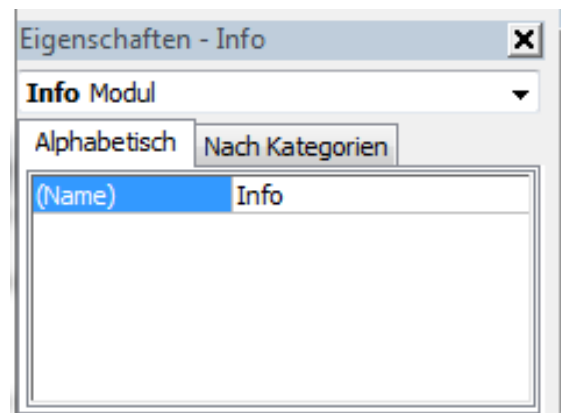
Navigationsbereich



Projekt-Explorer - Module

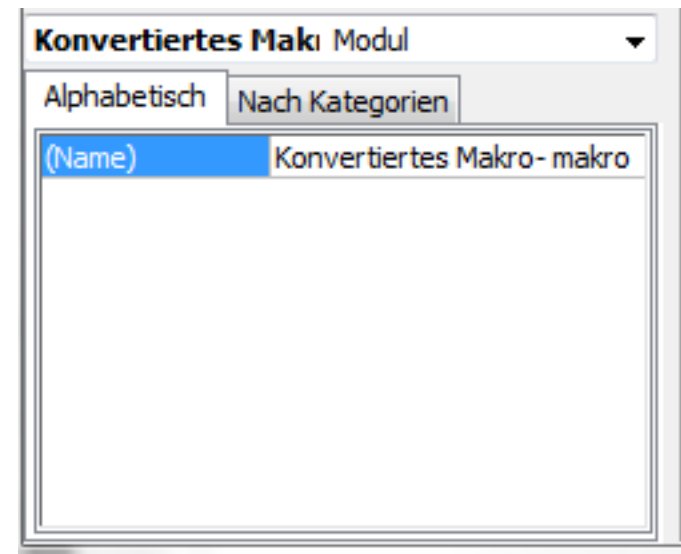


Eigenschaftenfenster



Modul umbenennen

- Das Eigenschaftsfenster im VBA-Editor ist geöffnet.
- Die Textzeile (Name) wird mit einem Mausklick aktiviert. Die Einfügemarke ist eingblendet.
- Mit Hilfe der Tastatur kann der vorgegebene Name überschrieben werden.



... speichern

- *Datei - ... speichern* im VBA-Editor.
- <STRG>+<S>.
- Nicht gespeicherte Module werden beim Schließen der Datenbank auf Nachfrage gespeichert.

... aus dem Navigationsbereich entfernen

- Mit Hilfe der linken Maustaste wird das zu löschende Modul im Navigationsbereich markiert.
- Durch Drücken der Taste <ENTF> wird der Löschvorgang gestartet.
- Wenn die eingeblendete Warnmeldung mit *Ja* bestätigt wird, wird das Modul aus der Datenbank entfernt. Der Löschvorgang kann nicht rückgängig gemacht werden.

... aus dem Projekt-Explorer entfernen

- Mit Hilfe eines rechten Mausklick wird das zu löschende Modul markiert und das dazugehörige Kontextmenü geöffnet.
- Mit Hilfe des Befehls *Entfernen von ...* im Kontextmenü wird der Löschvorgang gestartet und ein Hinweis eingeblendet.
- Wenn der Hinweis mit
 - *Ja* bestätigt wird, wird das Modul vor der Löschung gespeichert (exportiert).
 - *Nein* bestätigt wird, wird das Modul ohne Speicherung gelöscht.
 - *Abbrechen* bestätigt wird, wird der Löschvorgang abgebrochen.
- Der Löschvorgang kann nicht rückgängig gemacht werden!

... importieren und exportieren

- *Datei – Datei importieren*. Ein gespeichertes Standardmodul wird in das aktuelle Projekt geladen.
- *Datei – Datei exportieren* lädt eine Kopie des Standardmoduls in die aktuelle Datenbank.

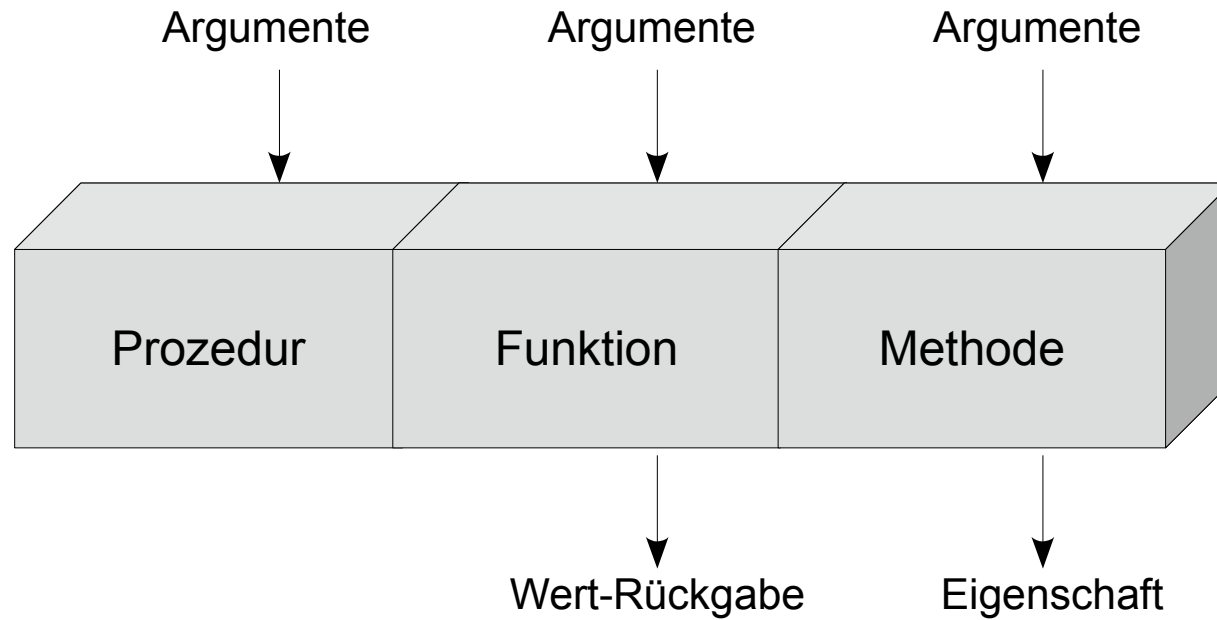
Subroutinen (Unterprogramme) ...

