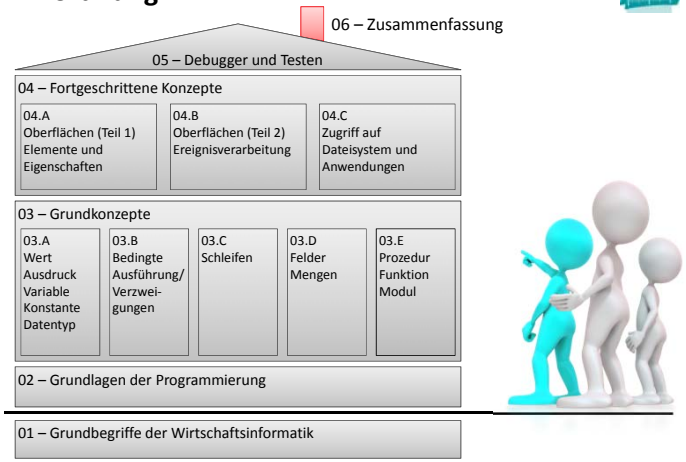
BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Wirtschaftsinformatik 1
LE 11 – Zusammenfassung

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

Einordnung



Inhalt

Einstieg
- LE 01: Wirtschaftsinformatik
- LE 02: Programmierung

Grundkonzepte
- LE 03: Variablen & Datentypen
- LE 04: Verzweigungen
- LE 05: Schleifen
- LE 06: Mengen & Felder
- LE 07: Prozeduren & Funktionen

Fortgeschrittene Konzepte
- LE 08: Ereignisse & GUI
- LE 09: Dateisystem
- LE 10: Fehler, Debugger & Testen

Abschluss

LE 01 – Einführung in die Wirtschaftsinformatik

Wirtschaftsinformatik
- als interdisziplinäre, anwendungsorientierte und gestaltungsorientierte Wissenschaft, deren Erkenntnisgegenstand soziotechnische Systeme sind
- umfasst auch Konzeption, Entwicklung, Einführung, Wartung und Nutzung der computergestützten Verarbeitung von Informationen für betriebswirtschaftliche Aufgaben in Wirtschaftsunternehmen und unternehmensübergreifenden Netzen
- sowie zunehmend deren Management und Innovation

LE 01 – Einführung in die Wirtschaftsinformatik

Grundbegriffe

- Information, Daten, Wissen
- Computer, EVA-Prinzip und Hardware
- Software

LE 01 – Einführung in die Wirtschaftsinformatik

Grundbegriffe

- Information, Daten, Wissen

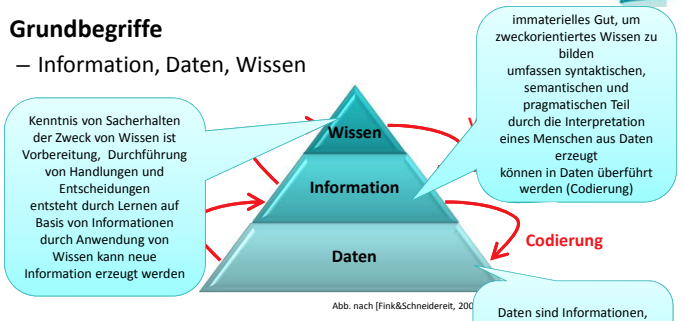


Abb. nach [Fink&Schneidereit, 200

- Computer, EVA-Prinzip und Hardware
- Software

LE 01 – Einführung in die Wirtschaftsinformatik

Grundbegriffe

- Information, Daten, Wissen
- Computer, EVA-Prinzip und Hardware
  - Universell einsetzbare, programmgesteuerte Maschine zur Speicherung und Verarbeitung von Daten
  - Erklärung anhand von Eingabe (E), Verarbeitung (V) und Ausgabe(A) mit entsprechenden Hardwarekomponenten (physische Teile)
  - Wichtige Komponenten
    - CPU zur Verarbeitung, d.h. zur Steuerung der abzuarbeitenden Verarbeitungsvorschriften und zur Ausführung von Rechenoperationen
    - Arbeitsspeicher für Befehle und Daten
  - Von-Neumann-Rechner als Grundlegendes Architekturprinzip
- Software

LE 01 – Einführung in die Wirtschaftsinformatik

Grundbegriffe

- Information, Daten, Wissen
- Computer, EVA-Prinzip und Hardware
- Software
  - Menge von Computer-Programmen mit den zugehörigen Daten und den begleitenden Dokumenten, die für ihre Anwendung notwendig oder hilfreich sind.[1]
  - als allgemeiner Begriff
    - für Software-System (fokussiert innere Struktur der Software) oder
    - Software-Produkt (fokussiert die Käufer- bzw. Auftraggebersicht auf die Software) verwendet[2]
  - …

1) vgl. [Fink et al., 2001], S. 30
2) vgl. [Hesse et al., 1984], S. 22

LE 01 – Einführung in die Wirtschaftsinformatik

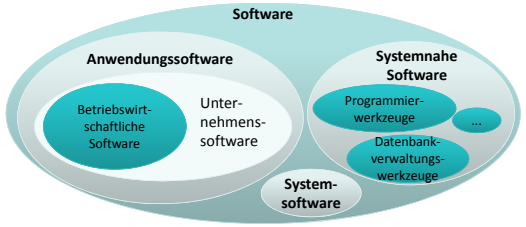
Grundbegriffe

- Information, Daten, Wissen
- Computer, EVA-Prinzip und Hardware
- Software
  - ...
  - Programm
    - enthält die präzisen und vollständigen Arbeitsanweisungen, die ein Computer benötigt, um Daten zu verarbeiten. (Details siehe LE02)
  - Dokumentation
    - alle Schriftstücke, die die Software selbst sowie
    - die Bedienung, den Betrieb, die Wartung und Weiterentwicklung beschreiben
    - Beispiele: Installationsanleitung, Benutzerhandbuch, Administrationshandbuch, Entwicklerdokumentation.

LE 01 – Einführung in die Wirtschaftsinformatik

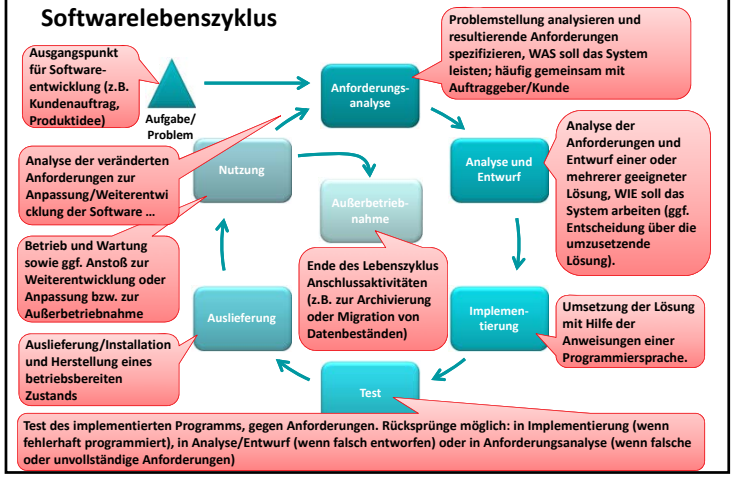
Grundbegriffe

- Information, Daten, Wissen
- Computer, EVA-Prinzip und Hardware
- Software



LE 01 – Einführung in die Wirtschaftsinformatik

Softwarelebenszyklus



LE 02 – Einführung in die Programmierung

Algorithmus

- Definition: präzise, vollständige, eindeutig formulierte, endliche Verarbeitungsvorschrift, die Ausgangssituation in ein Ergebnis überführt, das zur Lösung einer Aufgabe dienen soll.
- Bestandteile: Anweisungen, Ablauf, Verzweigungen, Schleifen, Unterprogramme
- Beschreibung: Struktogramme, Programmablaufpläne, UML-Aktivitätsdiagramme, ...

