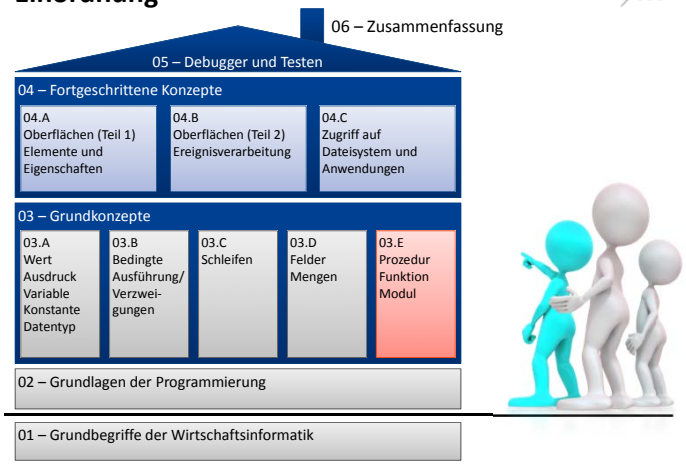
BHT Berliner Hochschule für Technik

Wirtschaftsinformatik 1
LE 07 – Prozeduren, Funktionen und Module

Prof. Dr. Thomas Off
http://www.ThomasOff.de/lehre/beuth/wi1

Einordnung



Inhalt

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Rückblick

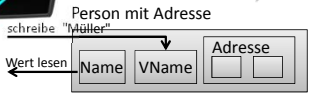
BHT

Rückblick

BHT

Zusammengesetzte Datentypen

- fassen mehrere Eigenschaften definierter Datentypen zusammen
- Repräsentieren häufig Dinge der Realität, z.B. "Person" mit Eigenschaften "Name", "Vorname" und "Adresse"
- Werden als Type definiert und zur Deklaration von Variablenbenutzt
- Zugriff auf einzelne Elemente der Variable des zusammengesetzten Datentypen über Punkt-Notation möglich (Lesen, Schreiben)




```
' Generelle Syntax
Type <Typbezeichner>
 <Eigenschaft1> As <Datentyp>
 <Eigenschaft2> As <Datentyp>
 ' ...
End Type
```

```
' Definition
Type TPerson
 strName As String
 adrWohnanschrift As TAdresse
End Type
```

```
' Deklaration und Nutzung
Dim perTom As TPerson
Let perTom.strName = "Tom"
Debug.Print perTom.strName
```


Rückblick

BHT

Einfache Felder (Array)

- speichern mehrere Werte des gleichen Datentyps
- unter einem gemeinsamen Namen (Bezeichner) zu speichern
- jeden Wert einzeln über einen Index anzusprechen
- innerhalb eines Bereichs zwischen Untergrenze und Obergrenze


```
' Generelle Syntax
Dim <Bez>(<n>) As <DTyp>

Let <Bez>(0) = <WertAusd>
Let <Bez>(1) = <WertAusd>
'...
```

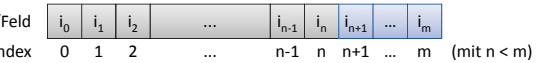
```
' Beispiel
Dim strFeld(2) As String

Let strFeld(0) = "Wert 1"
Let strFeld(1) = "Wert 2"
Let strFeld(2) = "Wert 3"

Debug.Print strFeld(1)
```


Rückblick

BHT

Dynamische Erweiterung des Feldes

- Ober- und Untergrenze legen mögliche Speicherplätze fest
- Erweiterung um zusätzliche Speicherplätze möglich



- Syntax
  - Vorhandene Inhalte werden bei Vergrößerung gelöscht

```
Dim <Bezeichner>() As <Datentyp>
ReDim <Bezeichner>(<n>)
```

  - Vorhandene Inhalte bleiben bei Vergrößerung erhalten

```
ReDim Preserve <Bezeichner>(<m>)
```

Rückblick

Mehrdimensionale Felder

– speichern Daten als Matrix, z.B. mit Zeilen und Spalten

Mehrdimensionales Feld	Index	0	1	2	...	n-1	n
	0	$i_{0,0}$	$i_{0,1}$	$i_{0,2}$	...	$i_{0,n-1}$	$i_{0,n}$
	1	$i_{1,0}$	$i_{1,1}$	$i_{1,2}$	...	$i_{1,n-1}$	$i_{1,n}$
	...	...	...	...	...	...	...
	m	$i_{m,0}$	$i_{m,1}$	$i_{m,2}$	...	$i_{m,n-1}$	$i_{m,n}$

– mehr als zwei Dimensionen möglich

– Syntax

```
' Mehrdimensionales Feld
Dim <Bezeichner>(<n>, <m>, ...) As <Datentyp>
```


Rückblick

Map in Form der VBA-Collection

– dient der Speicherung von Datenelementen auf die anhand eines eindeutigen Schlüssels zugegriffen werden kann

– Generelle Syntax für Deklaration und Initialisierung

```
' Deklaration
Dim <Bezeichner> As Collection
' Initialisierung
Set <Bezeichner> = New Collection
```

```
' Deklaration
Dim colKnd As Collection
' Initialisierung
Set colKnd = New Collection
```

– Generelle Syntax für Zugriffe

```
' Hinzufügen, Lesen und Entfernen
<CollectionBezeichner>.Add <WertAus
<CollectionBezeichner>.Item(<KeyAls
<CollectionBezeichner>.Remove(<KeyA
```

```
' Nutzung
colKnd.Add "Müller", "K1"
colKnd.Add "Yilmaz", "K4"
colKnd.Add "Meier", "K2"
Debug.Print colKnd.Item("K4")
colKnd.Remove("K2")
```


Inhalt

Inhalt

Inhalt

Ausgangspunkt: Unterprogramm

Unterprogramm als wichtiger Bestandteil von Algorithmen

– Teilvorschrift eines Algorithmus
  - die ein sinnvolles Zwischenergebnis produziert und
  - ggf. an mehrere Stellen im Algorithmus verwendet werden kann

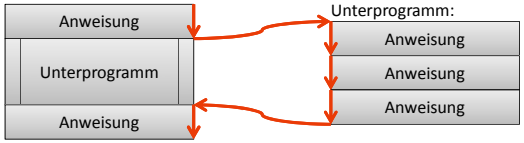
Beispiel

– Unterprogramm für "Wirf es weg" mit den Schritten
  - "Öffne den Mülleimer."
  - "Wirf es hinein."
  - "Schließe den Mülleimer."

– könnte verwendet werden bei
  - Kaffee kochen: "Wirf den Kaffeefilter weg"
  - Pizza zubereiten: "Wirf die leere Packung weg" oder "Wirf die verbrannte Pizza weg".

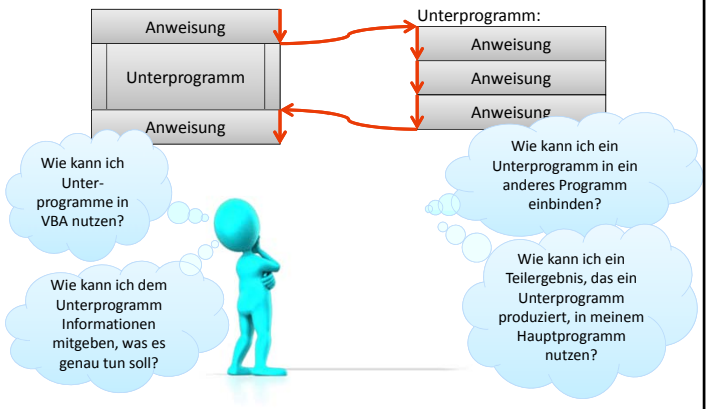
Ausgangspunkt: Unterprogramm

Ablauf des Algorithmus, der Unterprogramm verwendet



– Anstelle der auszuführenden Teilvorschrift wird im Algorithmus ein Verweis auf das Unterprogramm eingebunden
– wird Verweis erreicht, setzt Ausführung innerhalb des Unterprogramms fort
– am Ende des Unterprogramms wird mit nächster Anweisung im Algorithmus fortgefahren, in den Unterprogramm eingebunden

Ausgangspunkt: Unterprogramm



Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

Modul A

```
Sub prozedur1()
    Deklaration 1
    Deklaration 2
    Anweisung abc
    Anweisung def
    Anweisung xyz
    ...
End Sub

Sub prozedur2()
    Anweisung abc
End Sub

Sub prozedur3()
    ...
End Sub
```

Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Modul A

```
Sub prozedur1()
  Deklaration 1
  Deklaration 2
  Anweisung abc
  Anweisung def
  Anweisung xyz
  ...
End Sub
Sub prozedur2()
  Anweisung abc
End Sub
Sub prozedur3()
  ...
End Sub
```