- lösen Teilprobleme der Gesamtaufgabe.
- fassen Anweisungen, die ein bestimmtes Thema bearbeiten, zu einem Block zusammen.
- sind eine Abfolge von VBA-Befehlen und -Anweisungen zu einem bestimmten Thema, die zeilenweise abgearbeitet werden.
- fassen Code zusammen. Der Code kann von verschiedenen Stellen aufgerufen werden.
- können nicht verschachtelt werden.

Vorteile

- Die Aufgabenstellung wird in kleinere Routinen geteilt. Jede Routine spiegelt eine Aktion innerhalb der Aufgabe wieder.
- Kleinere Routinen sind besser lesbar und wartbar.
- Wiederverwendung von Code, der in verschiedenen Aufgabenstellungen vorkommt.
- Kapselung von Code, um diesen eigenständig zu testen. Sobald der Code fehlerfrei ist, kann dieser in dem Projekt von jeder beliebigen Stelle aufgerufen werden.
- Code kann ohne Einfluss auf andere Programmteile verändert werden, wenn die Schnittstelle nach außen nicht verändert wird.

Implementierung



Argumente einer Subroutine

- Die Anzahl und die Art der Argumente werden im Kopf einer Subroutine festgelegt.
- Die Werte der Argumente werden in der Subroutine verarbeitet.
- Argumente steuern den Ablauf einer Subroutine.

... aufrufen

- Benutzerdefinierte Subroutinen und Methoden werden mit ihren Namen aufgerufen.
- Subroutinen können durch Auslösung eines Ereignisses aufgerufen werden.

Funktionen ...

- beginnen in VBA mit « Function » und enden mit « End Function ».
- werden mit Hilfe Ihres Namens aufgerufen.
- geben einen Wert an den Aufrufer zurück.
- können in VBA integriert sein oder vom Entwickler selber geschrieben werden.
- können mit Hilfe von « Exit Function » vorzeitig beendet werden.

Beispiel

```
Sub AufrufFunktion()
```

```
  Const zahlA As Integer = 3
```

```
  Const zahlB As Integer = 4
```

```
  Dim ergebnis As Integer
```

```
  ergebnis = BionomischeFormel(zahlA, zahlB)
```

```
End Sub
```

```
Function BionomischeFormel(zahlA As Integer, zahlB As Integer) As Integer
```

```
  Dim ergebnis As Integer
```

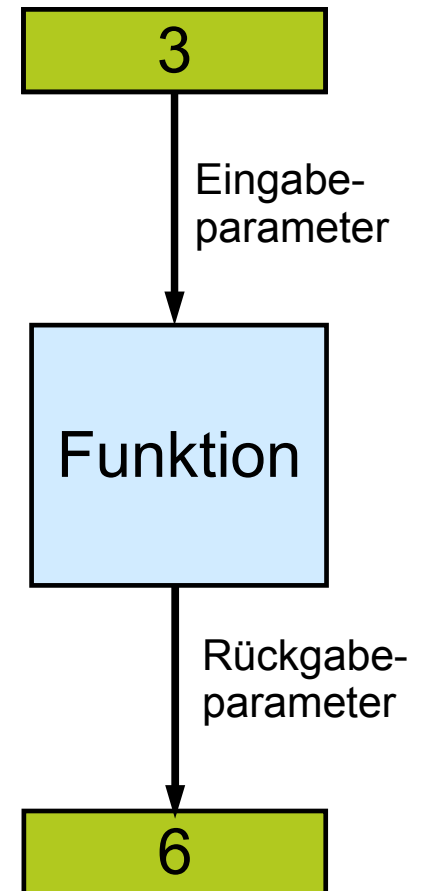
```
  ergebnis = (zahlA ^ 2) + (2 * zahlA * zahlB) + (zahlB ^ 2)
```

```
  BionomischeFormel = ergebnis
```

```
End Function
```

Arbeitsweise

- Der Funktion können Eingabeparameter (Argumente) übergeben werden.
- Diese Eingabeparameter werden in der Funktion verarbeitet. Die Verarbeitung selber ist dem Aufrufer aber nicht bekannt. Der Nutzer kennt nur die Schnittstelle (den Funktionskopf) nach außen.
- Die Funktion gibt einen Wert zurück. Wie dieser Wert berechnet wurde, weiß der Nutzer nicht. Der Nutzer kennt nur die Art des Rückgabewertes.



Integrierte Funktionen

- Mathematische Funktionen.
- Berechnung von Datums- und Zeitwerten.
- Bearbeitung von Strings.
- Konvertierung von Datentypen
- Daten aus Dateien oder vom Bildschirm ein- oder auslesen.
- Ordner oder Datei nutzen.
- Die Funktionen werden alphabetisch in der VBA-Hilfe (*Visual Basic-Sprachverzeichnis - Funktionen*) aufgelistet.
- Viele Funktionen, die in Abfragen etc. genutzt werden, sind auch in VBA vorhanden.

Meldungsfenster erzeugen

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

- Mit Hilfe der Funktion «MsgBox() » wird ein Meldungsfenster eingeblendet.
- Der Funktion können folgende Argumente übergeben werden:
 - Der anzuzeigende Text « prompt». Der Text informiert den Benutzer über Fehler etc.
 - Welche Schaltflächen werden im Meldungsfenster angezeigt? « buttons »
 - Ein Titel « title » für die Titelleiste des Fensters.
- Rückgabewert der Funktion:
 - Es wird ein Integer-Wert zurück gegeben. Mit Hilfe der Ganzzahl wird die gedrückte Schaltfläche codiert.

