

Visual Basic.NET

Programmierung

Zeitplan

Beginn	9.00 Uhr
Frühstück	10.30 Uhr - 10.50 Uhr
Mittag	12.30 Uhr - 13.30 Uhr
Kaffee	14.45 Uhr - 14.55 Uhr
Ende	16.00/30 Uhr

VB Programmierung - Inhalte

- Einführung in die Anwendungsentwicklung mit Visual Basic
- Entwicklungsumgebung Visual Studio
- Variablen, Datentypen und Operatoren
- Prozeduren und Programmablaufsteuerung
- Strukturierte Fehlerbehandlung und Debuggen
- Konzepte der Objektorientierten Programmierung
- Klassen, Objekte und Methoden
- Events und Eventhandler
- Strukturen und Schnittstellen
- Erstellen einfacher Windows Forms-Anwendungen
- Einsatz grundlegender Windows Forms-Steuerelemente

- Möglichkeiten der Programmiersprache BASIC kennenlernen
- Strukturiertes-, prozedurales Programmieren
 - Zerlegung des Problemraums
 - Kontrollstrukturen: Sequenzen, Verzweigungen, Schleifen
- Konzepte der objektorientierten Programmierung verstehen
- Mit Objekten arbeiten und Zugriffsmethoden erstellen
- Bestehenden Programmcode lesen und analysieren können
- Grafische Schnittstelle programmieren

Links

- http://www.tiobe.com/tiobe_index
- http://openbook.rheinwerk-verlag.de/einstieg_vb_2012/
- [https://msdn.microsoft.com/en-us/library/xk24xdbe\(v=vs.90\).aspx](https://msdn.microsoft.com/en-us/library/xk24xdbe(v=vs.90).aspx)
- <https://msdn.microsoft.com/en-us/library/sh9ywfdk.aspx>

Visual Basic

Visual Basic is a third-generation event-driven programming language and integrated development environment from Microsoft for its COM programming model first released in 1991 and declared legacy in 2008.

Microsoft intended Visual Basic to be relatively easy to learn and use.

Visual Basic was derived from BASIC and enables the rapid application development (RAD) of graphical user interface (GUI) applications, access to databases using Data Access Objects, Remote Data Objects, or ActiveX Data Objects, and creation of ActiveX controls and objects.

A programmer can create an application using the components provided by the Visual Basic program itself.

Over time the community of programmers developed third party components.

Programs written in Visual Basic can also use the Windows API, which requires external function declarations.

The final release was version 6 in 1998 (now known simply as Visual Basic).

A dialect of Visual Basic, Visual Basic for Applications (VBA), is used as a macro or scripting language within several Microsoft applications, including Microsoft Office.

https://en.wikipedia.org/wiki/Visual_Basic

Build Process - Scheme

source code: .vb



↓ PREPROCESSOR



vb code



libraries → ↓ COMPILER



CIL Bytecode

- common intermediate language
- assembly, executable code (debug, release)
- contains metadata (5)
 - describe types, members, references
 - locate and load classes
 - lay out instances in memory
 - resolve method invocations
 - run-time context boundaries



CLR(common language runtime) (9)

- interpreter, JIT (just-in-time compiler)
- producing and executing native code
- managed execution environment, managed code
- load and run
- enforce type safety
- array bound and index checking
- exception handling
- garbage collection
- threads

alternatively: NGEN(native image generator)

→ compiler outputs native code

- faster program execution
- needs still runtime environment
- loss of portability

Invoke compiler from command line (vbc)

1. Start cmd.exe: win+r, cmd
2. Create project folder on desktop: cd Desktop, md folder, cd folder
3. edit kommandozeile.vb (shift u. 1 \triangleq +)

```
Module Main
```