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Modul A

```
Sub prozedur1()
  Deklaration 1
  Deklaration 2
  Anweisung abc
  Aufruf der Prozedur2
  Anweisung xyz
  ...
End Sub
Sub prozedur2()
  Aufruf der Prozedur3
End Sub
Sub prozedur3()
  ...
End Sub
```


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Modul A

```
Sub prozedur1()
  Deklaration 1
  Deklaration 2
  Anweisung abc
  Aufruf der Prozedur 2
  Anweisung xyz
  ...
End Sub
Sub prozedur2()
  Aufruf der Prozedur3
End Sub
Sub prozedur3()
  ...
End Sub
```


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Modul A

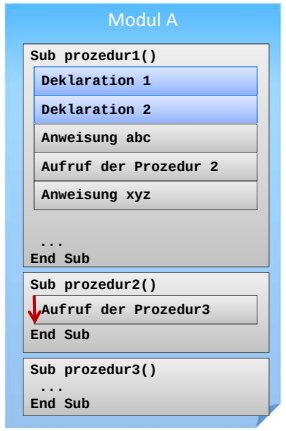
```
Sub prozedur1()
  Deklaration 1
  Deklaration 2
  Anweisung abc
  Aufruf der Prozedur 2
  Anweisung xyz
  ...
End Sub
Sub prozedur2()
  Aufruf der Prozedur3
End Sub
Sub prozedur3()
  ...
End Sub
```

Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

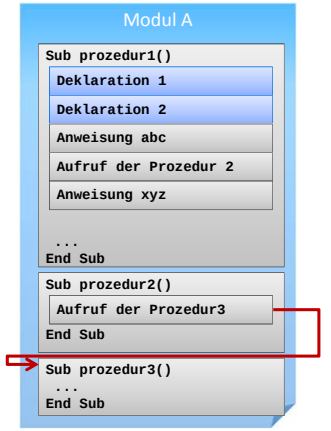


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

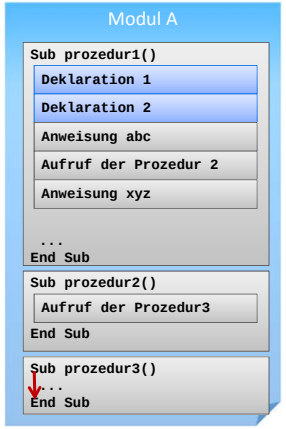


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

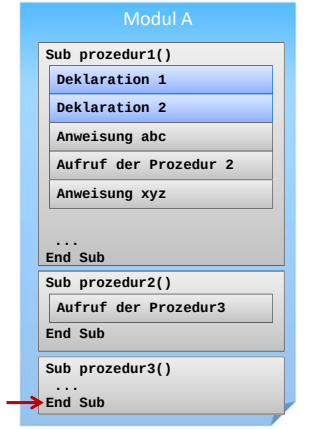


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

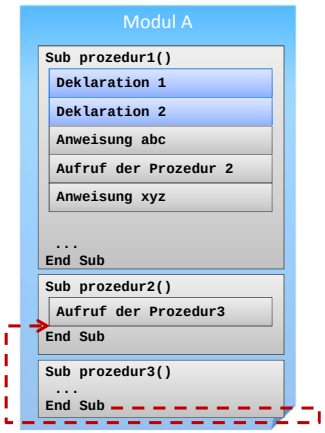


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

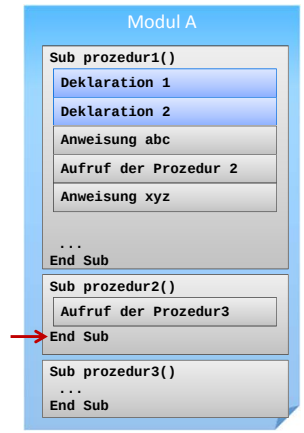


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

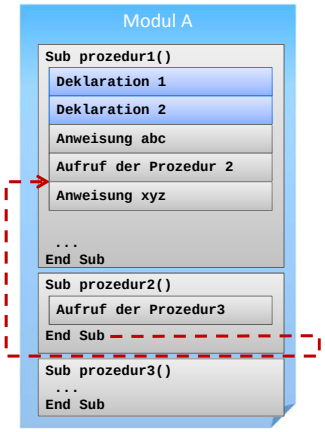


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

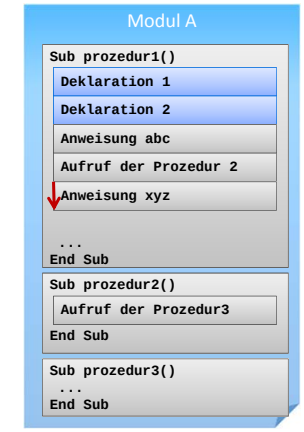


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen



Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Modul A

```
Sub prozedur1()
  Deklaration 1
  Deklaration 2
  Anweisung abc
  Aufruf der Prozedur 2
  Anweisung xyz
  ...
End Sub
Sub prozedur2()
  Aufruf der Prozedur3
End Sub
Sub prozedur3()
  ...
End Sub
```


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Modul A

```
Sub prozedur1()
  Deklaration 1
  Deklaration 2
  Anweisung abc
  Aufruf der Prozedur 2
  Anweisung xyz
  ...
End Sub
Sub prozedur2()
  Aufruf der Prozedur3
End Sub
Sub prozedur3()
  ...
End Sub
```


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Modul A

```
Sub prozedur1()
  Deklaration 1
  Deklaration 2
  Anweisung abc
  Aufruf der Prozedur 2
  Anweisung xyz
  ...
End Sub
Sub prozedur2()
  Aufruf der Prozedur3
End Sub
Sub prozedur3()
  ...
End Sub
```


Prozedur

Syntax

- Aufruf einer Prozedur (einfache Form)

```
Call <BezeichnerDerProzdeur>
```

- Deklaration einer Prozedur (einfache Form)

```
Sub <BezeichnerDerProzdeur>()
<Anweisung(en)>
End Sub
```

Beispiel

```
Sub tueEtwas()
 Debug.Print "Los jetzt!"
 Call machWas
 Debug.Print "Dann ist ja gut."
End Sub

Sub machWas()
 Debug.Print "Ich mach ja schon."
End Sub
```

```
Los jetzt!
Ich mach ja schon.
Dann ist ja gut.
```

Prozedur

Konvention für Bezeichner von Prozeduren

- Bezeichner von Prozeduren zusammengesetzt aus Verb + ggf. Objekt
- Beispiele

```
hinzufuegenProdukt
hinzufuegenKunde
hinzufuegen

loeschenKunde

gibArtikelNr

aktualisiereArtikelPreis
```

Prozedur: Beispiel 07.01

Ziel

- Aufruf mehrerer Prozeduren

Aufgabe

- rechts stehendes "Struktogramm" soll in VBA implementiert werden

Modul A

```
Sub prozedur1()
    "Vor Prozedur 2"
        Aufruf der Prozedur2
    "Nach Prozedur 2"
End Sub

Sub prozedur2()
    "In Prozedur 2"
    "Vor Prozedur 3"
        Aufruf der Prozedur3
    "Nach Prozedur 3"
End Sub

Sub prozedur3()
    "In Prozedur 3"
End Sub
```


Prozedur

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Modul A

```
Sub prozedur1()
    Deklaration 1
    Deklaration 2
    Anweisung abc
    Aufruf der Prozedur 2
    Anweisung xyz
    ...
End Sub

Sub prozedur2()
    Aufruf der Prozedur3
End Sub

Sub prozedur3()
    ...
End Sub
```


Prozedur mit Parametern

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...