Beispiel

```
Sub AufrufFunktion()
```

```
    Const zahlA As Integer = 3
```

```
    Const zahlB As Integer = 4
```

```
    Dim ergebnis As Integer
```

```
    Dim button As VbMsgBoxResult
```

```
    Dim info As String
```

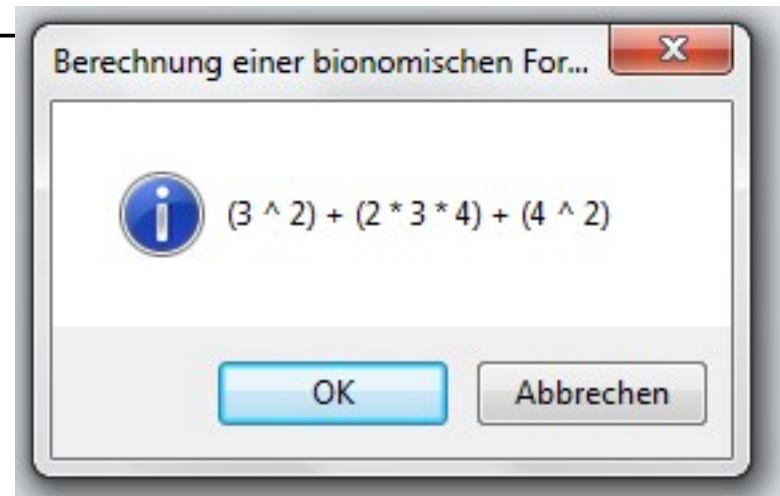
```
    ergebnis = BionomischeFormel(zahlA, zahlB)
```

```
    info = "(" & zahlA & " ^ 2) + (2 * " & zahlA & " * " & zahlB
```

```
    info = info & ") + (" & zahlB & " ^ 2)"
```

```
    button = MsgBox(info, vbOKCancel + vbInformation, "Bionomischen Formel")
```

```
End Sub
```



Argument-Listen umbrechen

```
Sub AufrufFunktion()  
    Dim button As VbMsgBoxResult  
    Dim info As String  
  
    info = "(" & zahlA & " ^ 2) + (2 * " & zahlA & " * " & zahlB  
    info = info & ") + (" & zahlB & " ^ 2)"  
    button = MsgBox(info, _  
                    vbOKCancel + vbInformation, _  
                    "Bionomischen Formel")  
  
End Sub
```

Zeilenfortsetzungszeichen ...

- wird aus dem Leerzeichen und dem Unterstrich zusammengesetzt.
- folgt direkt das Zeilenende.
- trennt lange Argument-Listen nach dem Komma. Dadurch wird die Lesbarkeit erhöht.
- kann keine langen Zeichenketten trennen.
- Nach dem Zeilenfortsetzungszeichen kann kein Kommentar eingefügt werden.

Eingaben am Bildschirm

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

- Die Funktion « InputBox() » blendet ein Fenster mit einem Textfeld zur Eingabe ein.
- Der Benutzer kann in das Textfeld einen Wert mit Hilfe der Tastatur schreiben.

Argumente und Rückgabewert

- Argumente:
 - Der anzuzeigende Text « prompt ». Der Text informiert den Benutzer über die Eingabemöglichkeiten.
 - Ein Titel « title » für die Titelleiste des Fensters.
 - Einen Standardwert « default » für das Textfeld festlegen.
- Rückgabewerte:
 - Es wird der Inhalt des Textfeldes zurückgegeben.
 - Der Rückgabewert ist immer vom Datentyp « String ».

Beispiel

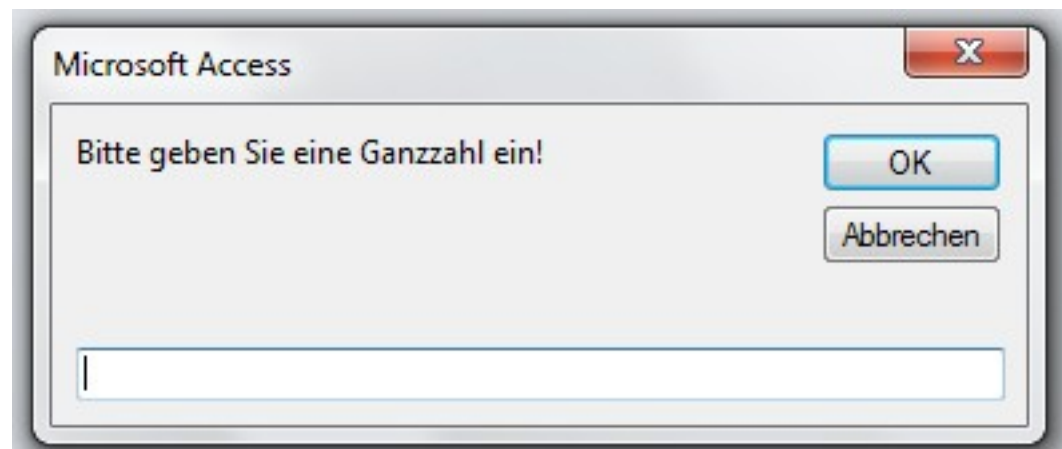
```
Sub Eingabe(zahl As Integer)
```

```
    Dim txtZahl As String
```

```
    txtZahl = InputBox("Bitte geben Sie eine Ganzzahl ein!")
```

```
    zahl = CInt(txtZahl)
```

```
End Sub
```



Konvertierung von Datentypen

- Unterschiedliche Datentypen in Ausdrücken werden häufig automatisch (implizit) umgewandelt.
- Mit Hilfe von Funktionen kann eine Konvertierung des Ausdrucks vom Programmierer explizit erzwungen werden.

Funktionen zur Konvertierung

	Konvertierung zu ...
CBool(variable)	Boolean
CByte(variable)	Byte
CInt(variable)	Integer
CLng(variable)	Long
CSng(variable)	Single
CDbl(variable)	Double
CCur(variable)	Currency
CDate(variable)	Date
CStr(variable)	String

Beispiel

```
Sub Eingabe(zahl As Integer)
```

```
    Dim txtZahl As String
```

```
    txtZahl = InputBox("Bitte geben Sie eine Ganzzahl ein!")
```

```
    zahl = CInt(txtZahl)
```

```
End Sub
```

Mathematische Funktionen nutzen

```
Sub Mathematik()
```

```
  Const zahl As Double = -3.5
```

```
  Dim sekans As Double
```

```
  Dim absolutwert As Double
```

```
  sekans = 1 / Cos(zahl)      ' Sekans berechnen
```

```
  absolutwert = Abs(zahl)    ' Absolutwert berechnen
```

```
End Sub
```

