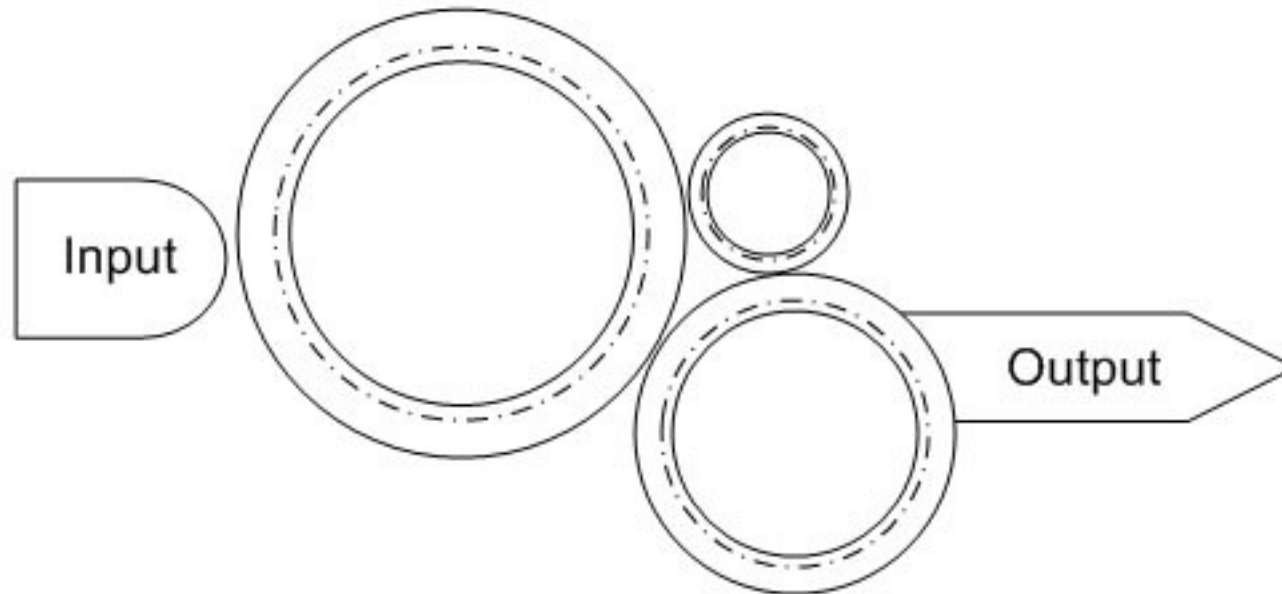


Access 2010 – Programmierung

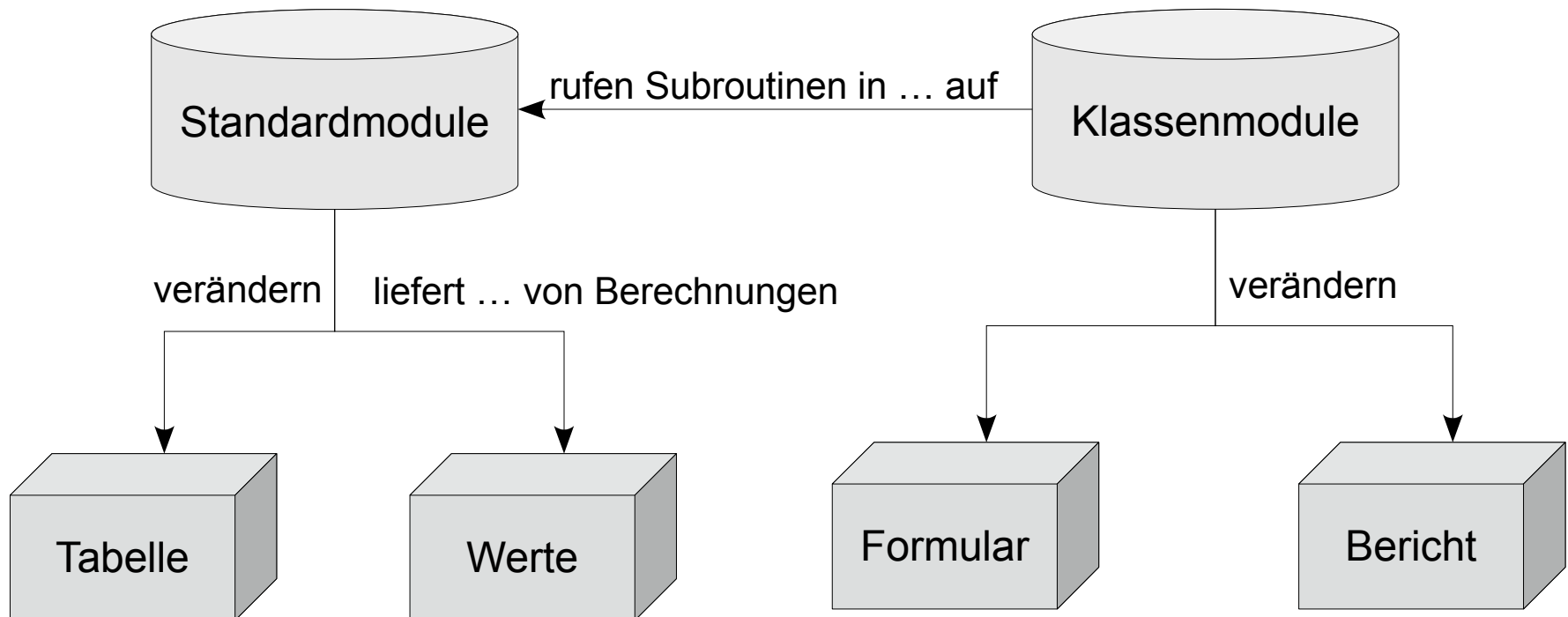
Prozeduren definieren



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 ...

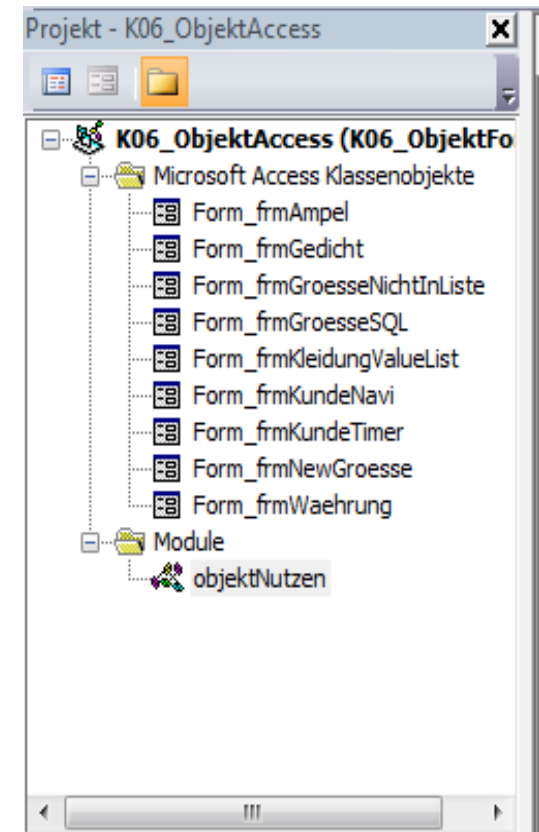
- stehen am Anfang des Moduls.
- « Option Compare Database » legt das Standardverfahren für Textvergleiche fest. Standardmäßig wird das Vergleichsverfahren der Datenbank genutzt.
- « Option Explicit ». Jede Variable muss vor ihrer Nutzung deklariert werden. Falls eine Variable nicht deklariert ist, wird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