Modul A

```
Sub prozedur1()
    Deklaration 1
    Deklaration 2
    Anweisung abc
    Aufruf der Prozedur 2
    Anweisung xyz
    ...
End Sub

Sub prozedur2()
    Aufruf der Prozedur3
End Sub

Sub prozedur3()
    ...
End Sub
```

Prozedur mit Parametern

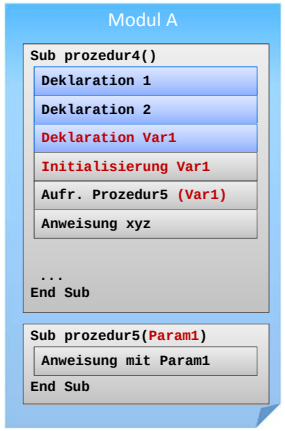
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

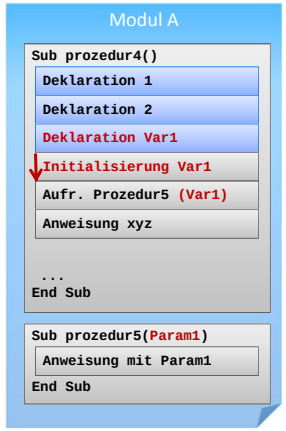
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

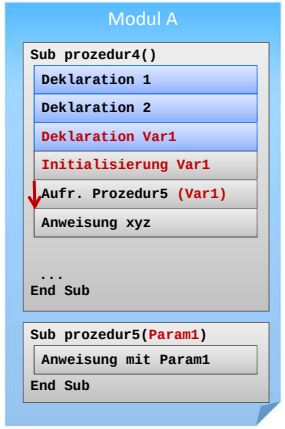
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

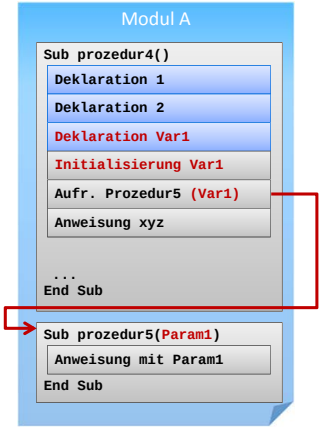
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

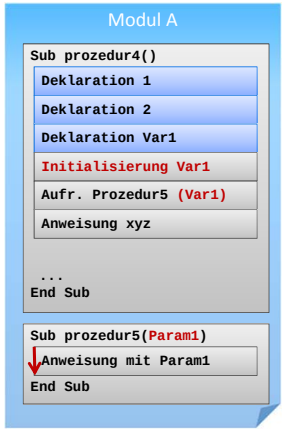
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

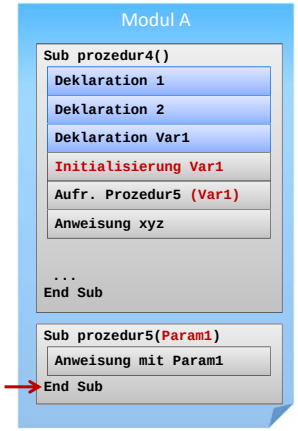
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

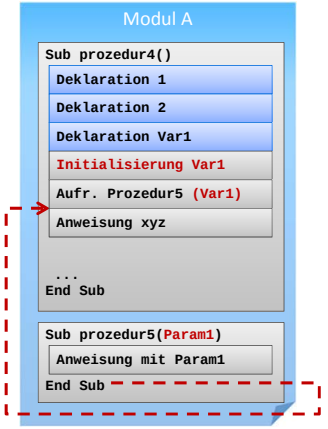
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

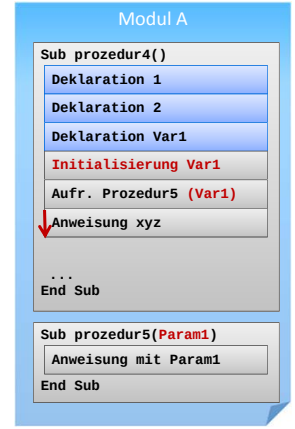
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

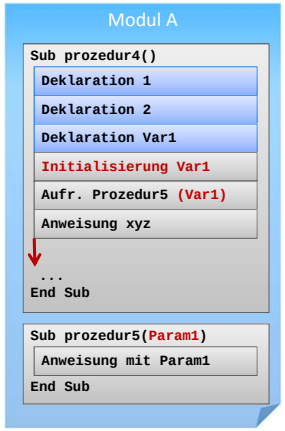
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

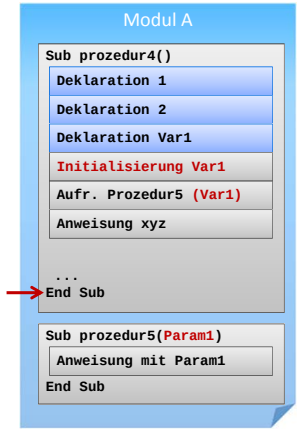
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

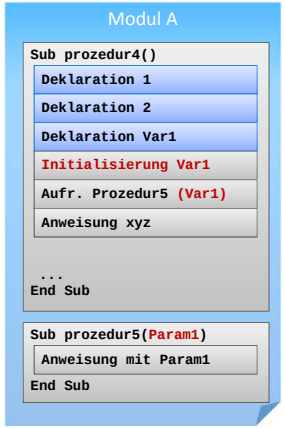
Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...



Prozedur mit Parametern

Syntax

- Aufruf einer Prozedur mit Parametern

```
Call <BezProzedur>(<Bez1>, <Bez2>, ...)
```

- Deklaration einer Prozedur mit Parametern

```
Sub <BezProzedur>(<BezParam1> As <DTyp>, ...)
<Anweisung(en)>
End Sub
```

Konvention

- Parameterbezeichner mit "p" + Präfix des Datentyps + Name
  - Vorname → **`pstrVorname`**
  - Geburtsdatum → **`pdatGebDatum`**

Prozedur mit Parametern

Beispiel

```
Sub losHoleGetraenke()

 Dim strWasser As String
 Dim strSaft As String

 Let strWasser = "Volvic"
 Let strSaft = "Apfelsaft"

 Debug.Print "Los, hole Getränke!"
 Call gehEinkaufen(strWasser)
 Call gehEinkaufen(strSaft)
 Call gehEinkaufen("Cola")
 Debug.Print "Danke."

End Sub

Sub gehEinkaufen(pstrProdukt As String)
 Debug.Print "Ich kaufe: " & pstrProdukt
End Sub
```

```
Los, hole Getränke!
Ich kaufe: Volvic
Ich kaufe: Apfelsaft
Ich kaufe: Cola
Danke.
```

Prozedur mit Parametern

Beispiel (Erweiterung)

```
Sub losHoleGetraenke()

 Dim strWasser As String
 Dim strSaft As String
 Dim bytFlaschen AS Byte
 Dim bytPack AS Byte

 Let strWasser = "Volvic"
 Let bytFlaschen = 3
 Let strSaft = "Apfelsaft"
 Let bytPack = 5

 Debug.Print "Los, hole Getränke!"
 Call gehEinkaufen(strWasser, bytFlaschen)
 Call gehEinkaufen(strSaft, bytPack)
 Debug.Print "Danke."
End Sub
```

```
Los, hole Getränke!
Ich kaufe: 3x Volvic
Ich kaufe: 5x Apfelsaft
Danke.
```

Stückzahl als zweiter Parameter

```
Sub gehEinkaufen(pstrProdukt As String, pbytStueck As Byte)
 Debug.Print "Ich kaufe: " & pbytStueck "x " & pstrProdukt
End Sub
```


Prozedur mit Parametern

Unterschiede in den vorherigen Beispielen

```
Sub losHoleGetraenke()

 ' ...
 Call gehEinkaufen(strWasser)
 Call gehEinkaufen(strSaft)
 ' ...

End Sub
```

Prozedur wird **zweimal** mit **einem** Parameter aufgerufen

```
Sub losHoleGetraenke()

 ' ...
 Call gehEinkaufen(strWasser, bytFlaschen)
 ' ...

End Sub
```

Prozedur wird **einmal** mit **zwei** Parametern aufgerufen

Prozedur mit Parametern: Beispiel 07.02

Ziel

– Aufruf einer Prozedur mit Parameterübergabe

Aufgabe

– rechts stehendes "Struktogramm" soll in VBA implementiert werden

Modul A

```
Sub prozedur4()
 Deklaration von Name
 Name := "Müller"
  Aufruf von Prozedur5
  mit Parameter Name
End Sub
```

```
Sub prozedur5(Nachname)
 "Name ist: " & Nachname
End Sub
```

Prozedur mit Parametern: Beispiel 07.02

BHT

Lösungsansatz (Teil 1)

```
Sub prozedur4()
    Deklaration von Name
    Name := "Müller"
        Aufruf von Prozedur5
        mit Parameter Name
End Sub

Sub prozedur3(Nachname)
    "Name ist: " & Nachname
End Sub
```

```
Sub prozedur4()

 Dim strName As String
 Let strName = "Müller"

 Call prozedur5(strName)

End Sub

Sub prozedur5(pstrNachname As String)

 Debug.Print "Name ist: " & pstrNachname

End Sub
```


Prozedur mit Parametern: Beispiel 07.02

BHT

Lösungsansatz (Teil 2)

```
Sub prozedur4()
    Deklaration von Name
    Name := "Müller"
        Aufruf von Prozedur5
        mit Parameter Name
End Sub

Sub prozedur3(Nachname)
    "Name ist: " & Nachname
End Sub
```

```
Sub prozedur4()

 Dim strName As String
 Let strName = "Müller"

 Call prozedur5(strName)

End Sub

Sub prozedur5(pstrNachname As String)

 Debug.Print "Name ist: " & pstrNachname

End Sub
```

```
Direktbereich
Name ist: Müller
```


Prozedur mit Parametern

BHT

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Prozedur beim Aufruf übergeben werden
- im Kopf der aufgerufenen Prozedur (Signatur) deklariert
- ...