Datumsfunktionen nutzen

```
Sub DatumFunc()  
    Dim heute As Date  
    Dim aktuellZeit As Date  
    Dim gestern As Date  
    Dim morgen As Date  
  
    heute = Date                ' Aktuelles Systemdatum  
    aktuellZeit = Time         ' Aktuelle Systemzeit  
  
    gestern = DateDiff("d", 1, heute) ' heute - 1 (in Tagen)  
    morgen = DateAdd("d", 1, heute)   ' heute + 1 (in Tagen)  
End Sub
```

Stringfunktionen nutzen

```
Dim myText As String
Dim zeichen As String
Dim count As Integer
Dim pos As Integer
Dim laenge As Integer

count = 0
laenge = Len(myText)
For pos = 1 To laenge
    zeichen = Mid(myText, pos, 1)
    If (zeichen Like Chr(13)) Then
        If (Mid(myText, pos + 1, 1) Like Chr(10)) Then
            count = count + 1
        End If
    End If
Next pos
```

Benutzerdefinierte Funktionen in VBA

```
Function BionomFormel(zahlA As Integer, zahlB As Integer) As Integer
    Dim ergebnis As Integer

    ergebnis = (zahlA ^ 2) + (2 * zahlA * zahlB) + (zahlB ^ 2)

    BionomFormel = ergebnis

End Function
```

Funktionskopf

« Funktion Eingabe() As String »

« Function BinomFormel(zahlA As Integer, zahlB As Integer)
As Integer »

- Der Funktionskopf beginnt mit dem Schlüsselwort « Function ».
- Dem Schlüsselwort folgt der Name der Funktion. Der Name ist frei wählbar.
- In runden Klammern steht die Argumentliste. Die Argumentliste beschreibt die Anzahl und Art der Eingabeparameter.
- Jede Funktion hat einen bestimmten Datentyp « As ... ».

Regeln für den Funktionsnamen

- Jeder benutzerdefinierter Name beginnt mit einem Buchstaben.
- Die Bezeichnung besteht nur aus den Buchstaben a..z, A..Z, dem Unterstrich und den Zahlen 0..9.
- Jeder Funktionsname wird in einem Modul nur einmal genutzt.
- Die Groß- und Kleinschreibung von benutzerdefinierten Namen wird nicht beachtet.
- Ein benutzerdefinierter Namen kann maximal 255 Zeichen haben.
- VBA-Schlüsselwörter oder von VBA, definierte Funktionsnamen können nicht als benutzerdefinierte Namen genutzt werden.

Geeignete Funktionsnamen ...

- geben Auskunft über die Nutzung der Funktion.
- beschreiben die in der Funktion, gekapselten Aktionen.
- entsprechen dem Sprachraum des Entwicklers.
- sind an die Sprache der realen Welt angelehnt.

Zusammengesetzte Funktionsnamen

- « Function BionomischeFormel() ». Jedes Wort in dem benutzerdefinierten Namen beginnt mit einem Großbuchstaben.
- « Function Umrechnung_Meter_Centimeter() ». Die Wörter werden mit Hilfe des Unterstrichs getrennt. Der Unterstrich ersetzt Leerzeichen in einem Wort.

Argumentliste ...

Function BinomFormel(zahlA As Integer, zahlB As Integer) As Integer

Function AusgabeText() As String

- beginnt und endet mit den runden Klammern.
- steht direkt hinter dem Funktionsnamen.
- kann beliebig viele Argumente enthalten.
- kann leer sein. Der Funktion werden keine Eingabeparameter übergeben.

Die Argumente in der Liste ...

- symbolisieren einen Platzhalter für einen Wert, der der Funktion übergeben wird.
- werden wie Variablen definiert. Das Schlüsselwort «Dim» regelt den Zugriff auf eine Variable und wird in der Argumentliste nicht benötigt.
- bekommen mit Hilfe von «As » einen Datentyp zugewiesen. Es kann jeder beliebiger Datentyp genutzt werden.

Datentyp einer Funktion ...

Function BinomFormel(zahlA As Integer, zahlB As Integer) As Integer

- wird durch das Schlüsselwort « As » am Ende des Funktionskopfes festgelegt. Es kann jeder beliebige Datentyp genutzt werden.
- Die Argumentliste und die Angabe des Datentyps werden durch ein Leerzeichen getrennt.
- legt die Verwendung des Rückgabewertes der Funktion fest.
- ist optional. Wenn keine Angabe vorhanden ist, wird ein Datentyp in Abhängigkeit des Rückgabewertes ermittelt.
Nachteil: Der Nutzer kann die Weiterverarbeitung des Wertes nicht planen.

Rückgabewert festlegen

« BionomFormel = ergebnis »

« Funktionsname = Ausdruck »

- An einer beliebigen Position innerhalb der Funktion wird dem Funktionsnamen ein Wert zugewiesen. Falls kein Rückgabewert festgelegt wird, wird ein Standardwert entsprechend des Datentyps der Funktion zurückgegeben.
- Der Funktionsname ist wie der Variablenname ein Platzhalter für einen bestimmten Wert.
- Der Wert des Ausdrucks rechts von dem Gleichheitszeichen sollte den gleichen Datentyp besitzen wie die Funktion.
- Die Übergabe eines Wertes an die Funktion beendet aber die Funktion nicht. Eine Funktion wird nur durch « End Function » oder « Exit Function » beendet.

Ein Rückgabewert

```
Function BionomFormel(zahlA As Integer, zahlB As Integer) As Integer
```

```
    Dim ergebnis As Integer
```

```
    ergebnis = (zahlA ^ 2) + (2 * zahlA * zahlB) + (zahlB ^ 2)
```

```
    BionomFormel = ergebnis
```

```
End Function
```

```
Sub Aufruffunktion()
```

```
    Dim ergebnis As Integer
```

```
    ergebnis = BionomischeFormel(zahl01, zahl02)
```

```
End Sub
```

Rückgabewert als Array (Feld)

```
Function AllMonate() As String()  
    Dim monatName(11) As String  
  
    monatName(0) = "Januar"  
    monatName(1) = "Februar"  
  
    AllMonate = monatName  
End Function
```

```
Sub Aufruffunktion()  
    Dim monatsnamen() As String  
  
    monatsnamen = AllMonate()  
End Sub
```

Benutzerdefinierten Typ als Rückgabewert nutzen

```
Function NewMitarbeiter(cvsText As String) As T_Mitarbeiter
    Dim mitarbeiter As T_Mitarbeiter

    mitarbeiter.vorname = cvsTeile(0)
    mitarbeiter.nachname = cvsTeile(1)
    NewMitarbeiter = mitarbeiter
End Function
```

```
Sub Aufruffunktion()
    Dim neuMitarbeiter As T_Mitarbeiter

    neuMitarbeiter = NewMitarbeiter("Hans;Müller)
End Sub
```

Benutzerdefinierte Typen ...