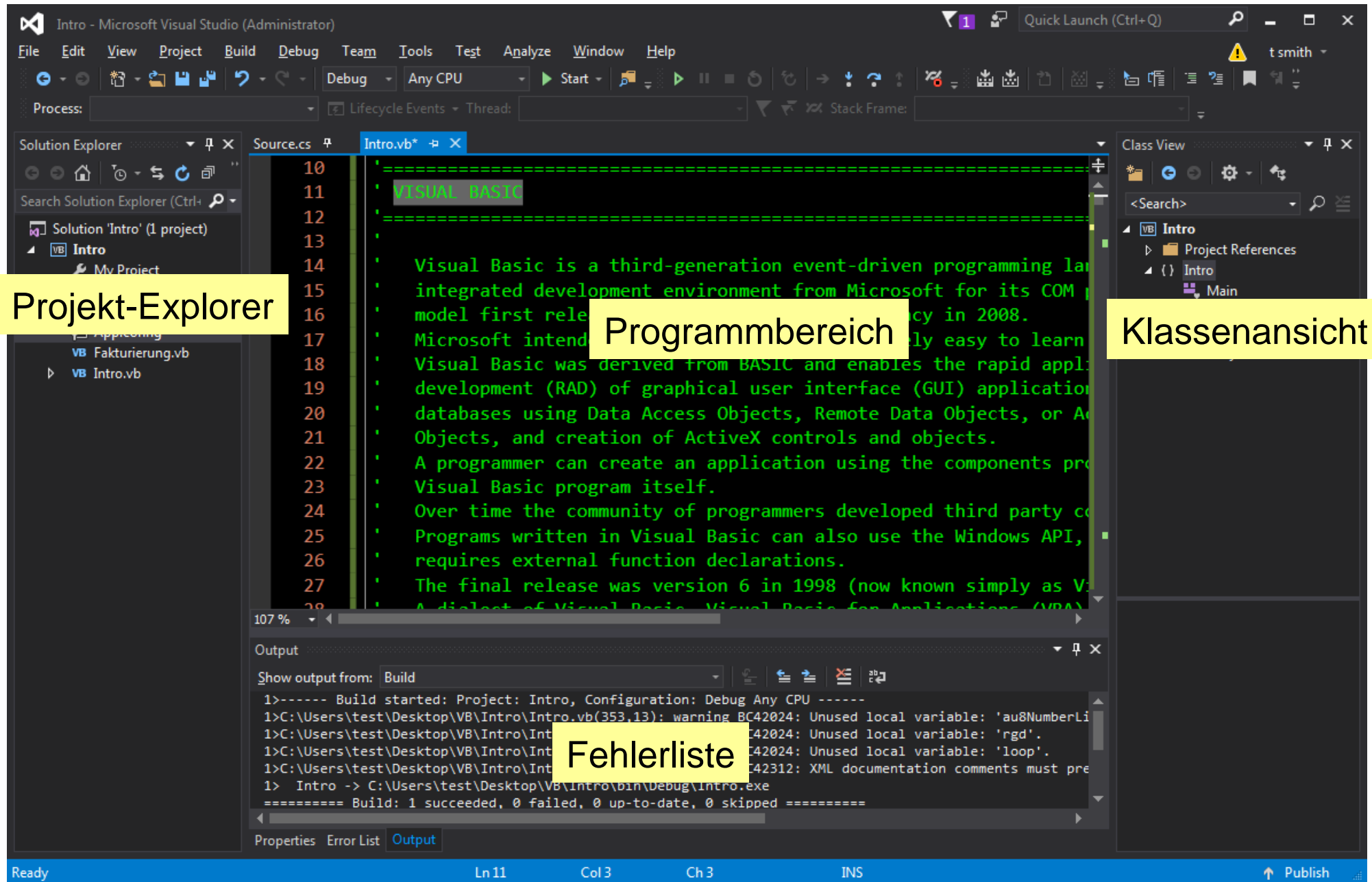
```
    Sub Main(args As String())
```

```
        System.Console.WriteLine("Hello " + args(0) + _  
            ", today is " + System.DateTime.Now)
```

```
    End Sub
```

```
End Module
```

4. Compiler:
C:\Windows\Microsoft.NET\Framework\v4.0.30319\vbc kommandozeile.vb
5. Aufruf: kommandozeile username



IDE - Tastenkombinationen

addnewitem	ctrl+shift+a	nextwindow	ctrl+tab, ctrl+f6
run	ctrl+f5	nextsplitpane	f6
debug	f5	nextcategory	ctrl+pgdn
togbreakpoint	f9	solutionexplorer	ctrl+alt+l
stepinto	f11	classview	ctrl+shift+c
stepover	f10	collapseexpand	ctrl+m+m
gotonexterrortag	f8	collapseexpall	ctrl+m+l
gotodefinition	f12	togbookmark	ctrl+k+k
peekdefinition	alt+f12	nextbookmark	ctrl+k+n
completeword	ctrl+space	togwhitespace	ctrl+shift+8
codesnippet	ctrl+k+x	formatdocument	ctrl+k+d
moveline	alt+arrow		
insertlineabove	ctrl+enter		
incsearch	ctrl+i		
nextoccurence	f3		
replacenext	alt+r		
rename	ctrl+r+r		
gotomatchbrace	ctrl+`		
lowercase	ctrl+u		
uppercase	ctrl+shift+u		
comment	ctrl+k+c		
uncomment	ctrl+k+u		
worddeletetoend	ctrl+del		
linecut	ctrl+l		
properties	alt+enter, f4		
showparameterinf	ctrl+shift+space		
encapsulatefield	ctrl+r+e		