Datenelement und -struktur

- einfache Datenelemente, zum Schreiben und Lesen eines Werts;
- komplexe Datenelemente, die aus einfachen aufgebaut sind und
- komplexe Datenstrukturen, die Datenelemente in bestimmter Form organisieren und außer schreibendem und lesendem Zugriff spezielle Aktionsmöglichkeiten bieten (z.B. Einfügen, Entfernen).
- Beispiele: Verkettete Liste, Stapel, Schlange, Baum

LE 02 – Einführung in die Programmierung

Programm

- mit den Sprachmitteln einer konkreten Programmiersprache ausgedrückter Algorithmus in Verbindung mit den ebenso ausgedrückten Datenstrukturen zur Ausführung in einem Computer
- kann vorliegen als
  - Quellcode: Darstellung in einer lesbaren und verständlichen Programmiersprache
  - Maschinencode: Darstellung mit Befehlen aus dem Befehlsvorrats des konkret verwendeten Computers

Programmiersprache: Formale Sprache zur Formulierung von Programmen mit präziser Syntax und eindeutiger Semantik

Maschinensprache: Binäre und ausführbare Darstellung des Programms, abhängig von der verwendeten Hardware

LE 02 – Einführung in die Programmierung

Compiler

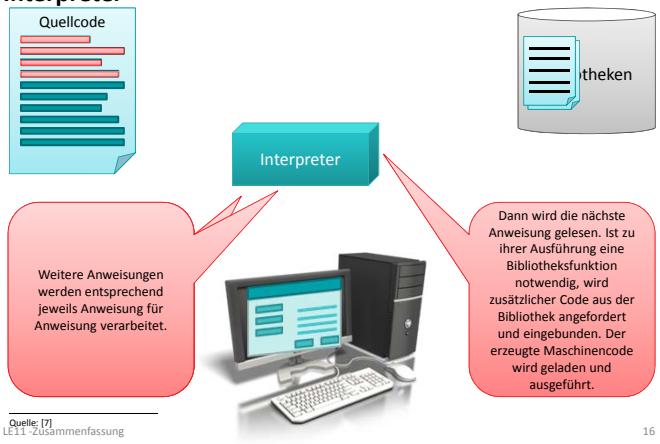
- fertiger Programmcode wird vollständig dem Compiler übergeben und
- über ein Zwischenformat in ausführbares Programm übersetzt, das geladen und ausgeführt werden kann

Interpreter

- liest eine Anweisung aus dem Quellcode und übersetzt sie in Maschinencode, lädt die Anweisung und führt sie aus
- dann wird mit der nächsten Anweisung fortgefahren

LE 02 – Einführung in die Programmierung

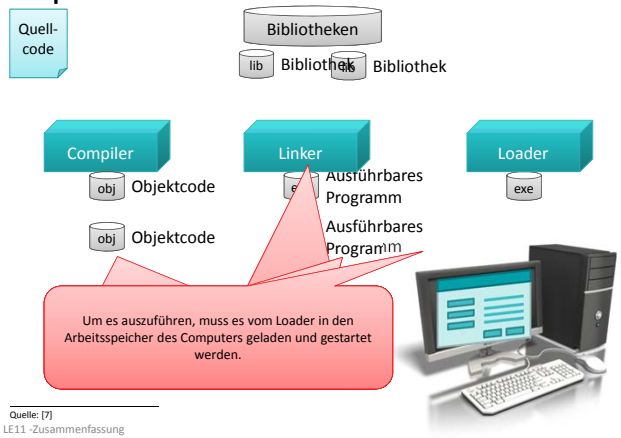
Interpreter



Quelle: [7]

LE 02 – Einführung in die Programmierung

Compiler



Quelle: [7]

LE 02 – Einführung in die Programmierung

Entwicklungsumgebung

- stellt ein oder mehrere Werkzeuge zur Programmentwicklung zur Verfügung (z.B. zur Erfassung von Quellcode, den Compiler oder Interpreter, Debugger, Werkzeuge für die Gestaltung von Benutzeroberflächen)
- hier Verwendung von Microsoft Access mit der integrierten Programmiersprache Visual Basic for Applications (VBA)

MS Access/VBA als Entwicklungsumgebung

- VBA Editor in MS Access Datenbank verwenden
- In Modul werden unsere Anweisungen (zunächst) als Prozedur zwischen Sub und End Sub geschrieben
- Ausführung des Programms
  - Eingabemarkierung muss innerhalb der Prozedur positioniert sein
  - Per Menüeintrag Ausführen>Sub/User Form ausführen, per "Play"-Icon ▶ oder Funktionstaste F5

Variable

- wird deklariert mit Schlüsselwort Dim
- hat einen **Bezeichner**
- und ist von einem definierten **Datentyp**
- **Werte** oder **Ausdrücke** werden ihr **zugewiesen**
- erstmalige Zuweisung heißt **Initialisierung**
- bietet **Zugriff** auf gespeicherten Wert
  - lesend
  - schreibend/ändernd

Syntax

```
Dim <VarBezeich> As <Datentyp>

Let <VarBezeich> = <Wert/Ausdr>
```

Beispiele

```
' Weniger gute Bezeichner
Dim i As Integer
Dim s As String
' Aussagekräftige Bezeichner
Dim bytAlter As Byte
Dim sglBetrag As Single
' Initialisierung mit Wert
Let bytAlter = 20
' Initialisierung mit Ausdruck
Let sglBetrag = bytAlter * 3
' Lesender Zugriff
Debug.Print bytAlter
' Ändernder Zugriff
Let bytAlter = bytAlter + 1
Let sglBetrag = 42
```


LE 03 – Variable, Konstante, Datentyp, …

Variable	Schubladen in Möbeln
– speichert Werte eines definierten Datentyps	– bieten Platz für Gegenstände eines bestimmten Typs
– Werte oder Ausdrücke werden ihr zugewiesen	– Gegenstände werden in ihnen abgelegt
– bietet • lesenden Zugriff und • schreibenden/ändernden Zugriff auf gespeicherten Wert	– es kann • nachgesehen werden was vorhanden ist • vorhandener Gegenstand durch anderen ersetzt werden
– hat einen Bezeichner (Namen)	– haben nur selten eine Bezeichnung

LE 03 – Variable, Konstante, Datentyp, …

Schubladen in Möbeln

- bieten Platz für Gegenstände eines bestimmten Typs
- Gegenstände werden in ihnen abgelegt
- es kann
  - nachgesehen werden was vorhanden ist
  - vorhandener Gegenstand durch anderen ersetzt werden
- haben nur selten eine Bezeichnung

Konstante

ist der Variable ähnlich

- hat einen Bezeichner
- speichert Werte eines definierten Datentyps
- Werte oder Ausdrücke werden ihr zugewiesen
- bietet lesenden Zugriff auf gespeicherten Wert

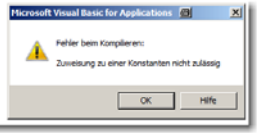
unterscheidet sich von der Variable

- Schlüsselwort zur Deklaration ist **`Const`**
- es folgen: Bezeichner, Datentyp und Zuweisung des Wertes
- Bezeichner per Konvention in Großbuchstaben
- zugewiesener Wert kann während der Programmausführung nicht verändert werden

Konstante

Veränderung des einmal zugewiesenen Wertes nicht mehr möglich

```
Sub Raumzeit()
 Const PI As Double = 3.14159265359

 ' Veränderung im Raum-Zeit-Kontinuum oder
 ' Versuch einer versehentlichen Änderung
 PI = 4.23
End Sub
```



Konstante

Einsatzmöglichkeiten

- gute Bezeichner für Konstanten unterstützen die Verwendung vordefinierter Werte
- Konstante in Verbindung mit unveränderbaren Werten, z.B.

```
Const PI As Double = 3.14159265359
```

- Konstante als symbolischer Namen
  - für selbst definierte Werte, die an nur an einer Stelle festgelegt und an mehreren Stellen verwendet werden sollen

```
Const MWST As Single = 0.19
```

  - für vordefinierte Werte, die in Verbindung mit Anweisungen eine besondere Bedeutung haben (nächste Folie)

LE 03 – Variable, Konstante, Datentyp, …

Konstante

- ist der Variable ähnlich, unterscheidet sich aber
  - Schlüsselwort zur Deklaration ist **`Const`**
  - zugewiesener Wert kann während der Programmausführung nicht verändert werden
- hat kaum etwas mit IKEA-Möbeln zu tun.