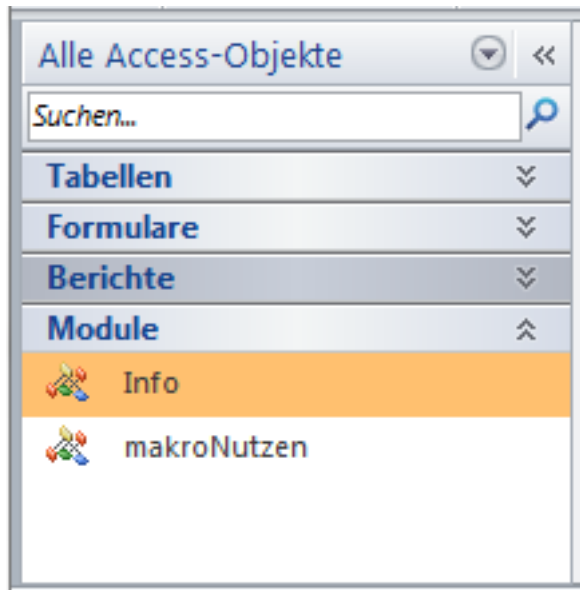


... erstellen

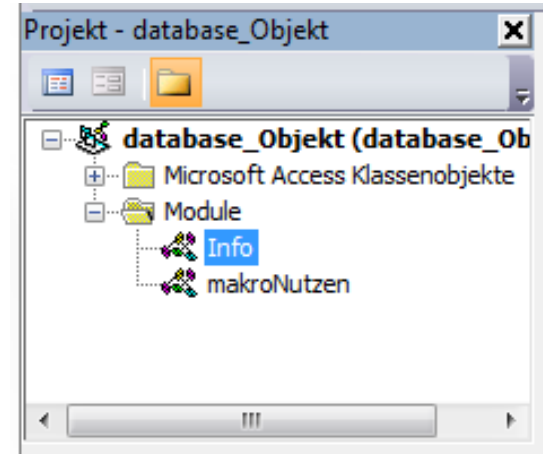
- Im Datenbank-Fenster:
 - Das Menüband *Erstellen* ist aktiv.
 - Mit einem Klick auf das Symbol *Modul* in dem Bereich Makros und Code wird ein eine leeres Modul im VBA-Editor angezeigt. Falls das Symbol nicht angezeigt wird, wird mit einem Klick auf den schwarzen Pfeil nach unten das Menü erweitert.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

Anzeige des Standard-Modulnamens ...