IDE - Autoergänzung

- Bsp.: Variable

```
Public Sub Print()  
    Dim anzBestellungen = 0  
    ...  
    anz  
End Sub
```



Tab-Taste ergänzt zu `anzBestellungen`

IDE - Einstellungen u. Snippets

Einstellungen

project/properties/build/warning level/level4

tools/options/text editor/basic/line numbers

tools/options/projects and solutions/always show error list...

tools/options/projects and solutions/show output...

tools/customize/commands/toolbar/debug/add command/debug/start without debugging

Code snippets(tools/code snippets manager, ctrl+k+x)

Anwendung: snippet + tab bzw. 2 x tab

applog	filreadbin	propwrite
appstop	filwritetext	secdecrypt
arrloc	for	secencrypt
arrsort	foreach	select
cbarraylist	fuction	struct
cbcomplete	funcgeneric	sub
cbdata	funcpararr	subover
coliter	generic	syspower
coliterdict	idisposable	sysres
condown	ifelse	sys time
conreadport	imagebyte	tbarray
doloopun	lbclear	testc
enum	lbdate	testm
enumcustom	mathrand	tryc
enumstr	opadd	trycf
event	osuser	typedate
except	pbiterate	typeremove
filcreatefile	propdef	typestrbyte
filcreatefold	property	typetime
fileexistfile	propread	

IDE - Einstellungen u. Snippets

User defined snippet, xml-file, extension *.snippet

tools/code snippets manager/basic/import

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
```

```
  <CodeSnippet Format="1.0.0">
```

```
    <Header>
```

```
      <Title>WriteLine</Title>
```

```
      <Shortcut>cw</Shortcut>
```

```
    </Header>
```

```
    <Snippet>
```

```
      <Code Language="VB"><![CDATA[System.Console.WriteLine("$end$")]]></Code>
```

```
    </Snippet>
```

```
  </CodeSnippet>
```

```
</CodeSnippets>
```



Main-Methode (ohne Argumente)

Module Main

Sub Main() ' startroutine of the module

' set startroutine

' 1. solution explorer

' 2. solution/project name/my project (wrench) 2xclick → properties
(or project/properties)

' 3. startup object: module name

' output ide

Global.System.Diagnostics.Debug.Print("debug output")

' (only in debug configuration, in release configuration, the debug
' elements will not be compiled into the code)

' output at command prompt (command line interpreter)

Global.System.Console.WriteLine("console output")

Global.System.Console.WriteLine("Hello {0}, today Is {1}",
Global.System.Environment.UserName,
Global.System.DateTime.Now)

' short form

Console.WriteLine("Hello {0}, today Is {1}",
Environment.UserName,
Now)

End Sub

End Module



Main-Methode (mit Argumente)

```
Module Main
  Sub Main(args() As String)
    ' setting:
    ' project/properties/debug/command line arguments
    For Each item As String In args
      System.Console.WriteLine(item)
    Next
  End Sub
End Module
```

Preprocessor Directives

```
#If False Then
Module M1
    Sub Main()
        System.Console.WriteLine("Today is " + WeekdayName( Weekday(Today) ) )
    End Sub
End Module
#End If
```

```
#If True Then
Imports Microsoft.VisualBasic.DateAndTime

Module M2
    Sub Main()
        Microsoft.VisualBasic.Interaction.MsgBox("Current Month: " + _
                                                    MonthName( Month(Today) ) )
    End Sub
End Module
#End If
```



Preprocessor Directives

```
#Const SELECTOR = False
Imports Microsoft.VisualBasic.DateAndTime

Module Main
    Sub Main()
        Dim culture As New System.Globalization.CultureInfo("en-US") ' de-DE, fr-FR
        Dim engFormat As System.Globalization.DateTimeFormatInfo = culture.DateTimeFormat
    #If SELECTOR Then
        System.Console.WriteLine("Today Is " + engFormat.GetDayName(Weekday(Today)))
    #Else
        System.Console.WriteLine("Current Month: " + engFormat.GetMonthName(Month(Today)))
    #End If
    End Sub
End Module
```



Preprocessor Directives