LE 03 – Variable, Konstante, Datentyp, …

VBA-Datentypen für natürliche/ganze Zahlen

Typ	Speicherbedarf	Kleinster Wert	Größter Wert
Byte	1 Byte	0	255
Integer	2 Byte	-32.768	32.767
Long	4 Byte	-2.147.483.648	2.147.483.647
LongLong	8 Byte	-9.223.372.036.854.775.808	9.223.372.036.854.775.807

Beispielcode

```
Dim ganzeZahl As Integer
Dim auchGanzeZahl As Long
```

Operatoren

Operator	Benennung	Beispiel	Art
+	Addition	5+6 ergibt 11	binär
-	Subtraktion	9-3 ergibt 6	binär
*	Multiplikation	9*8 ergibt 72	binär

LE 03 – Variable, Konstante, Datentyp, …

Operatoren (Fortsetzung)

Operator	Benennung	Beispiel	Art
/	Division	3/4 ergibt 0,75	binär
\	Ganzzahlen-Division	7\2 ergibt 3	binär
Mod	Restwert (Modulo)	7 Mod 2 ergibt 1	binär
-	Negation	-3 ergibt -3	unär
^	Potenz	2^2 ergibt 4 3^3^3 ergibt 19.683	binär

Vergleichsoperator	Benennung	Beispiel
<	Kleiner	3<5 ergibt True
<=	Kleiner gleich	3<=5 ergibt True
>	Größer	2>9 ergibt False
>=	Größer gleich	5>=3 ergibt True
=	Gleich	3=3 ergibt True
<>	Ungleich	3<>3 ergibt False

LE 03 – Variable, Konstante, Datentyp, …

VBA-Datentypen für Gleitkommazahlen

Typ	Speicherbedarf	Negative Werte	Positive Werte
Single	4 Bytes	-3,402823E38 bis -1,401298E-45	1,401298E-45 bis 3,402823E38
Double	8 Bytes	-1,79769313486231E308 bis -4,94065645841247E-324	4,94065645841247E-324 bis 1,79769313486232E308

Beispielcode

```
Dim gleitkommaZahl As Single
Dim genauereGleitkommaZahl As Double
```

Operatoren

– Wie ganze Zahlen

LE 03 – Variable, Konstante, Datentyp, …

VBA-Datentypen für Festkommazahlen

Typ	Speicher-bedarf	Wertebereich
Currency	8 Bytes	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807
Decimal	14 Bytes	+/-79.228.162.514.264.337.593.543.950.335 ohne Dezimalzeichen; +/-7,9228162514264337593543950335 mit 28 Nachkommastellen; die kleinste Zahl ungleich Null ist +/-0,0000000000000000000000000001.

Beispielcode

```
Dim waehrungsBetrag As Currency
' Deklaration von Decimal so nicht möglich
```

Operatoren

– Wie ganze Zahlen

LE 03 – Variable, Konstante, Datentyp, …

VBA-Datentyp für Wahrheitswerte

Typ	Speicherbedarf	Wertemenge
Boolean	2 Bytes	True oder False

Beispielcode

```
Dim ergebnisOk As Boolean
```

Wertemenge

Wert	Bedeutung
true	Wahr, Ja
false	Falsch, Nein

Hinweise

- Boolean repräsentiert eine Menge von Werten und keinen Bereich, denn es gibt "zwischen" true und false keine weiteren Werteausprägungen.
- Wir verwenden ausschließlich True und False und **nicht** deren Abbildung auf Integer-Werte (-1 für True bzw. 0 für False)

LE 03 – Variable, Konstante, Datentyp, …

Operatoren

Operator	Bezeichnung	Beispiel	Art
=	Gleich	True = False ergibt False	binär
<>	Ungleich	True <> False ergibt True	binär
And	Logisches Und	True And True ergibt True True And False ergibt False	binär
Or	Logisches Oder	True Or True ergibt True True Or False ergibt True False Or Fakse ergibt False	binär
Xor	Exklusives Oder	False Xor True ergibt True True Xor False ergibt True True Xor True ergibt False False Xor False ergibt False	binär
Not	Negation	Not True ergibt False Not False ergibt True	unär

LE 03 – Variable, Konstante, Datentyp, …

VBA-Datentyp für Datum und Zeit

Typ	Speicherbedarf	Wertebereich
Date	8 Bytes	Datum vom 01. Januar 0100 bis zum 31. Dezember 9999 Uhrzeit von 0:00:00 bis 23:59:59

Beispielcode

```
Dim gebDat As Date
' Im US-Datumsformat zuweisen mit #
Let gebDat = #10/13/1985# '13. Oktober 1985
' oder über den "Umweg" eines Strings
Let gebDat = "13.10.1985" 'auch 13. Oktober 1985
```

Operatoren

- Inkrement und Dekrement, d.h. tageweises Hochzählen bzw. Herunterzählen ("Addition und Subtraktion von Tagen")
- Vergleichsoperatoren

LE 03 – Variable, Konstante, Datentyp, …

Hilfsfunktionen für Datum/Zeit (von VBA bereitgestellt)

Funktion	Par.-Typ	Aufgabe	Beispiel
Month	Date	Monat erfragen	`Month(gebDat)` ergibt **10**
Day	Date	Tag erfragen	`Day(gebDat)` ergibt **13**
Year	Date	Jahr erfragen	`Year(gebDat)` ergibt **1985**
Weekday	Date	Wochentag erfragen	`Weekday(gebDat)` ergibt **1** [für Sonntag]
TimeValue	Date	Tageszeit erfragen	`TimeValue(gebDat)` ergibt **17:05:33**
Now	-	Tagesdatum erfragen	`Now()` ergibt **10.10.2011 12:35:36**

LE 03 – Variable, Konstante, Datentyp, …

VBA-Datentyp für Zeichenketten

Typ	Speicherbedarf	Wertebereich
String	1 Byte je Zeichen (und bei variabler Länge zzgl. 10 Bytes)	< 2,1 Mrd. Zeichen bei variabler Länge < ca. 65.400 Zeichen bei fester Länge

Beispielcode

```
Dim zeichenKette As String
```

Operatoren

Operator	Benennung	Beispiel	Art
&	Verkettung	"Com" & "puter" ergibt "Computer"	binär

LE 03 – Variable, Konstante, Datentyp, …

Hilfsfunktionen (von VBA bereitgestellt)

– für einzelne Zeichen

Funktion	Par.-Typ	Aufgabe	Beispiel
Chr	Byte	Umwandlung des Zeichencodes in ein Zeichen	Chr(77) ergibt "M"
Asc	String	Ermittlung des Zeichencodes (des ersten Zeichens eines Strings)	Asc("a") ergibt 97 Asc("abc") ergibt 97

– für Zeichenketten

Funktion	Par.-Typ	Aufgabe	Beispiel
Len	String	Ermittlung Stringlänge	Len("Hallo Welt") ergibt 10
Left, Right	String, Byte	Ermittlung linker bzw. rechter Teil	Left("Hallo", 2) ergibt "Ha" Right("Welt", 2) ergibt "lt"
Mid	String, Von als Byte, Länge als Byte	Ermittlung mittlerer Teil	Mid("Hallo Du", 4, 2) ergibt "lo"

LE 03 – Variable, Konstante, Datentyp, …

Hilfsfunktionen (von VBA bereitgestellt)

– für Zeichenketten (Fortsetzung)

Funktion	Par.-Typ	Aufgabe	Beispiel
Ltrim	String	Führende Leerzeichen entfernen	Ltrim(" Welt ") ergibt "Welt "
Rtrim	String	Endende Leerzeichen entfernen	Rtrim(" Welt ") ergibt " Welt"
Trim	String	Führende und endende Leerzeichen entfernen	Trim(" Welt ") ergibt "Welt"
Instr	Basis als String, Teil als String	Position von Teil in Basis	Instr("Hallo Welt", "lo") ergibt 4
Val	String	Liefert den numerischen Wert der Zeichenkette	Val("42") ergibt 42 Val("0815") ergibt 815 Val("0.07") ergibt 0,07 Val("") ergibt 0 Val("Hallo") ergibt 0

Darstellung von Werten und Literalen

Datentyp bestimmt über die Darstellung der Werte, d.h.