- werden am Anfang eines Moduls definiert.
- beginnen mit « Type » und enden mit « End Type »
- klammern Variablen, die ein bestimmtes Objekt beschreiben.
- definieren einen eigenen Datentyp.

... definieren

Option Compare Database

Private Type T_Mitarbeiter

vorname As String

nachname As String

strasse As String

wohnort As String

geburtsdatum As Date

beschaeftigSeit As Date

gehalt As Currency

End Type

Kopf der Typ-Definition

« Private Type T_Mitarbeiter »

- Mit Hilfe von « Private » wird der Zugriff auf den neuen Datentyp geregelt. Der Typ kann nur innerhalb des Moduls genutzt werden, in dem die Definition steht.
- Das Schlüsselwort « Type » kennzeichnet eine benutzerdefinierte Typ-Definition.
- Dem Schlüsselwort folgt der Name des neuen Datentyps. Der Name ist frei wählbar. Bei der Auswahl müssen aber die gleichen Konventionen wie bei Variablen, Subroutine-Name etc. beachtet werden.

Variablen in einem benutzerdefinierten Typ

- symbolisieren einen Platzhalter für eine Eigenschaft des zu beschreibenden Objekts.
- werden wie Variablen definiert. Das Schlüsselwort «Dim» regelt den Zugriff auf eine Variable und wird in einer Typ-Definition nicht benötigt.
- bekommen mit Hilfe von «As » einen Datentyp zugewiesen. Es kann jeder beliebiger Datentyp genutzt werden.

... nutzen

```
Function NewMitarbeiter(cvsText As String) As T_Mitarbeiter
    Dim cvsTeile() As String
    Dim mitarbeiter As T_Mitarbeiter

    cvsTeile = Split(cvsText, ";")

    mitarbeiter.vorname = cvsTeile(0)
    mitarbeiter.nachname = cvsTeile(1)
    mitarbeiter.strasse = cvsTeile(2)
    mitarbeiter.wohnort = cvsTeile(3)
    mitarbeiter.geburtsdatum = CDate(cvsTeile(4))
    mitarbeiter.beschaefigtSeit = CDate(cvsTeile(5))
    mitarbeiter.gehalt = CCur(cvsTeile(6))

End Function
```

Variablen vom ...

« mitarbeiter As T_Mitarbeiter ».

- Als Datentyp für eine Variable wird der Name des benutzerdefinierten Typs genutzt.
- Der benutzerdefinierte Typ wird einer Variablen genauso wie ein Standard-Datentyp zugewiesen.
- Die Variable ist ein Platzhalter für die Struktur des benutzerdefinierten Typs.

Variablen des benutzerdefinierten Typs nutzen

«mitarbeiter.vorname = cvsTeile(0)».

- Die Variable « mitarbeiter » kann nur auf gekapselten Variablen in dem angegebenen Datentyp « T_Mitarbeiter » zugreifen und verändern.
- Der Punkt verbindet die Variable « mitarbeiter » mit den, in dem angegebenen Datentyp, gekapselten Variablen.
- Als Zuweisungsoperator wird das Gleichheitszeichen genutzt.

Aufruf einer Funktion aus einer Subroutine

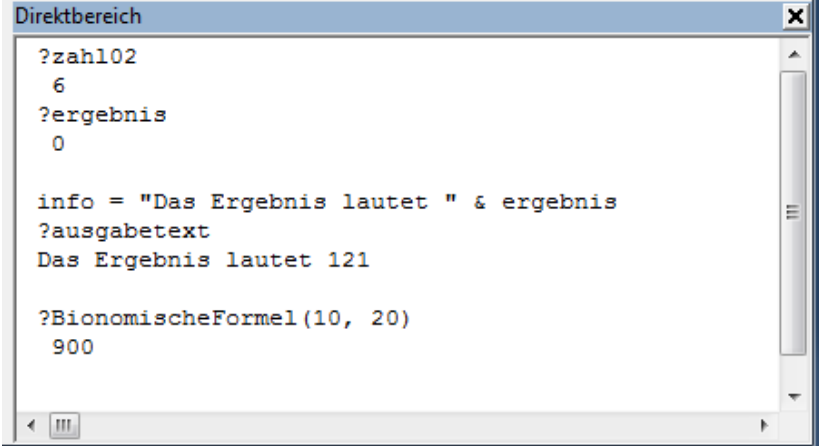
« gestern = DateDiff("d", 1, heute) »

« ergebnis = BionomFormel(zahlA, zahlB) »

- Die Funktion wird mit ihrem Namen aufgerufen.
- Dem Funktionsnamen folgt eine Argumentliste. Die Argumentliste wird durch Klammern begrenzt. Die Argumente werden durch Kommata getrennt. Die Anzahl der Argumente entspricht standardmäßig der Anzahl der Parameter der aufzurufenden Funktion. Die Argumente werden den Parametern von links nach rechts zugeordnet.
- Mit Hilfe des Gleichheitszeichen wird einer Variable der Rückgabewert der Funktion zugewiesen. Eine Funktion kann aber auch wie eine Prozedur ohne Berücksichtigung des Rückgabewertes aufgerufen werden.

Direktfenster / -bereich zum Testen nutzen

- Testen von Subroutinen und Variablen.
- Die Tests im Direktfenster werden immer in einem bestimmten Kontext gestartet.



```
Direktbereich
?zahl02
6
?ergebnis
0

info = "Das Ergebnis lautet " & ergebnis
?ausgabertext
Das Ergebnis lautet 121

?BionomischeFormel(10, 20)
900
```

Voraussetzung

- Das Programm wird im Einzelschritt-Modus durchlaufen. In der Prozedur ist ein Haltepunkt definiert.
- Das Direktfenster wird mit Hilfe des Menüs *Ansicht – Direktfenster* eingeblendet.
- Die zu testenden Subroutinen sind definiert.