liefert keinen Ergebniswert zurück

```
Modul A
Sub prozedur4()
    Deklaration 1
    Deklaration 2
    Deklaration Var1
    Initialisierung Var1
    Aufr. Prozedur5 (Var1)
    Anweisung xyz
    ...
End Sub

Sub prozedur5(Param1)
    Anweisung mit Param1
End Sub
```


Prozedur vorzeitig verlassen mit Exit Sub

BHT

Gelegentlich kann es sinnvoll sein,

- eine Prozedur zu verlassen, noch bevor diese das **End Sub** erreicht
- dazu dient die Anweisung **Exit Sub**

Beispiel

```
' Bestellung eines Artikels in einer bestimmten Stückzahl
Sub bestelleArtikel(pintStueckzahl As Integer, _
                    pintArtikelNr As Integer)

 If pintStueckzahl <= 0 Then
  ' Wenn kein Stück oder negative Stückzahl Prozedur verlassen
  Debug.Print "Fehler! Stückzahl muss größer 0 sein."
  Exit Sub
 End If

 ' Artikel bestellen
 ' ...
End Sub
```

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Prozedur mit Parametern

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden
kann
– z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
– andere Prozeduren/Funktionen aufrufen
Parameter
– können der Prozedur beim Aufruf übergeben werden
– im Kopf der aufgerufenen Prozedur (Signatur) deklariert
– ...
liefert keinen Ergebniswert zurück

Modul A

```
Sub prozedur4()
    Deklaration 1
    Deklaration 2
    Deklaration Var1
    Initialisierung Var1
    Aufr. Prozedur5 (Var1)
    Anweisung xyz

    ...
End Sub

Sub prozedur5(Param1)
    Anweisung mit Param1
End Sub
```

Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

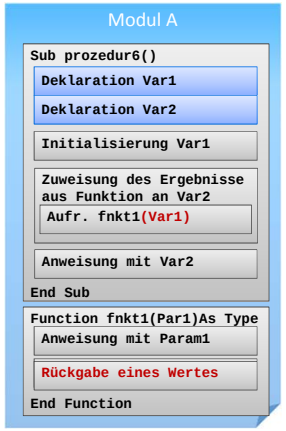
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

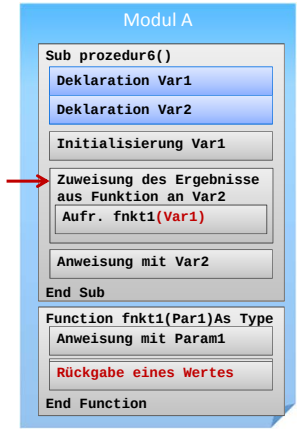
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

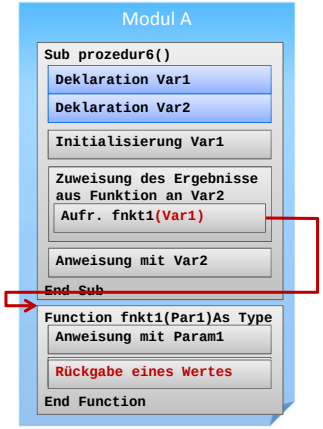
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

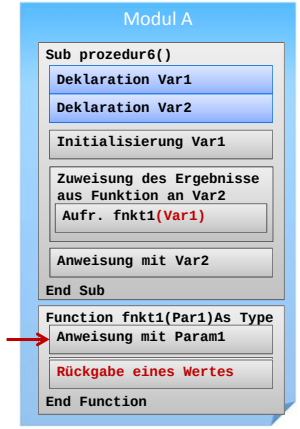
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

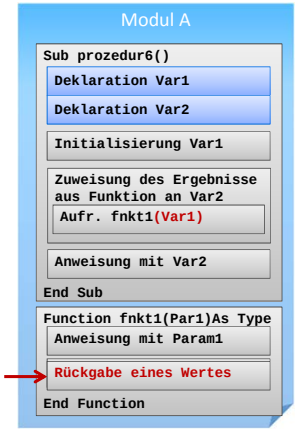
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

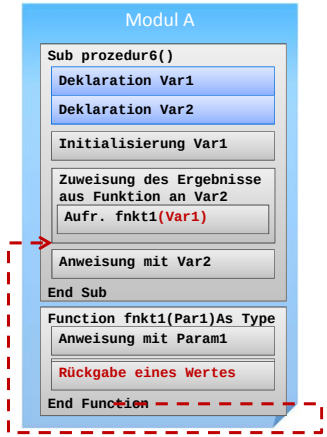
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

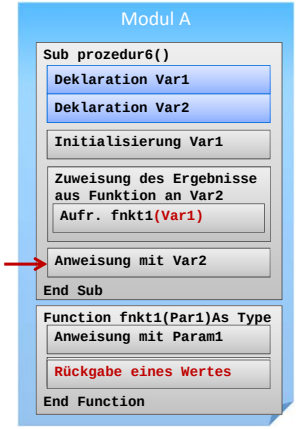
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

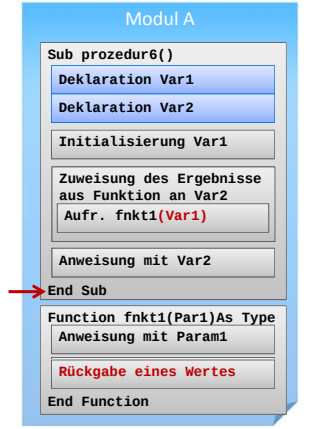
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

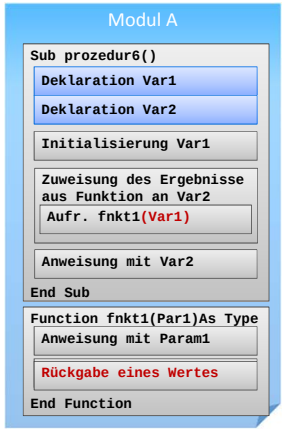
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion

Syntax

- Aufruf einer Funktion mit Parametern und Rückgabewert sollte innerhalb einer Zuweisung erfolgen, um Ergebnis zu verarbeiten

```
Let <Var> = <BezFnkt>(<BezParam1>, <BezParam2>, ...)
```

-

```
Function <BezFnkt>(<BezParam1> As <DTyp>, ...) As <DTyp>
 <Anweisung(en)>
 Let <BezFnkt> = <RückgabeWertOderAusdruck>
End Function
```

Ko

- wie bei Prozeduren

Funktion

Beispiel

```
Sub prozedur6()
 Dim strName As String
 Dim strGruss As String

 Let strName = "Michael"

 Let strGruss = hallo(strName)

 Debug.Print strGruss
End Sub

Function hallo(pstrVorname As String) _
 As String

 Dim strBegruessung As String
 Let strBegruessung = "Hallo " & _
  pstrVorname & "!"
 Let hallo = strBegruessung

End Function
```

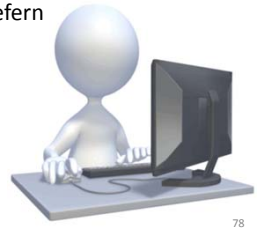
Funktion: Beispiel 07.03

Ziel

- Nutzung von Funktionen und Parametern

Aufgabe:

- Schreiben Sie eine Funktion, die den Nachnamen einer Person und ein Kennzeichen für das Geschlecht als Parameter übergeben bekommt
- Sie soll die die Anrede der Person "Sehr geehrte Frau " bzw. "Sehr geehrter Herr" als String zurückliefern
- Rufen Sie die Funktion mit mehreren Beispielwerten aus einer anderen Prozedur auf



Funktion

Zusammenfassung von Deklarationen und Anweisungen, die ausgeführt werden

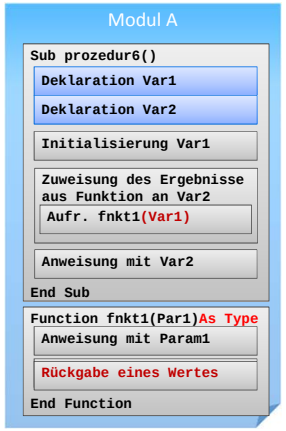
kann

- z.B. aus anderen Prozeduren/Funktionen aufgerufen werden
- andere Prozeduren/Funktionen aufrufen

Parameter

- können der Funktion beim Aufruf übergeben werden
- im Kopf der aufgerufenen Funktion (Signatur) deklariert
- ...

liefert einen Ergebniswert und wird in Zuweisung verwendet



Funktion vorzeitig Verlassen mit Exit Function

Gelegentlich kann es sinnvoll sein,

- eine Funktion zu verlassen, noch bevor diese das **End Function** erreicht
- dazu dient die Anweisung **Exit Function**