– Ganze Zahlen (ohne Anführungszeichen)

```
Let bytAlter = 23
```

– Gebrochene Zahlen mit Punkt anstelle des Kommas und ohne Anführungszeichen!

```
Let cruPreis = 24.99
```

– Texte als String in Anführungszeichen

```
Let strName = "Müller"
```

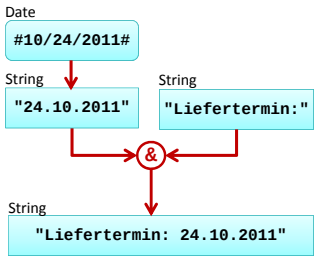
– Datum in "amerikanischer" Schreibeweise mit Doppelkreuz

```
Let datGebDat = #12/24/1998# ' Weihnachten geboren
```

Typumwandlungen

Implizite Umwandlung

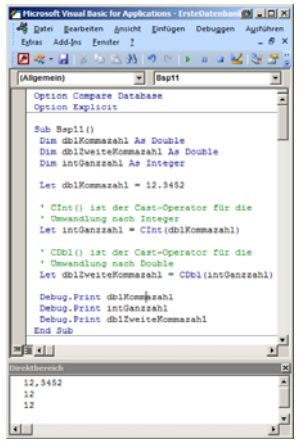
- häufige Problemstellung bei Auswertung von Ausdrücken und bei Zuweisungen
- Werte von Variablen mit unterschiedlichen Datentypen in Ausdruck
- Implizite Typumwandlung geht ohne Informationsverlust einher, z.B.
  - Alle Datentypen nach String
  - Byte nach Integer, Long, Decimal
  - Currency nach Decimal



Typumwandlungen

Explizite Umwandlung

- können mit (gewolltem) Informationsverlust einhergehen, z.B.
  - Double nach Integer unter Verlust der Kommastellen
- Verwendung von Cast-Operatoren, um Umwandlung zu steuern
  - **`CBool, CInt, CLng, CStr, CDate, CDbl`**
  - bei **`CInt`** und **`CLng`** in Verbindung mit Rundung
  - Details auf nächster Folie



Typumwandlungsfunktionen[1]

Funktion	Rückgabetyp	Bereich des Arguments *Ausdruck*
CBool	Boolean	Eine gültige Zeichenfolge oder ein gültiger numerischer Ausdruck.
CByte	Byte	0 bis 255.
CCur	Currency	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807.
CDate	Date	Ein beliebiger gültiger Datumsausdruck.
CDbl	Double	-1,79769313486231E308 bis -4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte.
CInt	Integer	-32.768 bis 32.767; Nachkommastellen werden gerundet.
CLng	Long	-2.147.483.648 bis 2.147.483.647; Nachkommastellen werden gerundet.
CLngLng	LongLong	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807; Dezimalen werden gerundet (nur auf 64-Bit-Plattformen zulässig)
CSng	Single	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte.
Cvar	Variant	Numerische Werte im Bereich des Typs **Double**. Nichtnumerische Werte im Bereich des Typs **String**.
CStr	String	Rückgabe für CStr hängt vom Argument *Ausdruck* ab.

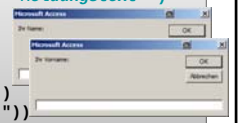
1) Quelle: [3]

LE 04 – Ein-/Ausgabe und Verzweigungen

Eingabe und Ausgabe

- Eingabe mittels Dialog

```
' Generelle Syntax:
' Let <Variable-vom-Typ-String> = InputBox("<Meldungstext>")
' Beispiele
Let strName = InputBox("Ihr Name:")
Let strVorname = InputBox("Ihr Vorname:")
Let bytAlter = Val(InputBox("Ihr Alter:"))
Let bytAlter = CByte(InputBox("Ihr Alter:"))
Let datGebDat = DateValue(InputBox("GebDat:"))
```

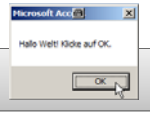


- Ausgabe im Debug-Bereich

```
' Beispiel
Debug.Print "Hallo Welt!"
```

- Ausgabe im Meldungsfenster

```
' Generelle Syntax: MsgBox ("<Meldungstext>")
' Beispiel:
MsgBox ("Hallo Welt! Klicke auf OK.")
```



LE 04 – Ein-/Ausgabe und Verzweigungen

Ausgangspunkt

- Beschränkung bisheriger Programme auf linearen Programmablauf
- Notwendigkeit zu Verzweigungen

Konzepte

- Arten von Verzweigungen, z.B.

Bedingung ist wahr: Ja	Bedingung ist wahr: Nein
Anweisung 4/ Anweisungsblock 4	Anw 5/ Anwb 5

Variable A: Wert 1	Wert 2	Wert 3	Sonst
Anw4/ Anwb4	Anw5/ Anwb5	Anw6/ Anwb6	A7/ Ab7

- Formulierung von Bedingungen für Verzweigungen
  - als Ausdrücke, in der Regel mit Vergleichsoperator
  - mit Wahrheitswert als Ergebnis der Auswertung
  - Einsatz logischer Operationen

LE 04 – Ein-/Ausgabe und Verzweigungen

Verzweigungen mit VBA

- Einfach Verzweigungen in Form von If-Then-Else-End If
- Mehrfach Verzweigungen
  - ElseIf-Erweiterung
  - Select-Case-Anweisung

```
' Beispiel
If intZahl > 4 Then
 Debug.Print "Größer 4"
Else
 Debug.Print "Kleiner 5"
End If
```

```
' Beispiel
Select Case bytGroesse
Case 36
 Debug.Print "S"
Case 38
 Debug.Print "M"
Case 40
 Debug.Print "L"
Case Else
 Debug.Print "XL"
End Select
```

```
' Beispiel
If intZahl > 4 Then
 Debug.Print "Größer 4"
ElseIf intZahl = 4 Then
 Debug.Print "Gleich 4"
Else
 Debug.Print "Kleiner 4"
End If
```


LE 05 – Schleifen

Vorprüfende/Kopfprüfende Schleife

- Einsatzzweck
  - erst Bedingung prüfen
  - dann ggf. Anweisung ausführen
  - anschließend Wiederholung der Prüfung usw.

```
Wdhl. solange Bedingung
   Anweisung
```

- Generelle Syntax

```
Do While <Bedingung>
<Anweisung(en)>
Loop
```

Solange wie

```
Do Until <Bedingung>
<Anweisung(en)>
Loop
```

Solange bis

- Beispiel

```
Dim i As Integer
Let i = 0
Do While i < 5
 Let i = i + 1
 Debug.Print i
Loop
```

```
Dim j As Byte
Let j = 0
Do Until j  > 4
 Let j = j + 1
 Debug.Print j
Loop
```


LE 05 – Schleifen

Nachprüfende/Fußprüfende Schleife

- Einsatzzweck
  - erst Anweisung ausgeführt
  - dann Bedingung prüfen
  - anschließend Wiederholung der Anweisungsausführung usw.

```
   Anweisung
Wdhl. solange Bedingung
```

- Generelle Syntax

```
Do
 <Anweisung(en)>
Loop While <Beding.>
```

Solange wie

```
Do
 <Anweisung(en)>
Loop Until <Beding.>
```

Solange bis

- Beispiel

```
Dim i As Integer
Let i = 0
Do
 Let i = i + 1
 Debug.Print i
Loop While i < 5
```

```
Dim j As Byte
Let j = 0
Do
 Let j = j + 1
 Debug.Print j
Loop Until j > 4
```

LE 05 – Schleifen

Zählerschleifen

- Einsatz
  - Vorher bekannte Anzahl von Wiederholungen
  - Anzahl gesteuert über Start und Ende
  - Ausführung der Anweisung solange Anzahl Wiederholungen das Ende noch überschritten hat

```
Zähle von Start bis Ende
    Anweisung
```

- Generelle Syntax

```
For <Var> = <Wert/Ausdr> To <Ausdr> Step <Schrittw>
 <Anweisung(en)>
Next
```

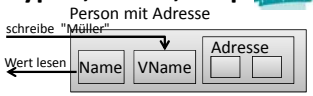
- Beispiel

```
Dim i As Integer
For i = 1 To 10 Step 2
 Debug.Print i
Next
```


LE 06 – Zusammenges. Datentypen, Felder, Map

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
```

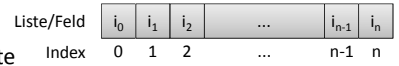
```
' Definition
Type TPerson
 strName As String
 adrWohnanschrift As TAdresse
End Type
```

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


LE 06 – Zusammenges. Datentypen, Felder, Map

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



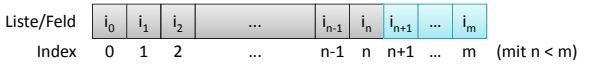