Eingabe in das Direktfenster

- Anweisungen können manuell eingegeben oder aus dem Code-Bereich kopiert werden.
- Pro Zeile wird eine Anweisung in das Direktfenster eingegeben.
- Jede Zeile wird mit <RETURN> abgeschlossen. Die Anweisung wird automatisch ausgeführt.

Test-Möglichkeiten im Direktfenster

« ?Left("Eisbären",3) ».

« ?FunctionRueckgabe.BionomischeFormel(2, 3) ».

- Es können nur Prozeduren getestet werden, die einen Rückgabewert haben.
- Der Aufruf beginnt mit einem Fragezeichen.
- Der Rückgabewert der Funktion wird nach dem Drücken der <Return>-Taste in der nächsten Zeile ausgegeben.

Anweisungen aus dem Direktfenster löschen

- Mit Hilfe der Maus werden Anweisungen im Direktfenster markiert. Mit Hilfe der Tastenkombination <STRG>+<A> wird das Direktfenster vollständig markiert.
- Mit Hilfe von <ENTF> wird der markierte Text aus dem Direktfenster gelöscht.

Zuordnung der Argumente

Sub Aufruf()

Dim zahlA As Integer

Dim zahlB As Integer

zahlA = 3

zahlB = 4

Call BionomischeFormel(zahlA, zahlB)

End Sub

Funktion BionomischeFormel(zahlA As Integer, zahlB As Integer) As Integer

BionomischeFormel = $(zahlA^2) + (2 * zahlA * zahlB) + (zahlB^2)$

End Sub

Erläuterung

- Die Anzahl der Parameter im Aufruf entspricht der Anzahl der Elemente in der Argumentliste.
- Die Parameter werden den Argumenten von links nach rechts zugeordnet.
- Der Wert des ersten Parameters im Aufruf wird dem ersten Element in der Argumentliste zugeordnet und so weiter.
- Die Parameter werden den Argumenten in Abhängigkeit ihrer Position in der Liste zugeordnet.
- Um eventuelle Fehler auszuschließen sollten der Parameter sowie das Argument den gleichen Datentyp haben.

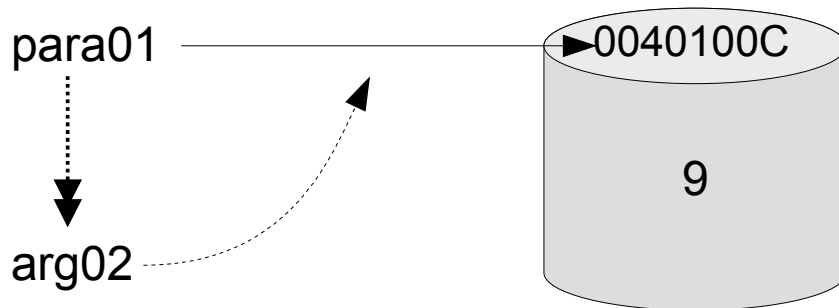
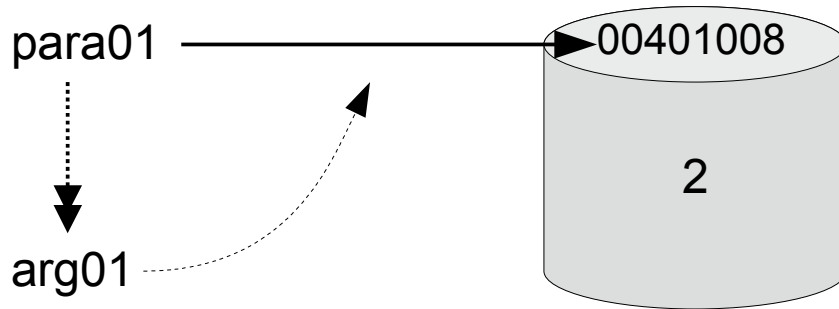
call bei reference ...

« Function Func(arg01 As Integer, arg02 As Integer) »

« Function Func(ByRef arg01 As Integer, ByRef arg02 As Integer) »

- ist die Standardübergabe von Argumenten.
- Der zu übergebene Parameter verweist auf einen Speicherplatz, an dem ein bestimmter Wert abgelegt ist.
- Das Argument bekommt nicht den Wert übergeben, sondern die Position des Speicherplatzes. Das Argument verweist auf den gleichen Speicherplatz und damit Wert wie der Parameter.

Grafische Darstellung



call bei value ...

« Function F(ByVal arg01 As Integer, ByVal arg02 As Integer) »

- Der zu übergebene Parameter verweist auf einen Speicherplatz, an dem ein bestimmter Wert abgelegt ist.
- Das Argument bekommt eine Kopie des Wertes, auf den der zu geordnete Parameter verweist.
- Vorteil: Die Parameter können nicht durch die aufgerufene Funktion verändert werden.

Grafische Darstellung



Optionale Argumente nutzen

```
Function Umrechnung(zahl As Double, _  
                    Optional faktor As Integer, _  
                    Optional einheit As String) As String  
    Dim ergebnis As Double  
  
    ergebnis = zahl * faktor  
    Umrechnung = ergebnis & " " & einheit  
  
End Function
```

Optionale Argumente ...

- werden mit dem Schlüsselwortes « Optional » gekennzeichnet.
- müssen beim Aufruf nicht übergeben werden.
- stehen immer am Ende einer Argumentliste.
- können von jedem beliebigen Datentypen sein.
- können in der Argumentliste von Funktionen genutzt werden.

... schreibgeschützt setzen

```
Function Umrechnung(zahl As Double, _  
                    Optional ByVal faktor As Integer, _  
                    Optional einheit As String) As String  
    Dim ergebnis As Double  
  
    ergebnis = zahl * faktor  
    Umrechnung = ergebnis & " " & einheit  
  
End Function
```

Standardwerte nutzen

```
Function Umrechnung(zahl As Double, _  
                    Optional ByVal faktor As Integer = 10, _  
                    Optional einheit As String = "") As String  
    Dim ergebnis As Double  
  
    ergebnis = zahl * faktor  
    Umrechnung = ergebnis & " " & einheit  
  
End Function
```

Hinweise

- Wenn kein Standardwert gesetzt ist, wird automatisch ein Standardwert in Abhängigkeit des Datentyps genutzt.
- Wenn das Argument nicht übergeben wird, wird automatisch der angegebene Standardwert genutzt.
- Falls das Argument übergeben wird, wird der Wert des Arguments genutzt. Der Standardwert wird überschrieben.