Beispiel

```
' Division (Dividend geteilt durch Divisor)
Function dividiere(pintDividend As Integer, _
           pintDivisor As Integer) As Double

 If pintDivisor = 0 Then
   ' Division durch 0 würde Programm abbrechen,
   ' deshalb vorher prüfen
   Debug.Print "Fehler! Division durch 0."
   Exit Function
 End If

 '...
End Function
```

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Parameter in Prozeduren und Funktionen

Unterscheidung in

– formale Parameter
- in der Deklaration der Prozedur/Funktion angegebener Parameter
- vollständig deklariert in Prozedur/Funktion mit Bezeichner und Datentyp

– tatsächliche Parameter
- legen die tatsächlichen Werte der formalen Parameter beim Aufruf fest
- synonym: Argumente oder "aktueller" Parameter (actual Parameter)

```
Sub proz7()
 Dim strName As String

 Let strName = "Michael"

 Call proz8(strName)

End Sub


Sub proz8(pstrVorname As String)

 Debug.Print "Hallo " & _
  pstrVorname & "!"

End Sub
```


Parameter in Prozeduren und Funktionen

Arten der Parameterübergabe

– standardmäßig in VBA Parameterübergabe per Referenz

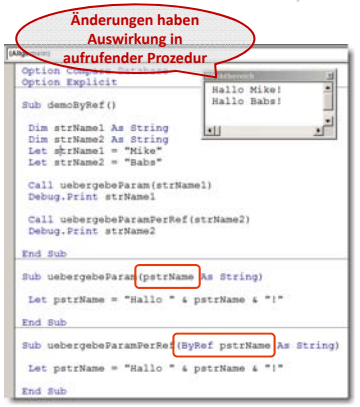
– alternativ Parameter-übergabe per Wert möglich

Parameter in Prozeduren und Funktionen

Arten der Parameterübergabe

– standardmäßig in VBA Parameterübergabe per Referenz
- Änderungen am Parameter-wert werden in aufrufenden Prozeduren/Funktionen sichtbar
- auch durch Schlüsselwort "ByRef" möglich

– alternativ Parameter-übergabe per Wert möglich

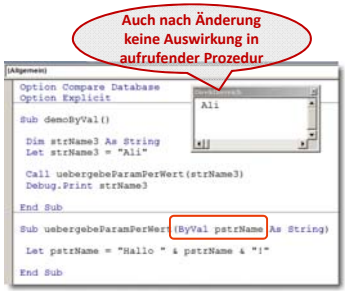


Parameter in Prozeduren und Funktionen

BHT

Arten der Parameterübergabe

- standardmäßig in VBA Parameterübergabe per Referenz
- alternativ Parameterübergabe per Wert möglich
  - Änderungen an Parametern nur innerhalb der aufgerufenen Prozedur/Funktion
  - haben keine Auswirkungen in der aufrufenden Prozedur/Funktion
  - Schlüsselwort "ByVal"



Parameter in Prozeduren und Funktionen

BHT

Arten der Parameterübergabe

- standardmäßig in VBA Parameterübergabe per Referenz
- alternativ Parameterübergabe per Wert möglich

Parameter: Beispiel 07.04

BHT

Ziel

- Verschiedene Möglichkeiten zur Parameterübergabe nutzen

Aufgabe

- Schreiben Sie eine Prozedur in der Sie eine Variable für einen Nachnamen deklarieren und initialisieren.
- Rufen Sie aus dieser Prozedur eine andere Prozedur auf, der Sie zunächst per Wert die Variable übergeben.
- Die aufgerufene Prozedur soll den übergebenen Parameterwert um eine Begrüßung ergänzen (z.B. "Hallo").
- Geben Sie die Begrüßung dann in der Prozedur im Direktbereich aus.
- Geben Sie in der aufrufenden Prozedur die Variable für den Nachnamen aus.

Parameter: Beispiel 07.05

BHT

Ziel

- Verschiedene Möglichkeiten zur Parameterübergabe nutzen

Aufgabe

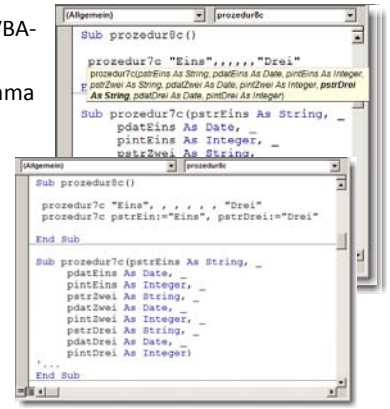
- Ändern Sie das vorherige Beispiel so, dass die Parameterübergabe nun per Referenz erfolgt
- Welche Änderung stellen Sie fest? Wie kann sie erklärt werden?

Parameter in Prozeduren und Funktionen

BHT

Lange Parameterlisten

- Eingabeunterstützung im VBA-Editor
- viele Parameter durch Komma getrennt aufzählbar
- alternativ Verwendung benannter Parameter

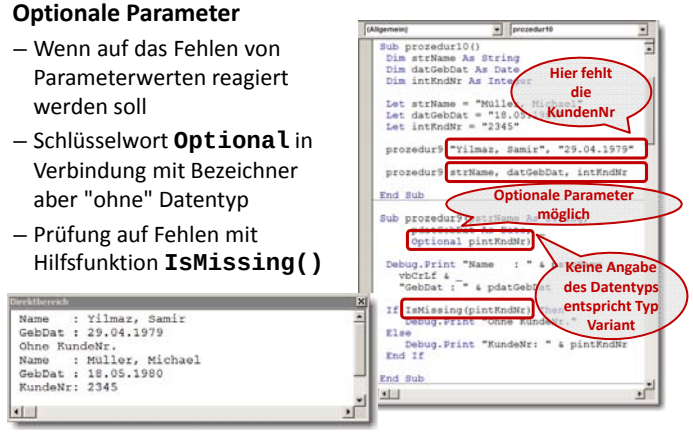


Parameter in Prozeduren und Funktionen

BHT

Optionale Parameter

- Wenn auf das Fehlen von Parameterwerten reagiert werden soll
- Schlüsselwort **`Optional`** in Verbindung mit Bezeichner aber "ohne" Datentyp
- Prüfung auf Fehlen mit Hilfsfunktion **`IsMissing()`**



Parameter: Beispiel 07.06

BHT

Ziel

- Verschiedene Möglichkeiten zur Parameterübergabe nutzen

Aufgabe

- Erweitern Sie das vorherige Programm so, dass das Name ein optionaler Parameter ist
- Wird kein Name ausgegeben soll eine neutrale Ausgabe erfolgen (z.B. "Hallo Sie da!")

Inhalt

BHT

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Module

Strukturierung großer Anwendungen
– in fachliche Komponenten
– in Schichten

Online Shop mit Kundenverwaltung, Produktkatalog Bestellungsabwicklung

Module

Strukturierung großer Anwendungen
– in fachliche Komponenten
- umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
- kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]

Online Shop mit Kundenverwaltung, Produktkatalog Bestellungsabwicklung

Module

Strukturierung großer Anwendungen

– in fachliche Komponenten
- umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
- kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]

Module

Strukturierung großer Anwendungen

– in fachliche Komponenten
- umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
- kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]

Module

Strukturierung großer Anwendungen

– in fachliche Komponenten
- umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
- kann seine Bestandteile anderen Modulen zur Verfügung stellen

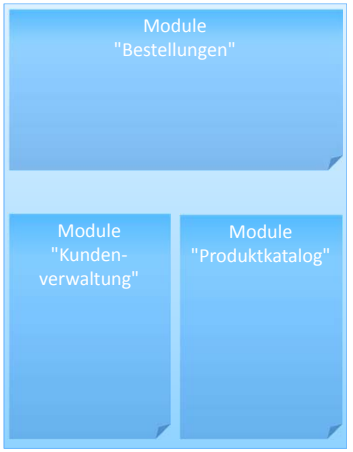
– in Schichten [...]