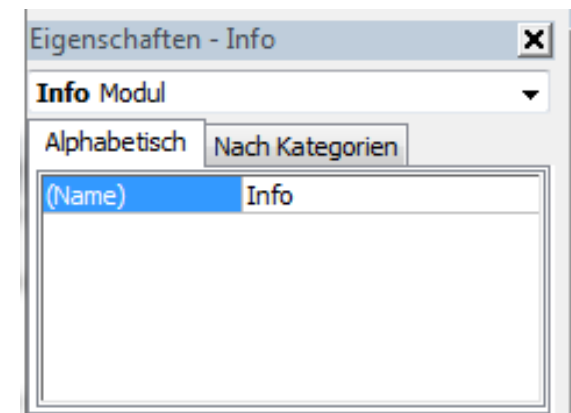
Navigationsbereich



Projekt-Explorer - Module

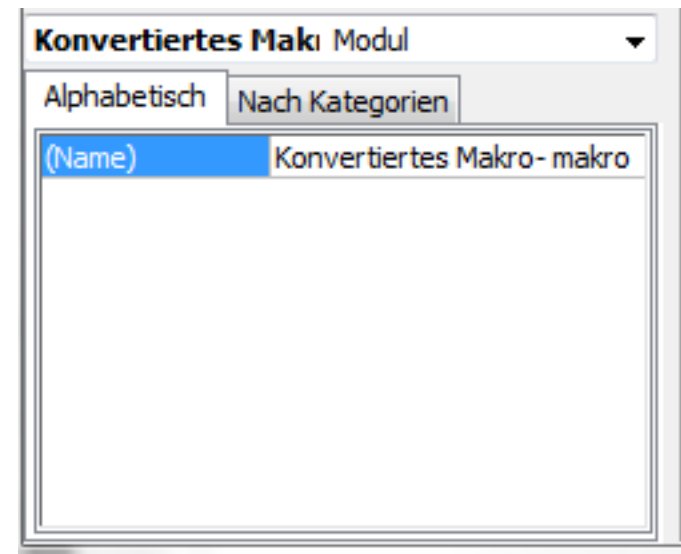


Eigenschaftenfenster



Modul umbenennen

- Das Eigenschaftsfenster im VBA-Editor ist geöffnet.
- Die Textzeile (Name) wird mit einem Mausklick aktiviert. Die Einfügemarke ist eingeblendet.
- 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.

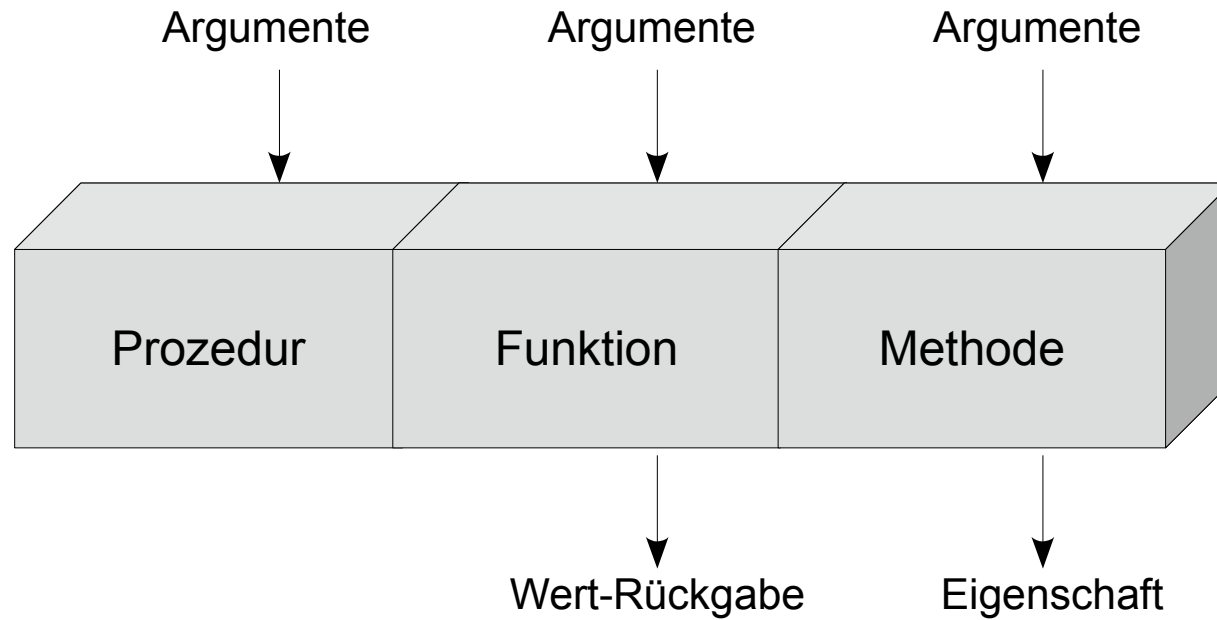
Subroutinen (Unterprogramme) ...

- 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.

Prozeduren ...

- beginnen in VBA mit « Sub » und enden mit « End Sub »
- können auf ein Ereignis reagieren.
- können mit Hilfe Ihres Namens aufgerufen werden.
- geben keinen Wert an den Aufrufer zurück.
- können mit Hilfe von « Exit Sub » vorzeitig beendet werden.