```
' Generelle Syntax
Dim <Bez>(<n>) As <DTyp>

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)
```


LE 06 – Zusammenges. Datentypen, Felder, Map

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>)
```

LE 06 – Zusammenges. Datentypen, Felder, Map

Mehrdimensionale Felder

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

	Index	0	1	2	...	n-1	n
Mehrdimensionales Feld	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>(<m>, <n>, ...) As <Datentyp>
```


LE 06 – Zusammenges. Datentypen, Felder, Map

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
```

```
Dim colKnd As Collection
Set colKnd = New Collection
```

```
colKnd.Add "Müller", "K1"
colKnd.Add "Yilmaz", "K4"
colKnd.Add "Meier", "K2"

Debug.Print colKnd.Item("K4")

colKnd.Remove("K2")
```

– Generelle Syntax für Zugriffe

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


Zusammengesetzte Datentypen: Beispiel 06.01

Ziel

– Definition und Nutzung eines zusammengesetzten Datentypen

Aufgabe

– Definieren Sie einen Datentyp für Kunden mit
  - Kundenummer
  - Name, Vorname
  - Straße und HausNr.
  - Plz und Ort
– Nutzen Sie den Datentyps zur Deklaration von drei Variablen des Typs Kunde
– Initialisieren Sie die Variablen mit Werten
– Geben Sie die Variablen im Direktbereich aus

Zusammengesetzte Datentypen: Beispiel 06.01

Ziel

– Nutzung eines Feldes von zusammengesetzten Datenelementen

Aufgabe

– Deklarieren Sie ein Feld für Kunden (aus der vorherigen Übung)
– Initialisieren Sie das Feld mit drei verschiedenen Kunden
– Geben Sie alle Kunden im Direktbereich aus, mittels
  - einer Zählerschleife
  - einer vorprüfenden Schleife
  - einer nachprüfenden Schleife
– Was passiert bei der nachprüfenden Schleife, wenn das Feld keine Element hat?

Zusammengesetzte Datentypen: Beispiel 06.01

Ziel

– Definition und Nutzung mehrdimensionalen Feldes

Aufgabe

– Deklarieren Sie ein mehrdimensionales Feld für einen Kinosaal (Reihen und Sitze)
– Belegen Sie einige Sitze im Kino
– Geben Sie das mehrdimensionale Feld vollständig im Direktbereich aus

LE 07 – Prozeduren, Funktionen, Module, …

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

LE 07 – Prozeduren, Funktionen, Module, …

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**

LE 07 – Prozeduren, Funktionen, Module, …

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
```

LE 07 – Prozeduren, Funktionen, Module, …

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)

LE 07 – Prozeduren, Funktionen, Module, …

Syntax für den Zugriff auf Modulbestandteile

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

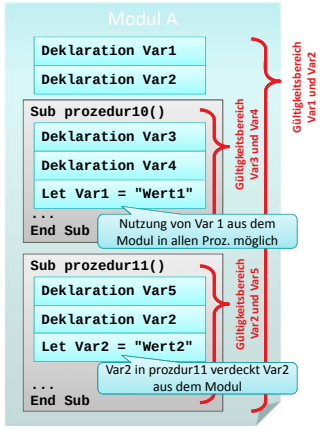
```
' Generelle Syntax
<BezeichnerAnderesModul>.<BezeichnerDesModulbestandteils>
```

```
' 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
```


LE 07 – Prozeduren, Funktionen, Module, …

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



LE 07 – Prozeduren, Funktionen, Module, …

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

LE 07 – Prozeduren, Funktionen, Module, …

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>
```


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 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
```


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

Parameter: Beispiel 07.04

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

Ziel

– Verschiedene Möglichkeiten zur Parameterübergabe nutzen

Aufgabe

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

Module: Beispiel 07.09

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

LE 08 – Oberflächen und Ereignisverarbeitung

Wichtige Oberflächenelemente

– Eingabe und Auswahl
  • Textfelder
  • Aufklappliste/Kombinationsfeld
  • Mehrfachauswahllisten
  • Radioknöpfe (Optionsfeld)
  • Kontrollkästchen (Checkbox)
– Aktionselemente
  • Schaltfläche (einfach)
  • Umschaltfläche (Troggle)
– Container
  • Rahmen/Gruppen
  • Registerkartensatz mit Registerkarten
  • Fenster/Dialoge (in Access als Formulare)

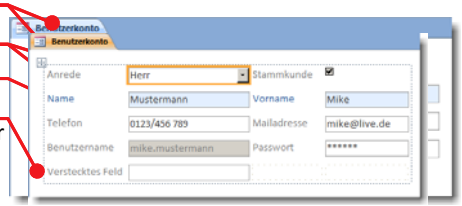
LE 08 – Oberflächen und Ereignisverarbeitung

Oberflächenelemente haben Eigenschaften,

- die Ihr Aussehen festlegen
- die Interaktionsmöglichkeiten des Benutzers beeinflussen
- Auswirkung auf die Programmierung haben (siehe Teil 2)
- ...

Beispielsweise haben die meisten Oberflächenelemente

- Beschriftung
- Farben
- Änderbarkeit
- Sichtbarkeit
- Name (im Formular nicht sichtbar)



LE 08 – Oberflächen und Ereignisverarbeitung

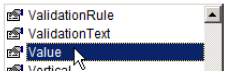
Zugriff aus VBA auf Oberflächenelemente über Namen

- Nutzung von Me (Referenz auf aktuelles Fenster)

```
Me.tx
```

- gefolgt vom Namen des Feldes "txtBezeichungsfeld1"

```
Me.txtBeispielfeld1.V
```



- gefolgt von der Eigenschaft, auf die Zugegriffen werden soll (hier Value)

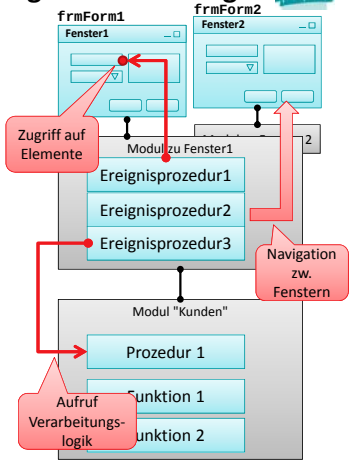
```
Me.txtBeispielfeld1.Value = "Hallo Welt!"
```

- jeweils getrennt durch Punkt "." (Punkt-Notation)

LE 08 – Oberflächen und Ereignisverarbeitung

Ereignisprozeduren bieten Zugriffsmöglichkeit auf die Oberflächenelemente und dienen zum

- Steuern der Elemente auf der Oberfläche
  - Aktivieren/Dekativieren bzw. Einblenden/Ausblenden von Elementen
  - Navigation zwischen Fenstern
  - ...
- Aufruf der Verarbeitungslogik
  - Übergabe der eingegebenen Daten zur Verarbeitung
  - Ermitteln der anzuzeigenden Daten
  - Ausführen von komplexen Berechnungen
  - ...



LE 08 – Oberflächen und Ereignisverarbeitung

Referenzvariable Me stellt Funktionen zur Verfügung

- Zugriff auf den Wert von Feldern liefert immer String

```
' Generelle Syntax
Let <VarString> = Me.<BezeichnerDesFeldes>.Value
Let <VarZahl> = Val(Me.<BezeichnerDesFeldes>.Value)
```

- Genereller Zugriff auf Eigenschaften von Elementen

```
' Generelle Syntax
Let <Var> = Me.<Bez>.<Eigenschaft> ' Lesen
Let Me.<Bez>.<Eigenschaft> = <Var> ' Schreiben/Ändern
```

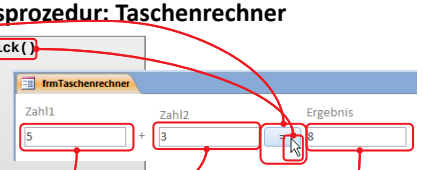
Beispiele

```
' Lesen von Werten
Let strName = Me.txtName.Value
Let intAlter = Val(Me.txtAlter.Value)
Let bolBrillentraeger = CBool(Me.chkBrille.Value)
' Ändern von Eigenschaften
Let Me.txtName.Visible = False
```

LE 08 – Oberflächen und Ereignisverarbeitung

Beispiel einer Ereignisprozedur: Taschenrechner