Module

Strukturierung großer Anwendungen

– in fachliche Komponenten
- umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
- kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]



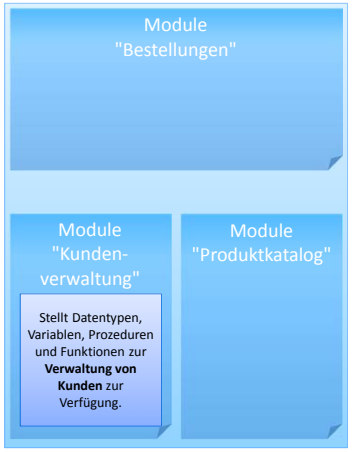
Module

BHT

Strukturierung großer Anwendungen

– in fachliche Komponenten
- umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
- kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]



Module

BHT

Strukturierung großer Anwendungen

– in fachliche Komponenten
- umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
- kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]

Module

BHT

Strukturierung großer Anwendungen

– in fachliche Komponenten
- umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
- kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]

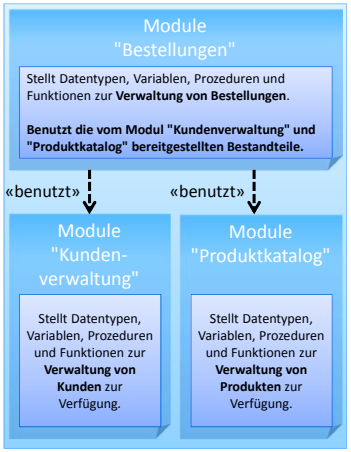
Module

BHT

Strukturierung großer Anwendungen

– in fachliche Komponenten
- umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
- kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]



Module

Strukturierung großer Anwendungen

– in fachliche Komponenten
  - umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
  - kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]

Module "Bestellungen"

```
Type TBestellung
'...
End Type

Dim bstBstl() As TBestellung

Function finde(pintBstID As
Integer) As TBestellung
'...
End Function

Sub zeige(pintBstID As Integer)
'...
End Sub

Sub hinzufuegen(pintBstID As
Integer, kndKunde As TKunde,
prdArtikel() As TProdukt)
'...
End Sub
```


Module

Syntax für Deklaration von Modulbestandteilen

– Typdefinitionen
– Variablen und Konstanten
– Funktionen
– Prozeduren

Module

Syntax für Deklaration von Modulbestandteilen

– Typdefinitionen

```
' Generelle Syntax
Type <Typbezeichner>
 <Eigenschaft1> As <Datentyp>
 <Eigenschaft2> As <Datentyp>
 ' ...
End Type
```

```
' Beispieldefinition
Type TPerson
 strName As String
 adrWohnanschrift As TAdresse
End Type
```

– Variablen
– Funktionen
– Prozeduren

Module

Syntax für Deklaration von Modulbestandteilen

– Typdefinitionen
– Variablen und Konstanten

```
' Generelle Syntax
Dim <VarBezeichner> As <Datentyp> ' Einfache Variable
Const <VarBezeichner> As <Datentyp> = <WertAusdr> ' Konstante
Dim <VarBezeichner>() As <Datentyp> ' Dynamisches Feld
' ...
```

```
' Beispieldeklaration
Dim strName As String
Const MWST As Single = 0.19
Dim strKundenNamen() As String
Dim kndKunden() As TKunde
Dim prdProdukte(1 To 100) As TProdukt
```

– Funktionen
– Prozeduren

Module

Syntax für Deklaration von Modulbestandteilen

– Typdefinitionen
– Variablen und Konstanten
– Funktionen

```
' Generelle Syntax
Function <BezFnkt>(<BezParam1> As <DTyp>, ...) As <DTyp>
 <Anweisung(en)>
 Let <BezFnkt> = <RückgabeWertOderAusdruck>
End Function
```

```
' Beispiel
Function gibAnredeKunde(pkndKunde As TKunde) As String
 Let gibAnredeKunde = "Sehr geehrte(r) " & pkndKunde.strName
End Function
```

– Prozeduren

Module

Syntax für Deklaration von Modulbestandteilen

– Typdefinitionen
– Variablen und Konstanten
– Funktionen
– Prozeduren

```
' Generelle Syntax
Sub <BezProzedur>(<BezParam1> As <DTyp>, ...)
 <Anweisung(en)>
End Sub
```

```
' Beispiel
Sub gibAusKundeAnrede(pkndKunde As TKunde)
 Debug.Print "Sehr geehrte(r) " & pkndKunde.strName
End Sub
```


Module

Syntax für Deklaration von Modulbestandteilen

– Typdefinitionen
– Variablen und Konstanten
– Funktionen
– Prozeduren
– ...

Module

Syntax für den Zugriff auf Modulbestandteile

– des eigenen Moduls direkt durch Verwendung des Bezeichners
– anderer Module durch Verwendung der Punkt Notation

```
' Generelle Syntax
<BezeichnerAnderesModul>.<BezeichnerDesModulbestandteils>
```

```
' Beispiele

' Zugriff auf Variable/Feld in anderem Modul
Debug.Print mdlKunden.intLetzteKundeNr
Let kndKunde42 = mdlKunde.kndKundenliste(42)

' Funktions- und Prozeduraufruf in anderem Modul
Let kndKunde42 = mdlKunden.gibKunde(42)
Call mdlProdukte.zeigeAlleProdukte
```

Module

Strukturierung großer Anwendungen

– in fachliche Komponenten
  - umfasst Deklarationen (z.B. Typen, Variablen), Prozeduren und Funktionen als Bestandteile
  - kann seine Bestandteile anderen Modulen zur Verfügung stellen

– in Schichten [...]

Module "Bestellungen"

```
Type TBestellung
'...
End Type

Dim bstBstl() As TBestellung

Function finde(pintBstID As
Integer) As TBestellung
'...
End Function

Sub zeige(pintBstID As Integer)
'...
End Sub

Sub hinzufuegen(pintBstID As
Integer, kndKunde As TKunde,
prdArtikel() As TProdukt)
'...
End Sub
```


Module

Strukturierung großer Anwendungen

– in fachliche Komponenten
– in Schichten

Online Shop mit Kundenverwaltung, Produktkatalog Bestellungsabwicklung

Module

Strukturierung großer Anwendungen

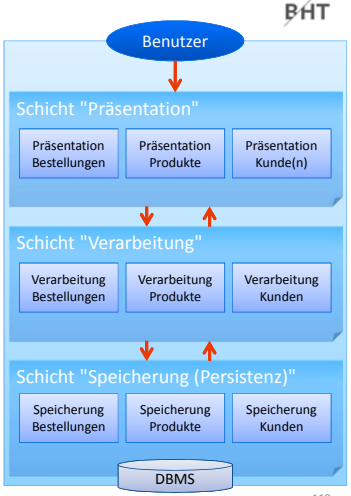
– in fachliche Komponenten
– in Schichten
  - Schichtenbildung als Architekturprinzip
    – Obere Schicht greift nur auf Bestandteile darunter liegender Schichten zu
    – Untere Schichten sind unabhängig von darüber liegenden
  - Beispiel: Trennung zwischen Präsentation, Verarbeitung und Speicherung von Daten

Online Shop mit Kundenverwaltung, Produktkatalog Bestellungsabwicklung

Module

Strukturierung großer Anwendungen

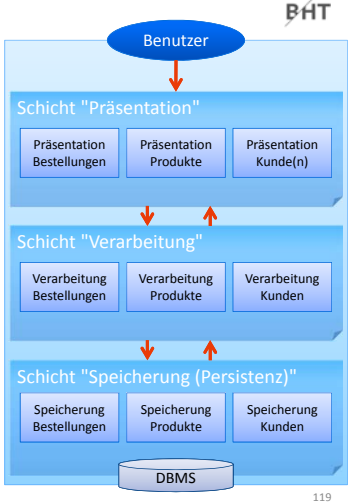
– in fachliche Komponenten
– in Schichten
  - Schichtenbildung als Architekturprinzip
    – Obere Schicht greift nur auf Bestandteile darunter liegender Schichten zu
    – Untere Schichten sind unabhängig von darüber liegenden
  - Beispiel: Trennung zwischen Präsentation, Verarbeitung und Speicherung von Daten



Module

Strukturierung großer Anwendungen

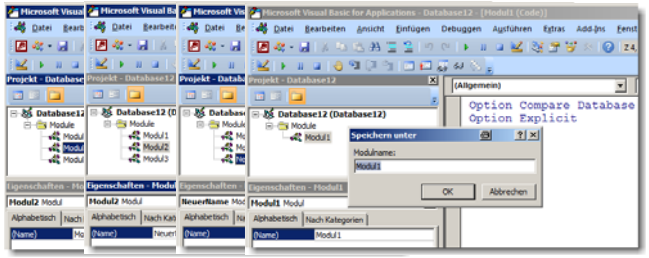
– in fachliche Komponenten
– in Schichten



Module

Umsetzung in VBA

– Module im Projektexplorer sichtbar
– Änderungen des Namens eines Moduls
  • im Fenster "Eigenschaften"
  • beim erstmaligen Speichern eines neue angelegten Moduls



Module

Konvention

– Variante 1: Präfix "mdl" + Modulbezeichner im Plural
– Variante 2: Präfix "mdl" + Modulbezeichner im Plural + Postfix für Zugehörigkeit zu einer Schicht)

Beispiele

```
' Variante 1
mdlKunden
mdlBestellungen
mdlProdukte

' Variante 2 (Vorschläge)
mdlKundenView
mdlKundenCtrl
mdlKundenData
```