Beispiel

```
Sub Aufruf()
```

```
  Dim zahlA As Integer
```

```
  Dim zahlB As Integer
```

```
  zahlA = 3
```

```
  zahlB = 4
```

```
  Call BionomischeFormel(zahlA, zahlB)
```

```
End Sub
```

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

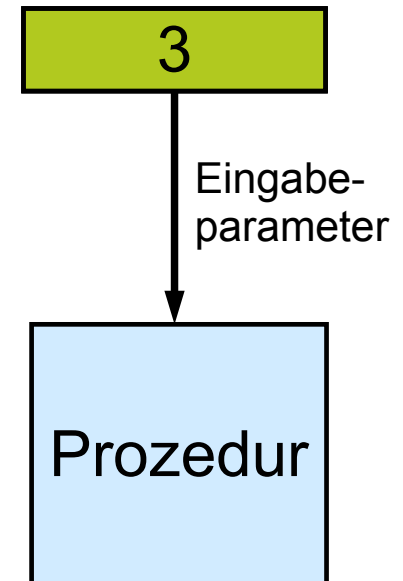
```
  Dim ergebnis As Integer
```

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

```
End Sub
```

Arbeitsweise

- Der Prozedur können Eingabeparameter (Argumente) übergeben werden.
- Diese Eingabeparameter werden in der Prozedur verarbeitet.
- Die Verarbeitung selber ist dem Aufrufer aber nicht bekannt. Der Nutzer kennt nur die Schnittstelle (den Prozedurkopf) nach außen.



Prozedurkopf

« Sub Aufruf() »

« Sub BionomischeFormel(zahlA As Integer, zahlB As Integer) »

- Der Prozedurkopf beginnt mit dem Schlüsselwort « Sub ».
- Dem Schlüsselwort folgt der Name der Prozedur. Der Name ist frei wählbar. Der Name von Ereignisprozeduren wird vom System festgelegt.
- In runden Klammern steht die Argumentliste. Die Argumentliste beschreibt die Anzahl und Art der Eingabeparameter. Die Argumente werden in der Liste durch Kommata getrennt.

Regeln für den benutzerdefinierte Prozedur-Namen

- 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 Prozedur-Name 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 Prozedur-Namen ...

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

Zusammengesetzte Prozedur-Namen

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

Argumentliste ...

« Sub Aufruf() »

« Sub BionomischeFormel(zahlA As Integer, zahlB As Integer) »

- beginnt und endet mit den runden Klammern.
- steht direkt hinter dem Prozedur-Namen.
- kann aus beliebig vielen Argumenten bestehen. Die Argumente werden durch Kommata getrennt.
- kann leer sein. Der Prozedur werden keine Argumente übergeben.

Argumente ...

- « Sub BionomischeFormel(zahlA As Integer, zahlB As Integer) »
- symbolisieren einen Platzhalter für einen Wert, der der Prozedur 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.

Prozeduren ohne Argumente in VBA starten

- Mit einem Mausklicks zwischen « Sub » und « End Sub » wird die Einfügemarke eingeblendet. Die Prozedur ist aktiv. Der Name wird im Kombinationsfeld Prozedur des Code-Fensters angezeigt.
- Der Befehl *Ausführen – Sub/UserForm ausführen* arbeitet die aktuell aktive Prozedur Zeile für Zeile ab.
- Andere Möglichkeiten:
 - <F5> drücken.
 - Grüner Pfeil nach rechts auf der Symbolleiste Voreinstellungen.