Aufruf einer Funktion mit optionalen Argumenten

```
Sub Start()  
  Dim ausgabe As String  
  
  ausgabe = Umrechnung(10, 100, "cm")  
  ausgabe = Umrechnung(10)  
  ausgabe = Umrechnung(10, 100)  
  ausgabe = Umrechnung(10, , "cm")  
  
End Sub
```

Hinweise

- Die Elemente in der Argumentliste werden durch Kommata getrennt.
- Die Argumente werden in Abhängigkeit der Position einem Parameter zugeordnet.
- Wenn ein Argument nicht gesetzt ist, muss aber das Komma als Trennzeichen gesetzt werden. Ausnahme: Falls das letzte Argument nicht angegeben wird, kann das Komma vor diesem Argument entfernt werden.

Besser:

```
Sub Start()
```

```
    Dim ausgabe As String
```

```
    ausgabe = Umrechnung(zahl:=10, faktor:=100, einheit:="cm")
```

```
    ausgabe = Umrechnung(zahl:=10)
```

```
    ausgabe = Umrechnung(zahl:=10, faktor:=100)
```

```
    ausgabe = Umrechnung(zahl:=10, einheit:="cm")
```

```
End Sub
```

Benannte Argumente ...

- werden bei der Übergabe an Funktionen genutzt.
- bekommen mit Hilfe des Operators := ein Parameter übergeben. Zwischen den Doppelpunkt und dem Gleichheitszeichen darf kein Leerzeichen stehen. Der Operator := besteht aus zwei Zeichen.
- nutzen den Namen des Arguments in der Argumentliste im Funktionskopf und nicht die Position.
- werden in der Argumentliste durch Kommata getrennt. Die Reihenfolge der Argumente ist aber unwichtig.

Beliebige Anzahl von Argumente übergeben

```
Function CelsiusToFahrenheit(messwert As Double, _  
                             ParamArray messung() As Variant) As Double()  
    Dim element As Variant  
  
    newTemperatur(count) = ((messwert * 9) / 5) + 32  
  
    For Each element In messung  
        If IsNumeric(element) Then  
            newTemperatur(count) = ((Cdbl(element) * 9) / 5) + 32  
        End If  
    Next  
  
    CelsiusToFahrenheit = newTemperatur  
End Function
```

ParamArray im Funktionskopf ...

- kennzeichnet ein dynamisches Array vom Datentyp « Variant ».
- ist eine Variable, die alle Parameter enthält, die keinen definierten Argument zugeordnet werden können.
- kennzeichnet den Container für alle Parameter, die nicht in der Argumentliste definiert ist.
- ist immer das letzte Argument in der Argumentliste.
- kann nicht mit den Schlüsselwörtern « ByVal », « ByRef » oder « Optional » genutzt werden.

Aufruf

```
Sub Aufruf()
```

```
  Dim fahrenheit() As Double
```

```
  Dim element As Variant
```

```
  fahrenheit = CelsiusToFahrenheit(2.4)
```

```
  fahrenheit = CelsiusToFahrenheit(2.4, 3.4)
```

```
  fahrenheit = CelsiusToFahrenheit(2.4, 3.4, 5.6, 4#, 2.3, 4.5)
```

```
End Sub
```

Zuordnung

fahrenheit = CelsiusToFahrenheit(2.4)

Function CelsiusToFahrenheit(messwert As Double, ParamArray m() As Variant)

fahrenheit = CelsiusToFahrenheit(2.4, 3.4, 5.6, 4#, 2.3, 4.5)

Öffentliche Subroutinen ...

- können von jedem im Projekt genutzt werden.
- sind für alle Nutzer des Projektes sichtbar.
- sind häufig benutzerdefinierte Routinen. Benutzerdefinierte Routinen sind standardmäßig öffentlich.
- werden mit dem Schlüsselwort « Public » gekennzeichnet.

Beispiel

```
Public Function BionomischeFormel(zahlA As Integer, _  
                                zahlB As Integer) As Integer
```

```
    Dim ergebnis As Integer
```

```
    ergebnis = (zahlA ^ 2) + (2 * zahlA * zahlB) + (zahlB ^ 2)
```

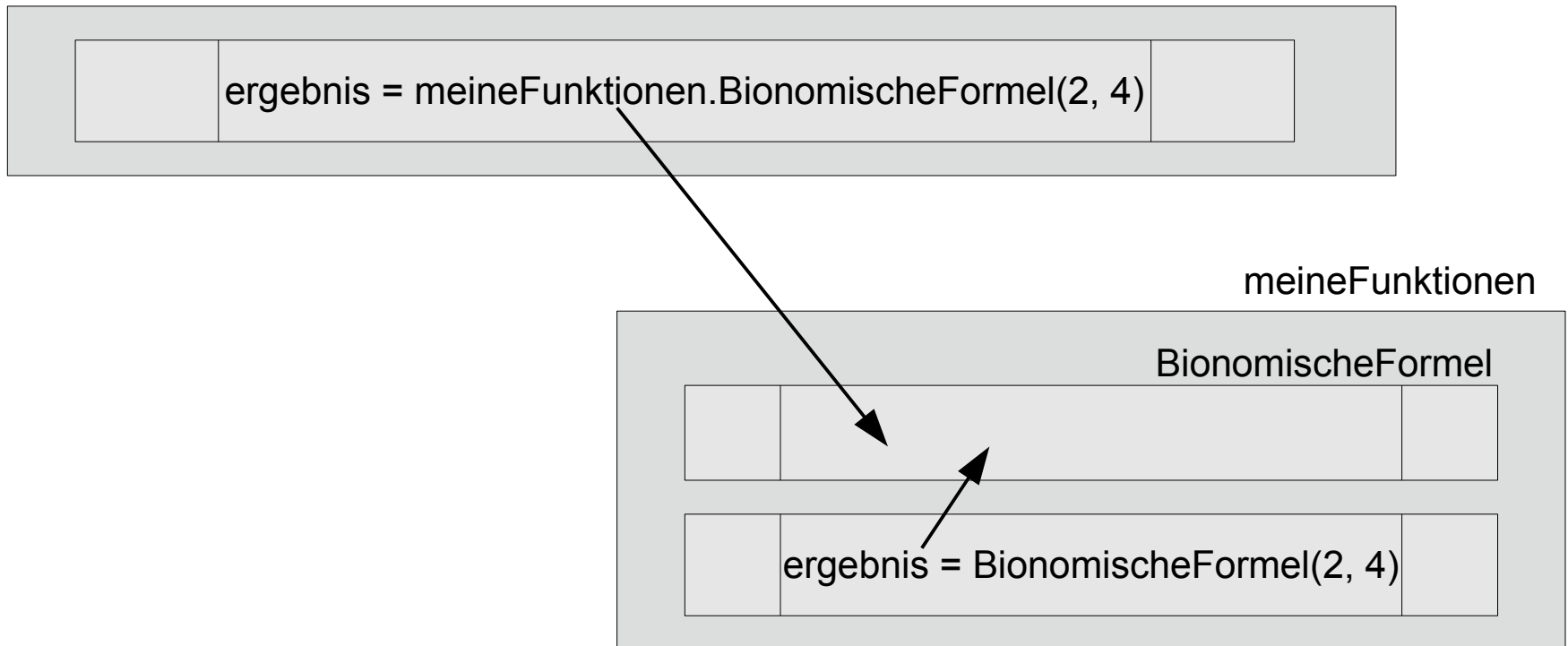
```
    BionomischeFormel = ergebnis
```

```
End Function
```