Module: Beispiel 07.07

Ziel

– Erstellen eines Moduls zur Verwaltung von Kunden

Aufgabe

– Definieren Sie einen Typ für Kunden (Name, Vorname, KundenNr)
– Deklarieren Sie innerhalb des Moduls ein dynamisches Feld
– Schreiben Sie Prozeduren innerhalb des Moduls für die folgenden Aufgaben:
  • Hinzufügen eines Kunden
  • Ermitteln eines Kunden anhand seiner KundenID
  • Ermitteln des Namens eines Kunden anhand seiner KundenID
  • Initialisierung mit drei Kunden, die zur Liste der Kunden hinzugefügt werden

Module: Beispiel 07.08

Ziel

– Erstellen eines Moduls zur Verwaltung von Produkten

Aufgabe

– Definieren Sie einen Typ für Produkte (Bezeichnung, Preis)
– Deklarieren Sie innerhalb des Moduls ein dynamisches Feld
– Schreiben Sie Prozeduren innerhalb des Moduls für die folgenden Aufgaben:
  • Hinzufügen eines Produktes
  • Ermitteln eines Produktes anhand der ProduktID
  • Ermitteln des Preises eines Produktes anhand der ProduktID
  • Initialisierung mit sechs Produkten, die zur Liste der Produkte hinzugefügt werden

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

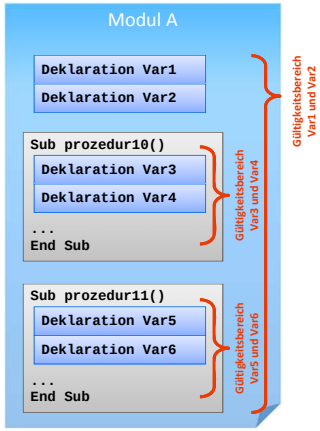
Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Gültigkeitsbereiche

BHT

Variablen und Konstanten

- sind innerhalb des Bereichs verwendbar, in dem sie deklariert wurden
- man nennt diesen Bereich "Gültigkeitsbereich"
  - Wurde Variable in einer Prozedur/Funktion deklariert ➔ innerhalb der Prozedur/Funktion gültig
  - Wurde Variable in einem Modul deklariert ➔ (mind.) in allen Prozeduren/Funktionen des Moduls gültig, abhängig von Ihrer Sichtbarkeit (nächste Folie) ggf. auch in anderen Module

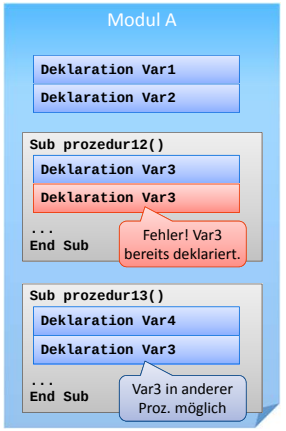


Gültigkeitsbereiche

BHT

Konsequenzen

- Weil Variablenbezeichner eindeutig sein müssen, sind verschiedene Variablen mit gleichem Bezeichner innerhalb der gleichen Prozedur/Funktion nicht möglich.
- In einem anderen Gültigkeitsbereich kann es eine andere Variable mit dem gleichen Bezeichner geben. Beide repräsentieren verschiedene Variablen (und Speicherbereiche).



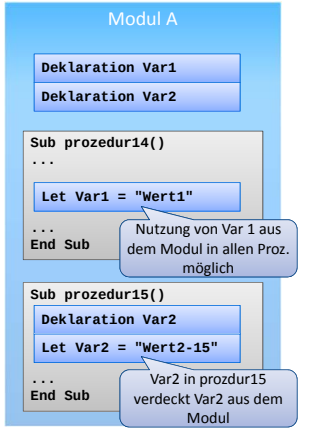
Gültigkeitsbereiche

BHT

Besonderheit "Verdecken"

- Wenn
  - eine Variable in einem übergeordneten Gültigkeitsbereich existiert und
  - sie in einem untergeordneten Gültigkeitsbereich nochmals deklariert wird
- dann
  - verdeckt die Variable im untergeordneten Gültigkeitsbereich die übergeordnete

In VBA kein Zugriff auf verdeckte Variablen



Module: Beispiel 07.09

BHT

Ziel

- Gültigkeitsbereiche und Verdecken nutzen

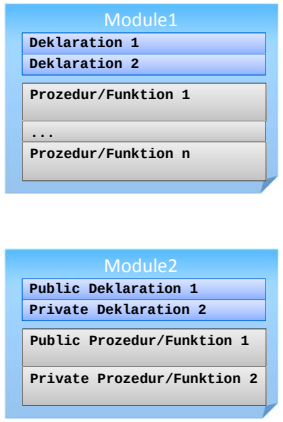
Aufgabe

- Implementieren Sie ein Modul
  - in dem Sie eine Variable 1 deklarieren
  - mit einer Prozedur A, die
    - eine Variable 1 und eine Variable 2 deklariert
    - beide Variablen initialisiert und die Werte ausgibt
  - mit einer Prozedur B, die
    - eine Variable 2 und eine Variable 3 deklariert
    - beide Variablen initialisiert
    - der Variable 1 einen Wert zuweist
    - die Werte der Variablen ausgibt
  - mit einer demo0709, die
    - die Variable 1 initialisiert und ausgibt
    - die beiden Prozeduren A und B aufruft
    - die Variable 1 ausgibt

Sichtbarkeit

Module können ihre Bestandteile anderen Modulen zur Verfügung stellen

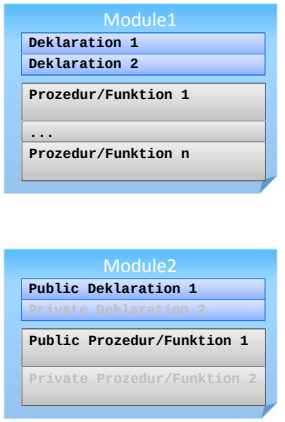
- Sichtbarkeit von Deklarationen und Prozeduren/Funktionen festlegen
- Standardmäßig alles sichtbar



Sichtbarkeit

Module können ihre Bestandteile anderen Modulen zur Verfügung stellen

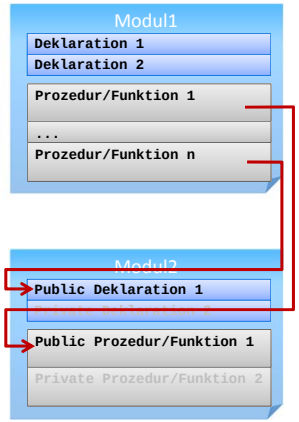
- Sichtbarkeit von Deklarationen und Prozeduren/Funktionen festlegen
- Standardmäßig alles sichtbar



Sichtbarkeit

Module können ihre Bestandteile anderen Modulen zur Verfügung stellen

- Sichtbarkeit von Deklarationen und Prozeduren/Funktionen festlegen
- Standardmäßig alles sichtbar



Sichtbarkeit

Syntax für den Zugriff auf sichtbare Modulbestandteile

- des eigenen Moduls direkt durch Verwendung des Bezeichners
- anderer Module durch Verwendung der Punkt Notation

```
' Generelle Syntax
<BezeichnerAnderesModul>.<BezeichnerDesModulbestandteils>
```

```
' Beispiele

' Zugriff auf Variable/Feld in anderem Modul
Debug.Print mdlKunden.intLetzteKundeNr
Let kndKunde42 = mdlKunde.kndKundenliste(42)

' Funktions- und Prozeduraufruf in anderem Modul
Let kndKunde42 = mdlKunden.gibKunde(42)
Call mdlProdukte.zeigeAlleProdukte
```

Sichtbarkeit

Syntax: Schlüsselwort Private oder Public in Verbindung mit

– Deklaration von Variablen auf Modulebene (anstelle von Dim)

```
Private | Public <Variable> As <Datentyp>
```

– Deklaration von Konstanten auf Modulebene

```
Private | Public Const <Konstante> As <DTyp> = <WertAusd>
```

– Zusammengesetzten Datentypen

```
Private | Public Type <Typbezeichner>
 <Eigenschaft> As <Datentyp>
End Type
```

– Prozeduren und Funktionen

```
Private | Public Sub <BezProzedur>(<Param> As <DTyp>)
Private | Public Function <BezFnkt>(<Param> As <DTyp>) As<DTyp>
```


Sichtbarkeit

Beispiele

```
Option Compare Database
Option Explicit

Public Type TProdukt
 intProdNr As Integer
 strBezeichung As String
 curPreis As Currency
End Type

Public Const curMwSt As Currecny = 0.19
Private prdMeineProdukte() As TProdukt

Public Sub hinzufuegenProdukt(prdProdukt As TProdukt)
 ' ...
End Sub

'...
```


Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen

Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Geheimnisprinzip

Ziel ist …

– leicht änderbare Software entwickeln, d.h. Änderungen in einer Prozedur/ Funktion bzw. in einem Modul wirken sich nicht auf andere Bestandteile aus

– leichte Nutzung von vorhandenen Funktionen/Prozeduren und Modulen ermöglichen, ohne ihre interne Umsetzung bzw. Struktur kennen zu müssen

Geheimnisprinzip

Wird erreicht durch …

– Verstecken der internen Umsetzung vor dem Zugriff von außen, durch Beschränkung der Sichtbarkeit interner Elemente und Strukturen

– Definition (und Dokumentation) einer Schnittstelle, die nach außen sichtbar ist und genutzt werden kann

Geheimnisprinzip

Beispiele

```
Option Compare Database
Option Explicit

' Liefert Anzahl Tage zwischen Anfang und Ende; wenn Anfang
' hinter Ende, dann 0
Public Function zaehleTage(pdatAnfang As Date, pdatEnde As
Date) As Integer

 Let zaehleTage = DateDiff("d", pdatAnfang, pdatEnde)

End Function
```

Alternative Umsetzung mit der Hilfsfunktion DateDiff lässt die Schnittstelle unverändert. Details der geänderten internen Umsetzung außen nicht sichtbar.

```
 For datStart = pdatAnfang + 1 To pdatEnde
  Let intAnzahl = intAnzahl + 1
 Next

 Let zaehleTage = intAnzahl

End Function
```

Umsetzung durch Abzählen der Tage zwischen Anfang und Ende.

Prozedur mit Parametern: Beispiel 07.10

Ziel

– Umsetzung des Geheimnisprinzips für Module zur Verwaltung von Kunden, Produkten und Bestellungen

Aufgabe

– Definieren Sie ein Modul für Bestellungen mit
  - einem Typ für Bestellungen (BestellNr, Datum, KundeNr, ProduktIDs)
  - einem dynamischen Feld für Bestellungen
  - Prozeduren zum
    – Hinzufügen einer Bestellung für einen Kunden mit einer Liste von Produkten
    – Ausgabe einer Bestellung
    – Ausgabe aller Bestellungen
    – Initialisierung mit drei Bestellungen, die zur Liste der Bestellungen hinzugefügt werden

– Passen Sie die Sichtbarkeit der Modulbestandteile aus den Beispielen 07.07 und 07.08 (Module für Kunden und Produkte) so an, dass das Geheimnisprinzip gewahrt bleibt

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen

Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Inhalt

Einordnung
Rückblick
Ausgangspunkt
Formen von Unterprogrammen
– Prozedur
– Funktion
– Parameter in Prozeduren und Funktionen
Module
– Einsatzmöglichkeiten und Verwendung in MS Access
– Gültigkeitsbereiche und Sichtbarkeit
– Geheimnisprinzip
Abschluss und Ausblick

Abschluss

Prozedur

– Form eines Unterprogramms, das keinen Ergebniswert zurückliefert

– Aufruf einer Prozedur (einfache Form)

```
Call <BezeichnerDerProzdeur>
```

– Deklaration einer Prozedur (einfache Form)

```
Sub <BezeichnerDerProzdeur>()
<Anweisung(en)>
End Sub
```

Konvention für Bezeichner von Prozeduren

– Bezeichner von Prozeduren zusammengesetzt aus Verb + ggf. Objekt

– Beispiele

Abschluss

Prozedur mit Parametern

– Aufruf einer Prozedur mit Parametern

```
Call <BezProzdeur>(<BezParam1>, <BezParam2>, ...)
```

– Deklaration einer Prozedur mit Parametern

```
Sub <BezProzdeur>(<BezParam1> As <DTyp>, ...)
<Anweisung(en)>
End Sub
```

Konvention

– Parameterbezeichner mit "p" + Präfix des Datentyps + Name
  - Vorname → **pstrVorname**
  - Geburtsdatum → **pdatGebDatum**

Abschluss

Funktion mit Parametern und Rückgabewert

– ist eine Form des Unterprogramms und liefert einen Ergebniswert zurück

– Aufruf einer Funktion mit Parametern und Rückgabewert sollte innerhalb einer Zuweisung erfolgen

```
Let <Var> = <BezFnkt>(<BezParam1>, <BezParam2>, ...)
```

– Deklaration einer Funktion mit Parametern und Rückgabewert

```
Function <BezFnkt>(<BezParam1> As <DTyp>, ...) As <DTyp>
 <Anweisung(en)>
 Let <BezFnkt> = <RückgabeWertOderAusdruck>
End Function
```


Abschluss

Modul

– dient der Gliederung großer Programme in einzelne Teile
  - fachliche Komponenten (z.B. Bestellungen, Kunden, Produkte)
  - in Schichten (z.B. für Präsentation, Verarbeitung und Speicherung)

– kann anderen Modulen Prozeduren, Funktionen und Variablen zur Verfügung stellen

– Namenskonvention
  - "mdl" + Bezeichnung im Plural (ggf. mit Postfix zur Zuordnung zu einer Schicht)

Abschluss

BHT

Syntax für den Zugriff auf Modulbestandteile

– des eigenen Moduls direkt durch Verwendung des Bezeichners
– anderer Module durch Verwendung der Punkt Notation

```
' Generelle Syntax
<BezeichnerAnderesModul>.<BezeichnerDesModulbestandteils>
```

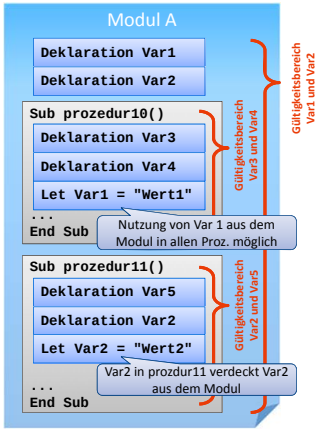
```
' Beispiele

' Zugriff auf Variable/Feld in anderem Modul
Debug.Print mdlKunden.intLetzteKundeNr
Let kndKunde42 = mdlKunde.kndKundenliste(42)

' Funktions- und Prozeduraufruf in anderem Modul
Let kndKunde42 = mdlKunden.gibKunde(42)
Call mdlProdukte.zeigeAlleProdukte
```


Abschluss

BHT

Gültigkeitsbereiche

– innerhalb der Bereiche sind Variablen/Konstanten deklariert und verwendbar
– Variablen/Konstanten übergeordneter Gültigkeitsbereiche in untergeordneten Gültigkeitsbereichen verwendbar
– Sonderfall des "Verdeckens" durch eine Variable mit gleichem Bezeichner im einem untergeordnetem Gültigkeitsbereich



Abschluss

BHT

Sichtbarkeit

– Elemente eines Moduls ein in anderen Modulen sichtbar, wenn das Element als **`Public`** deklariert wurde
– Elemente sind nur innerhalb ihres Moduls sichtbar, wenn das Element als **`Private`** deklariert wurde

Geheimnisprinzip

– dient dem Verbergen der internen Realisierung von Funktionen/Prozeduren und Modulen
– durch Einschränkungen der Sichtbarkeit und eine definierte Schnittstelle nach außen

Abschluss

BHT

Syntax: Schlüsselwort Private oder Public in Verbindung mit

– Deklaration von Variablen auf Modulebene (anstelle von Dim)

```
Private | Public <Variable> As <Datentyp>
```

– Deklaration von Konstanten auf Modulebene

```
Private | Public Const <Konstante> As <DTyp> = <WertAusd>
```

– Zusammengesetzten Datentypen

```
Private | Public Type <Typbezeichner>
 <Eigenschaft> As <Datentyp>
End Type
```

– Prozeduren und Funktionen

```
Private | Public Sub <BezProzedur>(<Param> As <DTyp>)
Private | Public Function <BezFnkt>(<Param> As <DTyp>) As<DTyp>
```

BHT Berliner Hochschule für Technik

Wirtschaftsinformatik 1
LE 07 – Prozeduren, Funktionen und Module

Prof. Dr. Thomas Off
http://www.ThomasOff.de/lehre/beuth/wi1

Einordnung

BHT

06 – Zusammenfassung

05 – Debugger und Testen

04 – Fortgeschrittene Konzepte

- 04.A Oberflächen (Teil 1) Elemente und Eigenschaften
- 04.B Oberflächen (Teil 2) Ereignisverarbeitung
- 04.C Zugriff auf Dateisystem und Anwendungen

03 – Grundkonzepte

- 03.A Wert Ausdruck Variable Konstante Datentyp
- 03.B Bedingte Ausführung/ Verzwei-gungen
- 03.C Schleifen
- 03.D Felder Mengen
- 03.E Prozedur Funktion Modul

02 – Grundlagen der Programmierung

01 – Grundbegriffe der Wirtschaftsinformatik

LE 07 - Prozeduren, Funktionen und Module 156