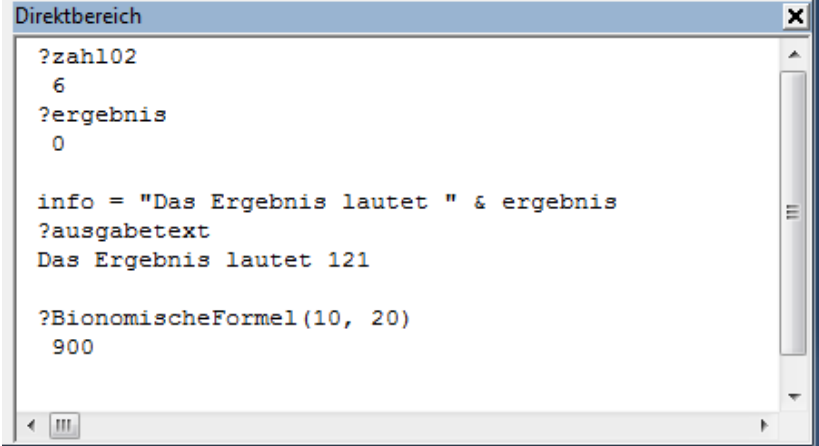


Aus einer anderen Prozedur aufrufen

- « BionomischeFormel zahlA, zahlB »
 - Die Prozedur wird mit ihren Namen aufgerufen.
 - Die Argumente werden nicht durch Klammern begrenzt.
 - Die Argumente werden durch Kommata getrennt.
- « Call BionomischeFormel(zahlA, zahlB) »
 - Die Anweisung beginnt mit dem Schlüsselwort « Call ».
 - Dem Schlüsselwort folgt der Name der aufzurufenden Prozedur.
 - Die Argumentliste wird durch Klammern begrenzt.
 - Die Argumente werden durch Kommata getrennt.

Direktfenster / -bereich zum Testen nutzen

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



```
Direktbereich
?zahl102
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

« Call meineProzeduren.BionomischeFormel(3, 4) ».

« Call Modul.Prozedur(argument01, argument02, ...) ».


- Die Prozedur wird mit Hilfe des Schlüsselwortes « Call » im Direktfenster aufgerufen.
- Anschließend wird die Prozedur mit Hilfe von <F8> Zeile für Zeile durchlaufen.

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 beim Aufruf

```
Sub Aufruf()  
  Dim zahlA As Integer  
  Dim zahlB As Integer  
  
  zahlA = 3  
  zahlB = 4  
  Call BionomischeFormel(zahlA, zahlB)  
End Sub
```



```
Sub BionomischeFormel(zahlA As Integer, zahlB As Integer)  
  Dim ergebnis As Integer  
  
  ergebnis = (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 ...

« Sub Proc(arg01 As Integer, arg02 As Integer) »

« Sub Proc(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.

call bei value ...

- « Sub Proc(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 Prozedur verändert werden.

Grafische Darstellung



Optionale Argumente nutzen

```
Sub Umrechnung(zahl As Double, _  
                Optional faktor As Integer, _  
                Optional einheit As String)
```

```
    Dim ergebnis As Double
```

```
    Dim ausgabe As String
```

```
    ergebnis = zahl * faktor
```

```
    ausgabe = ergebnis & " " & einheit
```

```
End Sub
```

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 Prozeduren genutzt werden.

... schreibgeschützt setzen

```
Sub Umrechnung(zahl As Double, _  
                Optional ByVal faktor As Integer, _  
                Optional einheit As String)
```

```
    Dim ergebnis As Double
```

```
    Dim ausgabe As String
```

```
    ergebnis = zahl * faktor
```

```
    ausgabe = ergebnis & " " & einheit
```

```
End Sub
```

Standardwerte nutzen

```
Sub Umrechnung(zahl As Double, _  
                Optional ByVal faktor As Integer = 10, _  
                Optional einheit As String = "")
```

```
    Dim ergebnis As Double
```

```
    Dim ausgabe As String
```

```
    ergebnis = zahl * faktor
```

```
    ausgabe = ergebnis & " " & einheit
```

```
End Sub
```

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 Prozedur mit optionalen Argumenten

```
Sub Start()
```

```
    Call Umrechnung(10, 100, "cm")
```

```
    Call Umrechnung(10)
```

```
    Call Umrechnung(10, 100)
```

```
    Call 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()
```

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

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

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

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

```
End Sub
```

Benannte Argumente ...