```
Private Sub btnBerechne_Click()
 ' Deklaration
 Dim strZahl1 As String
 Dim strZahl2 As String
 Dim dblZahl1 As Double
 Dim dblZahl2 As Double
 Dim dblErgebnis As Double

 ' Text aus Textfeldern auslesen
 Let strZahl1 = Me.txtZahl1.Value
 Let strZahl2 = Me.txtZahl2.Value

 ' Umwandeln in Zahlen
 Let dblZahl1 = Val(strZahl1)
 Let dblZahl2 = Val(strZahl2)

 ' Addition
 Let dblErgebnis = dblZahl1 + dblZahl2

 ' Ergebnis in Textfeld Ergebnis
 Let Me.txtErgebnis.Value = dblErgebnis

End Sub
```



LE 08 – Oberflächen und Ereignisverarbeitung

Navigation zwischen Fenstern mit DoCmd-Befehl

– Generelle Syntax zum Öffnen von Fenstern

```
' Generelle Syntax (Auszug)
DoCmd.OpenForm <Formularname>
```

– Generelle Syntax zum Schließen von Fenstern

```
' Generelle Syntax (Auszug)
DoCmd.Close <TypZuSchließendesObjekt>, <Name>
' Syntax zum Schließen von Formularen
DoCmd.Close acForm, <Formularname>
```

– Generelle Syntax zum Navigieren zwischen Fenstern

```
' Generelle Syntax (Auszug)
DoCmd.BrowseTo <TypZielOjekt>, <Name>
' Syntax zum Schließen von Formularen
DoCmd.BrowseTo acBrowseToForm, <Formularname>
```


Oberflächen und Ereignisse: Beispiel 08.10

Ziel

– Zusammenfassende Übung inkl. Eigenschaften

Aufgabe

– Erstellen Sie einen Begrüßungsdialog, mit einem Text und einer Schaltfläche, über die der Benutzer zum Taschenrechner navigieren kann
– Erstellen Sie die Ihnen bekannte Oberfläche eines Taschenrechners mit
  • drei Textfeldern
  • zwei Schaltflächen ("Berechnen" und "Leeren")
  • einer Aufklappliste für die Auswahl des Operators
– Erstellen Sie ein Modul, in dem alle notwendigen mathematischen Operationen als Funktionen umgesetzt sind
– Implementieren Sie die Ereignisprozedur für den Klick auf die Schaltflächen

LE 09 – Dateisystem und Dateizugriff

Zugriff auf Dateisystem

– mit Modul "FileSystem" grundlegende Möglichkeiten
  • Elemente auflisten

```
' Generelle Syntax mit Angabe des gewünschten Inhalts
' z.B. vbDirectory, vbHidden, vbSystem
Let <strElement> = FileSystem.Dir(<Pfad>, <GewünschteInhalte>)
Let <strElement> = FileSystem.Dir() ' Nächstes (im vorherigen Pfad)
```

  • Weitere: Verzeichnisse anlegen, löschen, ...
– mit FileSystem-Klasse aus MS Scripting Runtime bestehen weitergehend Möglichkeiten z.B.
  • Zugriff auf Laufwerksinformation,
  • Kopieren von Verzeichnissen

LE 09 – Dateisystem und Dateizugriff

Dialoge zur Auswahl von Dateien und Verzeichnissen

- sind sinnvoll, wenn vom Benutzer das Ziel zum Speichern oder Laden von Daten im Dateisystem selbst gewählt werden soll

Generelle Syntax

- Deklaration und Initialisierung

```
Dim <FileDlgObj> As Object
Set <FileDlgObj> = Application.FileDialog(<Zahl>)
```

- Konfiguration (z.B. Mehrfachauswahl)

```
<FileDlgObj>.AllowMultiSelect = True
```

- Anzeige

```
Let <intVar> = <FileDlgObj>.Show() ' Rückgabewert 0 = Abbruch
```

- Ergebnis in Collection "SelectedItems" enthalten

```
<FileDlgObj>.SelectedItems
```


LE 09 – Dateisystem und Dateizugriff

Generelle Syntax (Fortsetzung)

- Ergebnis in Collection "SelectedItems" enthalten

```
<FileDlgObj>.SelectedItems
```

Beispiel für Standarddialog zur Dateiauswahl

```
Dim intResult As Integer ' Rückgabewert
Dim i As Integer ' Schleifenvariable
Dim oFd As Object ' Variable für FileDialog
Set oFd = Application.FileDialog(3) ' Initial. als Dateiauswahl = 3

oFd.AllowMultiSelect = True ' Konfiguration, z.B. Mehrfachauswahl
Let intResult = oFd.Show ' Dialog anzeigen und Ergebnis merken

If intResult = 0 Then
 Exit Sub ' Abbruch durch Benutzer
End If

' Schleife über alle ausgewählten Dateien
For i = 1 To oFd.SelectedItems.Count
 Debug.Print oFd.SelectedItems(i)
Next
```


LE 09 – Dateisystem und Dateizugriff

Früheste Darstellung des Dateizugriffs in Basic

Quelle der Abbildung: [1]

... seither nahezu unverändert

Lizenzbedingung für diese Darstellung: Creative Commons-Lizenz Namensnennung Weitergabe unter gleichen Bedingungen 1.0 US-amerikanisch (nicht portiert)

LE 09 – Dateisystem und Dateizugriff

Zugriff auf Dateien

- Freie Dateinummer ermitteln
- Öffnen einer Datei (verkürzte Form)
  - Pfad: Angabe des Pfades zur Datei
  - Modus: Lesen (Input), Schreiben (Output), Ändern (Append), ...
  - Zugriff: Lesen (Read), Schreiben (Write) oder Ändern (ReadWrite)
  - Dateinummer: Zuvor mit FreeFile() ermittelte Nummer
- Schließen einer Datei

Generelle Syntax

```
Let <intVar> = FileSystem.FreeFile()
```

```
Open <Pfad> For <Modus> Access <Zugriff> As #<DateiNr>
```

```
Close #<DateiNr>
```

Beispiel

```
' Freie Nummer für Dateizugriff
Let intFNr = FileSystem.FreeFile()

' Datei öffnen (zum Schreiben)
Open "C:\Temp\doc3.txt" For Output _
  Access Write As #intFNr

' Datei verwenden
' ...

Close #intFNr  ' Datei schließen
```

LE 09 – Dateisystem und Dateizugriff

Zugriff auf Dateien (Forts.)

– Datei schreiben (Write)

```
Write #<DateiNr>, <WertAusdr> ' Variante 1
Write #<DateiNr>, <WertAusdr1>, <WertAusdr2>, ... ' Variante 2
Write #<DateiNr>, ' Leere Zeile
```

– Datei lesen (Input)

```
Input #<DateiNr>, <Variable> ' Variante 1
Input #<DateiNr>, <Var1>, <Var2>, ... ' Variante 2
```

– Datei zeilenweise lesen (Line Input)

```
Line Input #<DateiNr>, <StringVariable>
```

Beispiel (Schreibzugriff)

```
'...
' Beispiel 1 schreiben (mit Variablen)
Write #intFNr, strName, datGebDat, bolGeschlecht
' Beispiel 2 schreiben (hier auch Typumwandlung sinnvoll)
Write #intFNr, "Ali Yilmaz", CDate("19.05.1987"), False
'...
```


LE 10 – Fehler, Testen und Debugger

Debugger

– ermöglicht durch Haltepunkte, Variablenüberwachung und schrittweise Ausführung

– Nachvollziehen des tatsächlichen Programmablaufs und der Wertebelegung von Variablen

– dient der Analyse von identifizierten Fehlerzuständen

LE 10 – Fehler, Testen und Debugger

Fehlervermeidung/-auffindung

– Funktionen der Entwicklungsumgebung, des Compilers und weiterer Werkzeuge nutzen

– Programmierrichtlinien einführen

– Codereviews durchführen, ihre Ergebnisse dokumentieren

– Weitere Werkzeuge zur automatischen Analyse nutzen(z.B. Code Analysis, FindBugs, Checkstyle)

LE 10 – Fehler, Testen und Debugger

Testen

– Konstruktives und Destruktives Testen im Vier-Augen-Prinzip

– Wahl geeigneter Testfälle und Testdaten zur Abdeckung aller Eingabe-/Ausgabekombinationen

- Blackbox: Eingabedaten sollten repräsentativ für eine Gruppe von vergleichbaren Eingaben sein (Äquivalenzklassen) und Extremalwerte sowie Normalwerte abdecken
- Whitebox: Eingabedaten sollen so gewählt werden, dass alle Pfade innerhalb des Programms durchlaufen werden

– Testen von Modulen durch Erstellung von Testprozeduren