```
#Const SELECTOR = 3
Imports Microsoft.VisualBasic.DateAndTime

Module Main
    Sub Main()
        #If SELECTOR = 1 Then
            System.Console.WriteLine("Current Hour: " + Hour(Now).ToString)
        #ElseIf SELECTOR = 2 Then
            System.Console.WriteLine("Current Minute: " + Minute(Now).ToString)
        #Else
            System.Console.WriteLine("Current Second: " + Second(Now).ToString)
        #End If
    End Sub
End Module
```



Identifiers

Naming conventions

- begin with alphabetic character or `_`
- must only contain alphabetic characters, decimal digits, and underscores
- must contain at least one alphabetic character or decimal digit if it begins with an underscore
- must not be more than 1023 characters long
- keywords can't be used as a identifiers (unless escaped)

,

Capitalization Styles (Naming Guidelines, MSDN)

Identifier	Case	Example
Class	Pascal	AppDomain
Enum type	Pascal	ErrorLevel
Enum values	Pascal	FatalError
Event	Pascal	ValueChanged
Exception class	Pascal	WebException
Read-only Static field	Pascal	RedValue
Interface	Pascal	IDisposable
Method	Pascal	ToString
Namespace	Pascal	System.Drawing
Parameter	Camel	typeName
Property	Pascal	BackColor
Protected instance field	Camel	redValue
Public instance field	Pascal	RedValue

Naming Conventions

```
Module NamingConventions
```

```
Sub test()
```

```
Dim UpperCamelCase = 0 ' or PascalCase
```

```
Dim lowerCamelCase = 0
```

```
' keywords are normally not allowed for variable designation
```

```
' except usage of [ ]
```

```
' Dim switch = 1 ' error
```

```
' Dim loop ' error
```

```
Dim [switch] = 1 ' ok, escaped keyword
```

```
Dim [loop] ' ok
```

```
' Hungarian notation, Systems Hungarian
```

```
' prefix encodes the actual data type
```

```
Dim nOrderNum = 1000 ' integer
```

```
Dim lAccountNum = 112233 ' long
```

```
Dim bMarried = True ' boolean
```

```
Dim yAge = 30 ' byte
```

```
Dim chGroup = "C" ' char
```

```
Dim wItemId = 7788 ' word
```

```
Dim au8NumberList() As Byte ' array of unsigned 8-bit integers
```

```
' Hungarian notation, Apps Hungarian
```

```
' encode the logical data type rather than the physical data type
```

```
Dim rwPosition = 1 ' row
```

```
Dim fPassed = False ' flag
```

```
Dim rgd() As Double ' range, array with double values
```

```
Dim cul = 100 ' count, number of elements, type unsigned long
```

```
End Sub
```

```
End Module
```

Übung - Bezeichner

a) Welche der folgenden Variablendefinitionen sind korrekt? 

__882
_888
_
item-nr
max
Max
/m/
abc\$
gerundeterWert
durchschnittØ
Ødurchschnitt
AB123__45
12ABC
µm
ölstand
Frage?
hello😊
[Dim]
Dim
grad
grad°
bit
byte
bytebyte
Gauß
a!
text§1
Nr.
hundert€
hundertx10
zehnÅ
Bruch½

Lösung - Bezeichner

a)