- werden bei der Übergabe an Prozeduren 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 Prozedurkopf und nicht die Position.
- werden in der Argumentliste durch Kommata getrennt. Die Reihenfolge der Argumente ist aber unwichtig.

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.

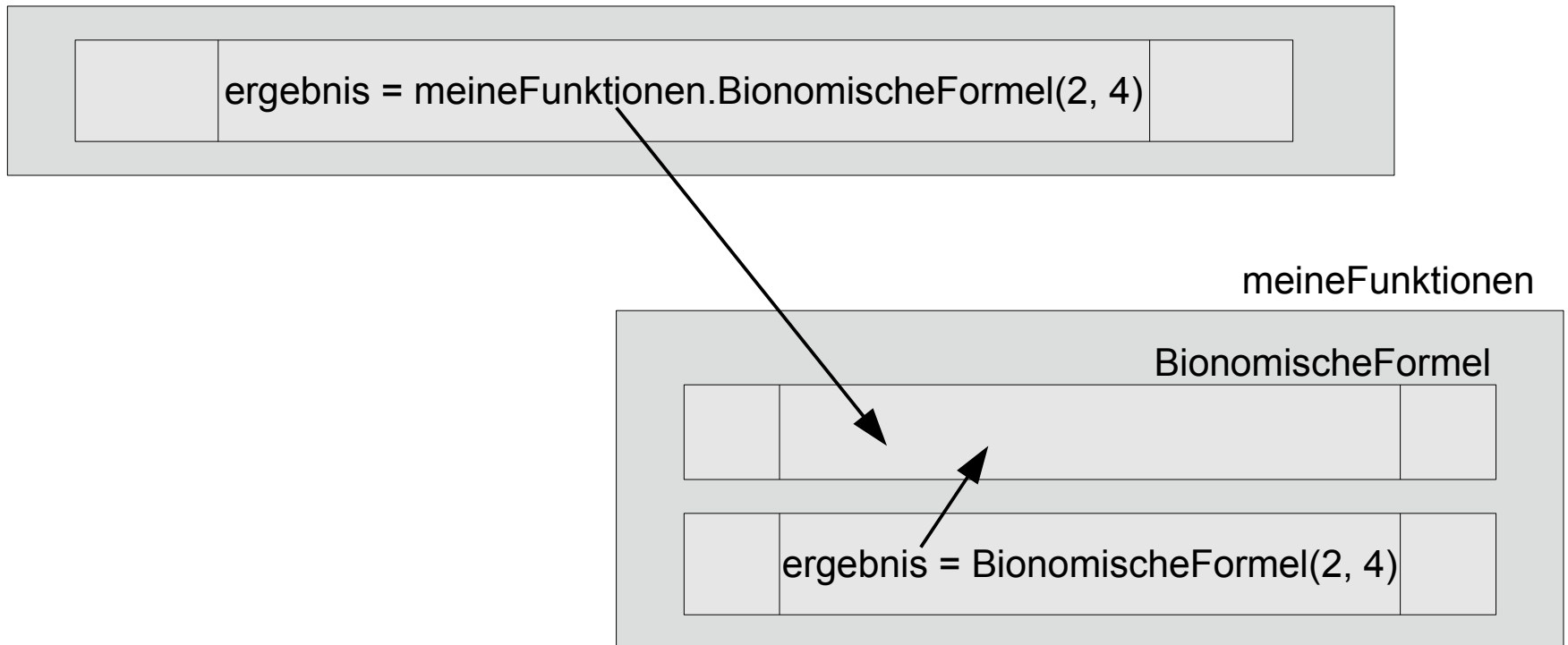
Beispiel

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

Aufruf

« Call meineFunktionen.BionomischeFormel(2, 4) »