```
' Unterbricht die Programm, wenn
' boolscher Ausdruck falsch
Debug.Assert <BoolscherAusdruck>
```

–

LE 10 – Testfall und Testdaten

Testfall besteht aus Testablauf und den dort verwendeten Daten

- Testablauf, z.B. für Testen eines Anmeldungsvorgang (Login)
  - Ablauf 1
    - Eingabe des korrekten Benutzernamens
    - Eingabe des korrekte Kennwortes
    - Betätigen der Login-Schalfläche
    - Anwendung startet
  - Ablauf 2
    - Eingabe des korrekten Benutzernamens
    - Eingabe des falsches Kennwortes
    - Betätigen der Login-Schalfläche
    - Fehlermeldung "Ungültige Anmeldung"
  - Ablauf 3
    - Eingabe des falschen Benutzernamens
    - Eingabe des korrekten Kennwortes
    - Betätigen der Login-Schalfläche
    - Fehlermeldung "Ungültige Anmeldung"
- Testdaten
  - Eingabe eines korrekten Benutzernamen-Kennwortpaars (User1, Passwort1)
  - Eingabe eines falschen Benutzernamen-Kennwortpaars (User2, 123)
- Dokumentation erfolgt in Testfallspezifikation

LE 10 – White-Box- und Black-Box-Tests

Es gibt verschiedene Arten von Tests

- White-Box-Test
  - Testen unter Berücksichtigung der inneren Struktur des Testobjektes
- Black-Box-Test
  - Testen ohne Kenntnis der inneren Struktur; Grundlage ist die Spezifikation

Beispiel Testfälle finden (White/Black-Box)

Beispiel für Testfallfindung: Primzahltest (vereinfacht! Gilt so nur für Eingaben >= 2)

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE 10 – Beispiel Primzahltest (VBA)

```
Public Sub demo1101()
 Dim intZahl As Integer
 Let intZahl = Val(InputBox("Bitte eine Zahl eingeben: "))
 Debug.Print "Ist die Zahl eine Primzahl: " & isPrim(intZahl)
End Sub

Private Function isPrim(pintZahl As Integer) As Boolean
 Dim i As Integer
 Dim bolIsPrim As Boolean

 Let i = 2
 Let bolIsPrim = True

 Do While (bolIsPrim And i <= pintZahl / 2)
  If (pintZahl Mod i = 0) Then
   Let bolIsPrim = False
  End If
  Let i = i + 1
 Loop

 Let isPrim = bolIsPrim
End Function
```

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE 10 – Beispiel Primzahltest (VBA)

```
Public Sub demo1101()
 Dim intZahl As Integer
 Let intZahl = Val(InputBox("Bitte eine Zahl eingeben: "))
 Debug.Print "Ist die Zahl eine Primzahl: " & isPrim(intZahl)
End Sub

Private Function isPrim(pintZahl As Integer) As Boolean
```

Black-Box-Test

Welche Tests sind mindestens erforderlich?

```
End Function
```

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

Eingabedaten Black-Box-Test

Mindestens 2 Eingabedaten:

- eine Primzahl, z.B. 7
- eine Nicht-Primzahl, z.B. 300

Oft ist es noch günstig Extremalwerte zu betrachten, hier z.B.

- die kleinste Primzahl: 2

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE 10 – Eingabedaten Black-Box-Test

Da man i.a. nicht alle Eingabedaten untersuchen kann beschränkt man sich oft auf Äquivalenzklassen;

- Hier ergeben sich natürlicherweise mindestens 2 Äquivalenzklassen:
  - Die Primzahlen und
  - die Nicht-Primzahlen
- Die Nicht-Primzahlen könnte man evtl. noch in zwei Klassen zerlegen:
  - Gerade Nicht-Primzahlen
  - Ungerade Nicht-Primzahlen

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE 10 – Eingabedaten Black-Box-Test

Innerhalb jeder Äquivalenzklasse sollte man dann ein paar typische Vertreter wählen, sowie Extremalwerte

- Somit sind für den Primzahltest z.B. folgende Eingabedaten sinnvoll:
  - Primzahlen: 2 (Extremalwert), 17, 8999 (große Primzahl, eine größte gibt es nicht)
  - Nicht-Primzahlen:
    - gerade: 4 (Extremalwert), 100, 134568 (große)
    - ungerade: 9 (Extremalwert), 153, 168651

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE 10 – Beispiel Primzahltest (VBA)

```
Public Sub demo1101()
 Dim intZahl As Integer
 Let intZahl = Val(InputBox("Bitte eine Zahl eingeben: "))
 Debug.Print "Ist die Zahl eine Primzahl: " & isPrim(intZahl)
End Sub

Private Function isPrim(pintZahl As Integer) As Boolean
 Dim i As Integer
 Dim bolIsPrim As Boolean

 Let i = 2
 Let bolIsPrim = True

 Do While (bolIsPrim And i <= pintZahl / 2)
  If (pintZahl Mod i = 0) Then
   Let bolIsPrim = False
  End If
  Let i = i + 1
 Loop