Aufruf

« ergebnis = meineFunktionen.BionomischeFormel(2, 4) »

« ergebnis = BionomischeFormel(zahl01, zahl02) »



Private Subroutinen ...

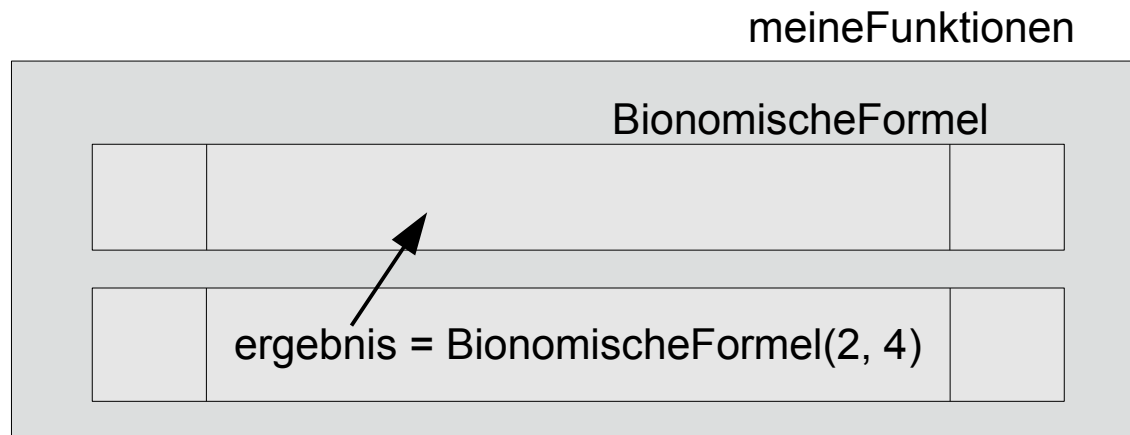
- können nur innerhalb des Moduls aufgerufen werden, in dem sie definiert sind.
- sind in einem Objekt gekapselt. Zum Beispiel Ereignisprozeduren sind an ein Formular, Bericht oder Steuerelement gebunden.
- werden von anderen Modulen nicht gesehen.
- werden mit dem Schlüsselwort « Private » gekennzeichnet.

Private benutzerdefinierte Subroutine

```
Private Function BionomischeFormel(zahlA As Integer, _  
                                   zahlB As Integer) As Integer  
  
    Dim ergebnis As Integer  
  
    ergebnis = (zahlA ^ 2) + (2 * zahlA * zahlB) + (zahlB ^ 2)  
  
    BionomischeFormel = ergebnis  
  
End Function
```

Aufruf

« `ergebnis = BionomischeFormel(zahl01, zahl02)` »



Öffentliche Variablen ...

« Public Const faktor As Integer = 10 »

« Public ausgabe As String »

- sind für alle Module sichtbar.
- können in allen Modulen eines Projekts genutzt werden.
- sind global.
- existieren nur auf Modul-Ebene.
- werden in den ersten Zeilen eines Moduls definiert.
- werden mit dem Schlüsselwort « Public » gekennzeichnet.
- sollten so wenig wie möglich genutzt werden.

Private Variablen in Modulen ...

« Private Const maxAnzahl As Integer = 100 »

« Private ausgabe As String »

- werden in den ersten Zeilen eines Moduls definiert.
- können überall in dem Modul, in dem sie definiert sind, genutzt werden. Die Variable ist aber nicht von außen sichtbar.
- Eine private Konstante muss mit dem Schlüsselwort « Const » gekennzeichnet werden.

Private Variablen in Subroutinen

« Const faktor As Integer = 10 »

« Dim zahl As Double »

- Variablen werden am Anfang einer Subroutine mit Hilfe des Schlüsselwortes « Dim » gekennzeichnet.
- Konstanten werden am Anfang einer Subroutine mit Hilfe des Schlüsselwortes « Const » gekennzeichnet.
- Die Variable kann überall in der Subroutine, in der sie definiert ist, genutzt werden. Die Variable ist aber nicht von außen sichtbar.
- Argumente sind private, lokale Variablen einer Subroutine.

Lokaler Gültigkeitsbereiche von Variablen

- Die Variablen sind in ihrem Modul oder Subroutine sichtbar und können dort genutzt werden. Die Sichtbarkeit einer Variablen / Konstanten wird durch deren Definition festgelegt.
- Die Variable ist nur in dem einen Modul gültig. Die Variable wird mit « Private » gekennzeichnet.
- Die Variable ist nur in einer Subroutine gültig. Die Variable wird mit « Dim » / « Const » gekennzeichnet.
- Die Lebensdauer beginnt mit dem Aufruf des Moduls oder der Subroutine und endet mit deren Ende.

Öffentlicher Gültigkeitsbereich von Variablen

- Die Variable ist in dem gesamten Projekt sichtbar. Eine öffentliche Variable kann aber durch eine lokale Variable gleichen Namens überdeckt werden. In diesem Bereich ist die globale Variable nicht nutzbar.
- Die Variable wird mit « Public » am Anfang eines Moduls gekennzeichnet.
- Die Lebensdauer ist durch die Lebensdauer des Projekts beschränkt.