```
Dim __882 = 0      ' ok
Dim _888 = 0       ' ok
Dim _ = 0          ' ok
Dim item-nr = 0    ' error
Dim max = 0        ' ok
Dim Max = 0        ' error
Dim /m/= 0         ' error
Dim abc$ = 0       ' ok
Dim gerundeterWert = 0 ' ok
Dim durchschnittØ = 0 ' ok
Dim Ødurchschnitt = 0 ' ok
Dim AB123__45 = 0   ' ok
Dim 12ABC = 0       ' error
Dim µm = 0          ' ok
Dim ölstand = 0     ' ok
Dim Frage? = 0      ' ok
Dim hello😊 = 0      ' error
Dim [Dim] = 0       ' ok
Dim Dim = 0         ' error
Dim grad = 0        ' ok
Dim grad° = 0       ' error
Dim bit = 0         ' ok
Dim byte = 0        ' error
Dim bytebyte = 0    ' ok
Dim Gauß = 0        ' ok
Dim a! = 0          ' ok
Dim text§1 = 0      ' error
Dim Nr. = 0         ' error
Dim hundert€ = 100  ' error
Dim hundertx10 = 1000 ' ok
Dim zehnÅ = 10      ' ok
Dim Bruch½ = 0.5    ' error
```

Primitive data types, value types

- All numeric data types
- Boolean, Char and Date
- All structures, even if their members are reference types
- Enumerations, since their underlying type is always SByte, Short, Integer, Long, Byte, UShort, UInteger, or ULong

Visual Basic type	CLR type	Storage allocation
Boolean	Boolean	Depends on platform
Byte	Byte	1 byte
Char (single character)	Char	2 bytes
Date	DateTime	8 bytes
Decimal	Decimal	16 bytes
Double (double-precision)	Double	8 bytes
Integer	Int32	4 bytes
Long (long integer)	Int64	8 bytes
Object	Object (class)	4 bytes on 32-bit platform
SByte	SByte	1 byte
Short (short integer)	Int16	2 bytes
Single (single-precision)	Single	4 bytes
String (variable-length)	String (class)	Depends on platform
UInteger	UInt32	4 bytes
ULong	UInt64	8 bytes
User-Defined (structure)	(inherits ValueType)	Depends on platform
UShort	UInt16	2 bytes

Data Type Summary → <https://msdn.microsoft.com/en-us/library/47zceaw7.aspx>
→ https://en.wikibooks.org/wiki/Visual_Basic/Data_Types

Value types, Integral types

Internal representation

Einerkomplement

1. Vorzeichenbit MSB
2. Restbits invertieren

Problem: Doppelte Darstellung der 0 (pos. und neg)

Zweierkomplement

1. Zahl vorzeichenlos
2. Bits invertieren
3. 1 addieren

Idee: Komplementärzahl zu einer Zahl, die addiert 0 ergibt,

- Prüfung des Vorzeichenbits entfällt
- keine doppelte Darstellung der 0
- keine zusätzliche Steuerlogik

z.B. (Wortlänge 3) : $010 + 101 = 111 + 1 = 0$

=> Komplement zu $010(2)$ ($= 2(10)$) ist $110(2)$ ($= -2(10)$)

Rechnen mit Restklassen Mod 256 und finden geeigneter Repräsentanten

(z.B. $0..255$ äq $-128..127$ sind Rep. dieser 256 Restklassen bei Wortlänge 8)

Veranschaulichung: Alle Bits haben die gleiche Wertigkeit wie bei positiver Darstellung. Nur das MSB erhält die negative Wertigkeit.

Variante (Verlassen des Wertebereichs, Addition der negativen Zahl)

Bsp.: Wortlänge 3 Bits, damit nächster Wert $= 2^3 = 8 = 1000(2)$,

Darstellung der Zahl $-1(10)$ im 2er-Komplement => $8 + (-1) = 7(10) = 111(2)$

Zuordnungen (Aufteilung des Wertebereichs In pos. und neg. Werte) :

(10) (1er-K) (2er-K)

0	0 000	0 000
1	1 001	1 001
2	2 010	2 010
3	3 011	3 011
4	-1 110	-1 111 (=8-1)
5	-2 101	-2 110 (=8-2)
6	-3 100	-3 101 (=8-3)
7	0 111	-4 100 (=8-4)

Wertebereich des 2er-K: $-4..+3$, 4 negative, 4 positive Zahlen (incl. 0)

Value types, Integral types

```
Module BasicDatatypes
Sub integralTypes()
    '-----
    ' byte
    '-----
    Dim by As Byte ' zero-initialized
    Debug.Print(by) ' 0
    Debug.Print(Byte.MinValue) ' 0
    Debug.Print(Byte.MaxValue) ' 255
    by = 255
    Try
        by += 1 ' runtime error, OverflowException
    Catch e As System.Exception
        Debug.Print(e.ToString)
    End Try
    ' by = 256 ' buildtime error

    ' corresponding type in the .NET Framework
    Dim byStruct As System.Byte ' a structure
    byStruct = by
    Debug.Print(by.CompareTo(byStruct)) ' 0, equal
    Debug.Print(by.GetType.ToString) ' System.Byte
    Debug.Print(byStruct.GetType.ToString) ' System.Byte

    Dim sby As SByte
    Dim sbyStruct As System.SByte
    Debug.Print(SByte.MinValue) ' -128
    Debug.Print(System.SByte.MaxValue) ' 127
    ' sby = SByte.MaxValue + 1 ' buildtime error
    sby = SByte.MaxValue
    Try
        sby += 1 ' runtime error
    Catch e As Exception
        Debug.Print(e.ToString)
    End Try

    ' sby = System.Convert.ToByte(128) ' runtime error
End Sub
```

Value types, Integral types

```
'-----  
' char  
'-----  
Dim c As Char = "ABC" ' ok but stores only A  
c = "Z" ' automatic conversion string to single unicode character  
c = "Z"c ' specified as a char literal  
c = "ABC"  
' c = "ABC"c ' error, only one character  
Debug.Print(c) ' A  
Debug.Print(Global.Microsoft.VisualBasic.Strings.Asc(c)) ' 65  
c = Char.MinValue ' 0  
c = Char.MaxValue ' &hffff (=65535)  
Dim smiley = Global.Microsoft.VisualBasic.Strings.Chr(1)  
System.Console.WriteLine("smiley=" + smiley)  
  
' corresponding type in the .NET Framework  
Dim cStruct As System.Char  
  
'-----  
' short  
'-----  
Dim s As Short  
Debug.Print(Short.MinValue) ' -32768  
Debug.Print(Short.MaxValue) ' 32767  
s = 22S ' literal type  
Dim i16 As System.Int16 ' corresponding .NET type  
  
Dim us As UShort  
Debug.Print(UShort.MaxValue) ' 65535  
us = 22US ' literal type  
Dim uint16 As System.UInt16 ' corresponding .NET type
```

Value types, Integral types