 Let isPrim = bolIsPrim
End Function
```

White-Box-Test

Welche Tests sind mindestens erforderlich?

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE 10 – Eingabedaten White-Box-Test

Welche Tests sind mindestens erforderlich?

Jeder mögliche „Pfad“ durch den Programmcode sollte mindestens einmal durchlaufen werden!

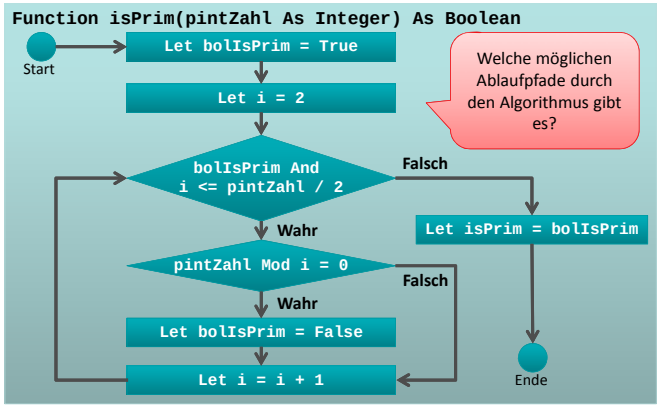
Bei Schleifen:

- Kein Durchlauf
- Ein Durchlauf
- Zwei Durchläufe
- Typische Anzahl Durchläufe
- Maximale Anzahl Durchläufe

Bei Bedingungen: jeder (Teil-)Ausdruck sollte wenigstens jeden der möglichen Werte einmal annehmen!

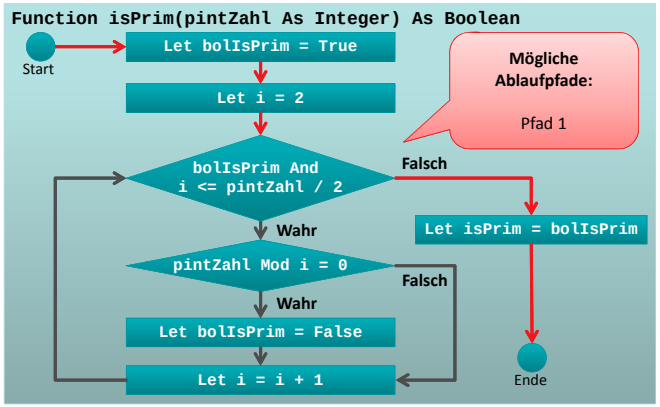
1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE 10 – Eingabedaten White-Box-Test[1]



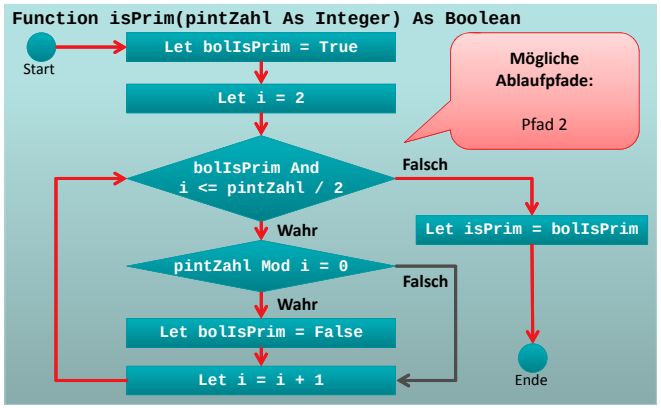
1) vgl. Beispiel aus Beuth-Hochschule für Technik, Prof. Dr. Ripphausen-Lipa, Skript "Programmierung 1"

LE 10 – Eingabedaten White-Box-Test[1]



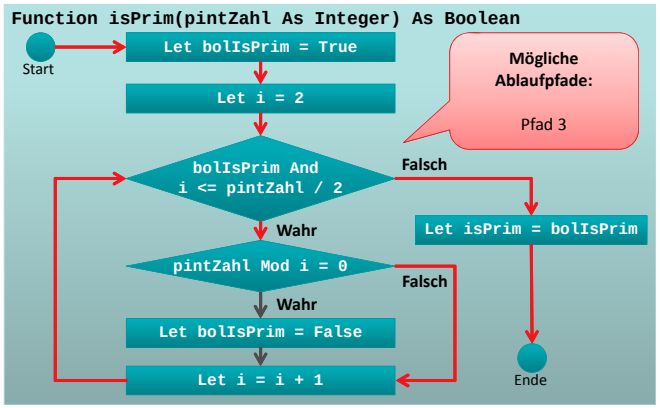
1) vgl. Beispiel aus Beuth-Hochschule für Technik, Prof. Dr. Ripphausen-Lipa, Skript "Programmierung 1"

LE 10 – Eingabedaten White-Box-Test[1]



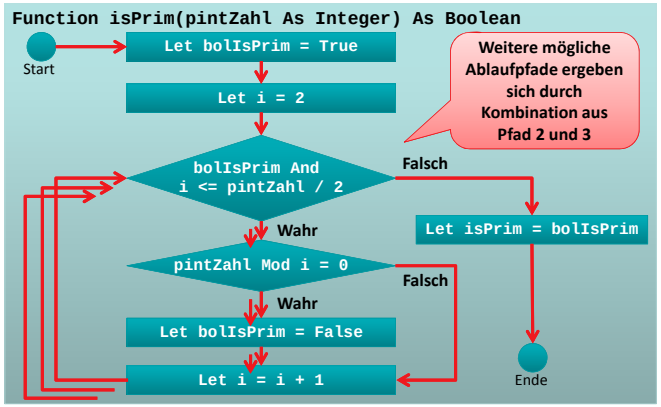
1) vgl. Beispiel aus Beuth-Hochschule für Technik, Prof. Dr. Ripphausen-Lipa, Skript "Programmierung 1"

LE 10 – Eingabedaten White-Box-Test[1]



1) vgl. Beispiel aus Beuth-Hochschule für Technik, Prof. Dr. Ripphausen-Lipa, Skript "Programmierung 1"

LE 10 – Eingabedaten White-Box-Test[1]



1) vgl. Beispiel aus Beuth-Hochschule für Technik, Prof. Dr. Ripphausen-Lipa, Skript "Programmierung 1"

Eingabedaten White-Box-Test1

Testdaten:

- Pfad 1: n = 4
- Pfad 2: n = 5 (einmal), n = 101 (mehrfach)
- Pfad 3: n = 2; n = 1 ← Zweiter Test zeigt falsches Ergebnis auf!
- Kombination Pfad 2, Pfad 1
  - Je einmal: z.B. 9
  - Pfad 2 dreimal, Pfad 1 einmal: 25
  - Pfad 2 mehrfach, Pfad 1 einmal: 121

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE 10 – Bemerkungen zum systematisches Testen

Es ist sinnvoll Testfälle mit gleichem Testablauf zu „automatisieren“, da bei jeder Änderung / Fehlerkorrektur am Besten alle Testfälle wieder durchlaufen werden (Hinweis: es gibt Tools, die dies unterstützen wie z.B. NUnit, JUnit)

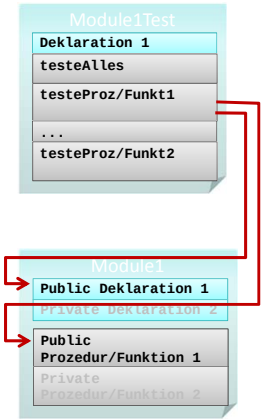
Es gibt sogar testgetriebenes Design / Vorgehen zur Entwicklung vor Software; bei dieser Methode werden zuerst die Testfälle entwickelt, bevor die eigentliche Software entwickelt wird

1) Quelle: Beuth-Hochschule für Technik, Ripphausen-Lipa, Skript Programmierung 1

LE 10 – Module und Testen

Umsetzung in VBA

- Modul und Testmodul bilden eine Einheit für den Test
- zu jedem Modul wird ein Testmodul erstellt (z.B. zu `Rechnung` das Modul `RechnungTest`)
- `prozedurNameTest` zur Test der Prozedur `prozedurName`
- `testeAlles()`-Prozedur zur Ausführung aller Test im Test-Modul (optional)



LE 10 – Module und Testen

Assertion (dt. Zusicherung)

- Prüfen einer vorher definierten Erwartung, die das Endergebnis erfüllen muss
- weicht das tatsächliche Ergebnis vom erwarteten Ergebnis ab, wird die Programmausführung unterbrochen
- hilft logische Fehler zu finden

VBA

```
' Unterbricht die Programm, wenn boolscher Ausdruck falsch
Debug.Assert <BoolscherAusdruck>
```

Innerhalb der Test-Prozeduren werden zu testende Funktionen innerhalb einer Debug.Assert-Anweisung aufgerufen.

LE 10 – Module und Testen

Beispiel: Modul mdlPrimzahlen

```
Option Compare Database
Option Explicit

Public Function isPrim(pintZahl As Integer) As Boolean

 ' ...

End Sub
```

Beispiel: Modul mdlPrimzahlenTest

```
Option Compare Database
Option Explicit

Private Sub isPrimTest()

 Debug.Assert mdlPrimzahlen.isPrim(0) = True
 Debug.Assert mdlPrimzahlen.isPrim(1) = True
 Debug.Assert mdlPrimzahlen.isPrim(2) = False
 Debug.Assert mdlPrimzahlen.isPrim(3) = True

End Sub
```

Inhalt

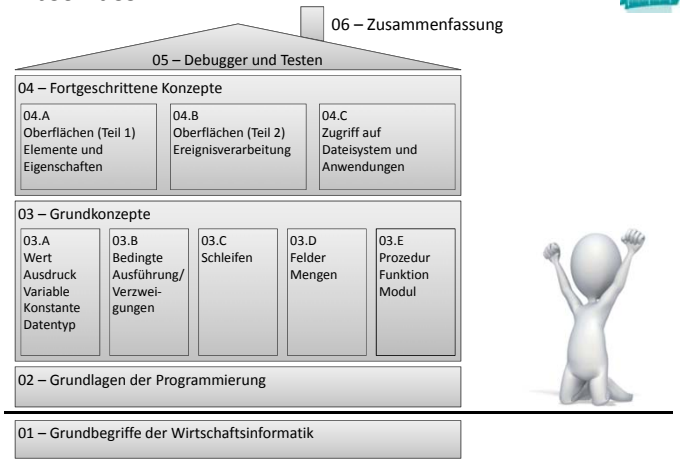
Einstieg
- LE 01: Wirtschaftsinformatik
- LE 02: Programmierung

Grundkonzepte
- LE 03: Variablen & Datentypen
- LE 04: Verzweigungen
- LE 05: Schleifen
- LE 06: Mengen & Felder
- LE 07: Prozeduren & Funktionen

Fortgeschrittene Konzepte
- LE 08: Ereignisse & GUI
- LE 09: Dateisystem
- LE 10: Fehler, Debugger & Testen

Abschluss

Abschluss



Literatur

- [Fink et al., 2001] A. Fink, G. Schneidereit, S. Voß: Grundlagen der Wirtschaftsinformatik. Physica-Verlag, Heidelberg (2001).
- [Hesse et al., 1984] Hesse, W.; Keutgen, H.; Luft, A.L; Rombach, H. D.: Ein Begriffssystem für die Softwaretechnik, in: Informatik-Spektrum, 7/1984, S. 200-213.

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Wirtschaftsinformatik 1
LE 11 – Zusammenfassung

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