« Call 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

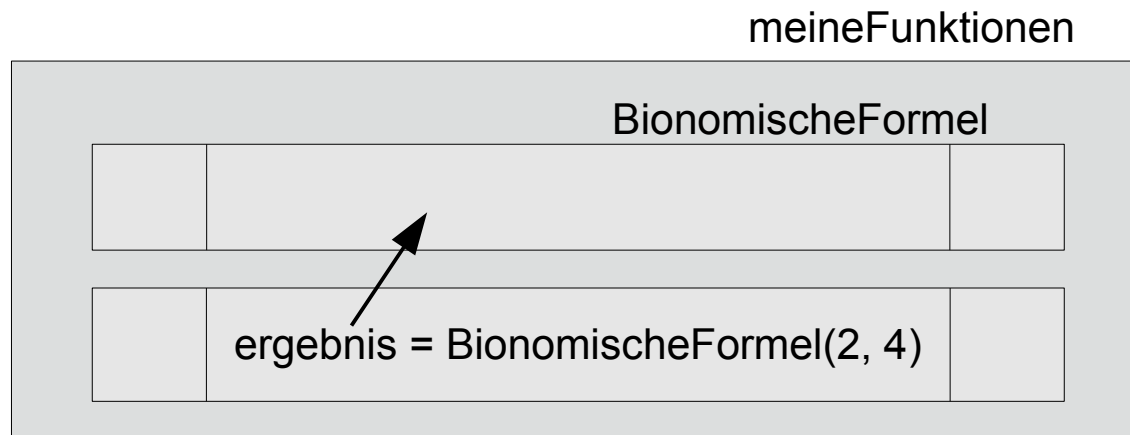
```
Private Sub BionomischeFormel(zahlA As Integer, _  
                             zahlB As Integer)  
  
    Dim ergebnis As Integer  
  
    ergebnis = (zahlA ^ 2) + (2 * zahlA * zahlB) + (zahlB ^ 2)  
  
End Sub
```

Ereignisprozedur

```
Private Sub cmdAddition_Click()  
    Dim wert01 As Integer  
    Dim wert02 As Integer  
    Dim summe As Integer  
    Dim ergebnis As Integer  
  
    wert01 = txtValueFirst.Value  
    wert02 = txtValueSecond.Value  
    summe = wert01 + wert02  
    ergebnis = summe * 2  
    txtSumme.Value = summe  
    txtErgebnis.Value = ergebnis  
End Sub
```

Aufruf

« Call 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.

Statische Variablen ...

- können nur in einer Subroutine definiert werden.
- werden nach dem Verlassen der Subroutine nicht aus dem Speicher gelöscht.
- Die Lebensdauer einer statischen Variablen entspricht der Lebensdauer des Moduls, in dem sie definiert ist.
- welche einen Datentyp „Zahl“ besitzen, beginnen bei 0.

Beispiel

```
Sub Start()  
    PlusEins    ' zaehler = 1  
    PlusEins    ' zaehler = 1  
    PlusEins    ' zaehler = 1  
End Sub
```

```
Sub PlusEins()  
    Dim zaehler As Integer  
  
    zaehler = zaehler + 1  
End Sub
```

```
Sub Start()  
    PlusEins    ' zaehler = 1  
    PlusEins    ' zaehler = 2  
    PlusEins    ' zaehler = 3  
End Sub
```

```
Sub PlusEins()  
    Static zaehler As Integer  
  
    zaehler = zaehler + 1  
End Sub
```

Anfangswerte setzen

```
Sub PlusEins()  
    Static zaehler  
  
    If IsEmpty(zaehler) Then  
        zaehler = 2  
    End If  
  
    zaehler = zaehler + 1  
End Sub
```

```
Sub PlusEins()  
    Static zaehler As Integer  
  
    If zaehler = 0 Then  
        zaehler = 10  
    Else  
        zaehler = zaehler + 1  
    End If  
End Sub
```