```
'-----  
' integer  
'-----  
  
Dim i As Integer  
Debug.Print(Integer.MinValue) ' -2.147.483.648  
Debug.Print(Integer.MaxValue) ' 2.147.483.647  
i = 1I ' literal type  
Dim j% ' shorthand  
Dim i32 As System.Int32 ' corresponding .NET type  
  
Dim ui As UInteger  
Debug.Print(UInteger.MaxValue) ' 4.294.967.295  
Dim ui32 As System.UInt32 ' corresponding .NET type  
Debug.Print(System.UInt32.MaxValue)  
  
Dim l As Long  
Debug.Print(Long.MinValue) ' -9.223.372.036.854.775.808  
Debug.Print(Long.MaxValue) ' 9.223.37.203.685.477.5807  
l = 22L  
Dim ll& ' shorthand  
Dim i64 As System.Int64 ' corresponding .NET type  
  
Dim ul As ULong  
Debug.Print(ULong.MaxValue) ' 18.446.744.073.709.551.615  
Dim ui64 As System.UInt64 ' corresponding .NET type  
  
'-----  
' const values  
'-----  
  
Const GÜTERSLOH As UShort = 33333  
Const MOON As UInt32 = 384400UI  
Const SUN As UInt32 = 149600000UI  
Const LICHTJAHR As ULong = 9460730472580UL
```

Value types, Integral types

```
'-----  
' literals  
'-----
```

	+-----+	+-----+	+-----+
	Literal type character	Data type	Example
	+-----+	+-----+	+-----+
	S	Short	i = 347S
	I	Integer	i = 347I
	L	Long	i = 347L
	D	Decimal	i = 347D
	F	Single	i = 347F
	R	Double	i = 347R
	US	UShort	i = 347US
	UI	UInteger	i = 347UI
	UL	ULong	i = 347UL
	C	Char	i = "."C
	+-----+	+-----+	+-----+

Value types, Integral types

```
i = &O1234567 ' octal 0-7
j = &HCAFFEE ' hexadecimal 0-F
```

```
System.Console.WriteLine(i) ' 342391
System.Console.WriteLine(j) ' 13303790
```

```
Const BLUE As Integer = &HFF
System.Console.WriteLine("RGB BLUE={0:X6}", BLUE) ' 0000FF
```

```
' input an octal value
Dim octalNumber% = Global.Microsoft.VisualBasic.Val(
    "&O" + Global.Microsoft.VisualBasic.InputBox("input octal number"))
System.Console.WriteLine("decimal value=" & octalNumber)
```

```
' input a binary value
Dim binaryNumber As String = InputBox("input binary number")
Dim binaryValue As ULong
For i = 1 To binaryNumber.Length
    binaryValue = binaryValue + Val(
        Global.Microsoft.VisualBasic.Mid(
            binaryNumber, binaryNumber.Length - i + 1, 1)) *
        Global.System.Math.Pow(2, i - 1)
Next
System.Console.WriteLine("binary number {0}={1}(10)",
    binaryNumber,
    binaryValue)
```

Value types, Integral types

```
' convert decimal to binary
' variant 1 (toString method)
i = 200
System.Console.WriteLine(System.Convert.ToString(i, 2)) ' 11001000

' variant 2 (consecutive division)
binaryNumber = String.Empty
Do
    binaryNumber = i Mod 2 & binaryNumber
    i = i \ 2
Loop While CBool(i)
System.Console.WriteLine(binaryNumber) ' 11001000

' convert decimal to hexadecimal
' variant 1 (hex function)
Dim strHex As String = Global.Microsoft.VisualBasic.Hex(13303790)
System.Console.WriteLine(strHex) ' CAFFEE

' variant 2 (convert class)
strHex = System.Convert.ToString(13303790, 16)
System.Console.WriteLine(strHex.ToUpper) ' CAFFEE

' variant 3 (format method)
System.Console.WriteLine(String.Format("{0:X}", 13303790)) ' CAFFEE
```

Value types, Integral types

identifier types

Identifier type character	Data type	Example
%	Integer	Dim L%
&	Long	Dim M&
@	Decimal	Const W@ = 37.5
!	Single	Dim Q!
#	Double	Dim X#
\$	String	Dim V\$ = "Secret"

enclosing/type characters

Data type	Enclosing character	Appended type character
Boolean	(none)	(none)
Byte	(none)	(none)
Char	"	C
Date	#	(none)
Decimal	(none)	D or @
Double	(none)	R or #
Integer	(none)	I or %
Long	(none)	L or &
Short	(none)	S
Single	(none)	F or !
String	"	(none)

Value types, Integral types

' force type char

Const ch As Char = "A"c

' DateTime constants

Const [date] As DateTime = #1/11/2011#

Const [time] As DateTime = #1:10:20 AM#

' force type long

Const lng As Long = 45L

' force type single

Const sng As Single = 45.55!

Value types, Integral types

```
'-----  
' default values  
'-----
```

```
Dim charVal As Char, intVal As Integer, singleVal As Single,  
    decimalVal As Decimal, boolVal As Boolean,  
    dateTimeVal As System.DateTime, strVal As System.String,  
    varTypeVal As VariantType, varVal
```

```
System.Console.WriteLine(charVal)      '\0  
System.Console.WriteLine(intVal)       ' 0  
System.Console.WriteLine(singleVal)    ' 0  
System.Console.WriteLine(decimalVal)   ' 0  
System.Console.WriteLine(boolVal)      ' false  
System.Console.WriteLine(dateTimeVal)  ' 01.01.0001 00:00:00  
System.Console.WriteLine(strVal)       ' "", warning not assigned  
System.Console.WriteLine(varTypeVal)   ' 0  
System.Console.WriteLine(varVal)       ' warning not assigned
```

```
varVal += 1
```

```
System.Console.WriteLine(varVal)      ' 1
```

Value types, Integral types

```
'-----  
' integral promotion  
'-----  
  
'   Objects of an integral type can be converted to another wider integral  
'   type (that is, a type that can represent a larger set of values).  
'   This widening type of conversion is called integral promotion.  
i16 = by ' by (byte) is widened to an integer  
by = i16 ' ok, but may result in runtime error  
  
i16 = 256  
' by = i16 ' runtime error  
' by = System.Convert.ToByte(i16) ' runtime error  
  
' i16 = ch; ' error, no implicit conversion  
i16 = Global.Microsoft.VisualBasic.Asc(ch) ' ok  
System.Console.WriteLine(i16) ' 65  
i16 = Global.Microsoft.VisualBasic.Val(ch) ' ok  
System.Console.WriteLine(i16) ' 0  
i16 = Global.Microsoft.VisualBasic.Val("1"c) ' ok  
System.Console.WriteLine(i16) ' 1  
  
i32 = 1.1R ' implicit conversion, but data gets lost  
System.Console.WriteLine(i32) ' 1  
i32 = 3 / 4 ' 0.75 round up  
System.Console.WriteLine(i32) ' 1
```

Value types, Integral types

```
i32 = 1 << 31
System.Console.WriteLine(i32)           ' -2147483648
System.Console.WriteLine(Integer.MinValue) ' -2147483648
```

```
i32 = 1 << 32 ' 1 (cyclic shifting)
i32 = 1 << 33 ' 2
```

```
System.Console.WriteLine(1L << 63) ' -9223372036854775808
System.Console.WriteLine(1UL << 63) ' 9223372036854775808
```

```
i64 = 100000000000
```

```
Try
```

```
    by = i64 ' runtime error, OverflowException
```

```
Catch
```

```
    System.Console.WriteLine("overflow")
```

```
End Try
```

```
End Sub
```

```
End Module
```

Value types, Floating point types

IEEE 754

32 Bits |S 31|E 30 - 23|M 22 - 0|

Sign 1 Bit: 0+ 1-

Exponent 8 Bits

Mantissa 23 Bits

Form: $\pm(d_1.d_2d_3d_4\dots d_n) \times 2^e$

Ex.: $0.10101(2) \times 2^{-1} = (0 \times 2^0 + 1 \times 2^{-1} + \dots + 1 \times 2^{-5}) \times 2^{-1} = 0.328125(10)$

Normalization: force the integer part of the mantissa to be exactly 1

=> $1.0101(2) \times 2^{-2}$

For neg. Exponent: subtract 127, Ex.: 2^{-2} , $-2 = 125 - 127 \triangleq 01111101 (=125)$

=> $0\ 01111101 \times 1.0101$, 1 predecimal won't be saved => $0\ 01111101\ 010100000000000000000000$ (32 Bit)

Ex.: $0.09375(10) = 2^{-4} + 2^{-5}$, M: 000110000000000000000000 (23 Bit)

M: $1.100000000000000000000000$ (normalized, 4 x shift left)

=> Exp.: 4 x shift right

E: $2^{-4} = 01111011 (=123 - 127)$

=> $01111011 \times 1.100000000000000000000000$

=> $0\ 01111011 \times 1.100000000000000000000000$ with Signbit

=> $0\ 01111011 \times\ 100000000000000000000000$ (32 Bit)

=> $00111101110000000000000000000000$

Online Calculator IEEE-754 Floating-Point Conversion:

<http://babbage.cs.qc.cuny.edu/IEEE-754.old/Decimal.html>

The IEEE standard has four different rounding modes

- Round to Nearest (default) – rounds to the nearest value; if the number falls midway it is rounded to the nearest value with an even (zero) least significant bit, which occurs 50% of the time
- Round toward 0 – directed rounding towards zero
- Round toward $+\infty$ – directed rounding towards positive infinity
- Round toward $-\infty$ – directed rounding towards negative infinity

Value types, Floating point types

Module BasicDatatypes

Sub floatingPointTypes()

Dim f As Single ' 32bit, $\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$, 7 digits

f = 1.0 ' double literal

f = 1.0F

f = 1.0E+10F

f = 1.0E-10F

f = Single.NaN

f = Single.NegativeInfinity ' very small numbers

f = Single.PositiveInfinity ' very large numbers

System.Console.WriteLine(f * 1) ' +unendlich

System.Console.WriteLine(System.Math.Pow(2, 2000)) ' +unendlich

System.Console.WriteLine(Single.Epsilon) ' smallest positive value > 0

System.Console.WriteLine(Single.MinValue)

System.Console.WriteLine(Single.MaxValue)

' or

System.Console.WriteLine(System.Single.Epsilon)

System.Console.WriteLine(System.Single.MinValue)

System.Console.WriteLine(System.Single.MaxValue)

Dim d As Double ' 64bit, $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$, 15-16 digits

d = 1.0R

System.Console.WriteLine(System.Double.Epsilon)

System.Console.WriteLine(System.Double.MinValue)

System.Console.WriteLine(System.Double.MaxValue)

Value types, Floating point types

```
' automatic promotion
d = f + d ' infinity
f = f + d ' infinity
d = 2D * 1.0E+30F ' but loss of precision, literal itself is imprecisely
System.Console.WriteLine(d) ' = 2,00000003009493E+30

' -----
' Rounding error
' -----
' Loss of significance and inaccurate calculations
System.Console.WriteLine("double calculation: " & 100000000.0 - 1) ' = 99999999
System.Console.WriteLine("single calculation: " & 1.0E+8F - 1)      ' = 1E+08

' infinite loop
#If True Then
    Dim l As Long = 0
    For f = 0 To 100000000.0
        l += 1
        If l >= 100000000.0 Then
            System.Console.WriteLine("{0:N1} {1:F} loop still busy", l, f)
        End If
    Next
#End If
```

Value types, Floating point types

```
System.Console.WriteLine("{0:F15}", 1.1 * 1.1)           ' = 1,210000000000000
System.Console.WriteLine("{0:F15}", 1.1R * 1.1)           ' = 1,210000000000000
System.Console.WriteLine("{0:F15}", 1.1F * 1.1)           ' = 1,210000026226040
System.Console.WriteLine("{0:F15}", 1.1 * 1.1F)           ' = 1,210000026226040
System.Console.WriteLine("{0:F15}", 1.1F * 1.1F)         ' = 1,210000000000000
System.Console.WriteLine("{0:F15}", 1190 * 0.19F / 1.19) ' = 189,999992306493000
System.Console.WriteLine("{0:F15}", 1190 * 0.19 / 1.19)  ' = 190,000000000000000
```

' rounding error in associativity

```
System.Console.WriteLine(1.0E+10F * (1.0E+10F - 1.0E+10F + 1.0E-10F)) ' = 1
```

- ' calculation error when adjusting exponents

```
System.Console.WriteLine(1.0E+10F * (1.0E+10F + 1.0E-10F - 1.0E+10F)) ' = 0,
```

```
System.Console.WriteLine(
    10000000000.0 * (10000000000.0 + 0.0000000001 - 10000000000.0)) ' = 0
```

```
System.Console.WriteLine(
    10000000000.0R * (10000000000.0R + 0.0000000001R - 10000000000.0R)) ' = 0
```

```
System.Console.WriteLine(
    10000000000D * (10000000000D + 0.0000000001D - 10000000000D))
```

Value types, Floating point types

```
System.Console.WriteLine(1000000 + 0.0000001F)      ' = 1000000
System.Console.WriteLine(1000000.0F + 0.0000001F)    ' = 1000000
System.Console.WriteLine(1000000 + 0.000001F)         ' = 1000000
System.Console.WriteLine(1000000 + 0.00001F)          ' = 1000000
System.Console.WriteLine(1000000 + 0.0001F)           ' = 1000000
System.Console.WriteLine(1000000 + 0.001F)            ' = 1000000
System.Console.WriteLine(1000000 + 0.01F)             ' = 1000000
System.Console.WriteLine(1000000.0F + 0.1F)           ' = 1000000
System.Console.WriteLine(1000000 + 1.0F)              ' = 1000001
```

```
System.Console.WriteLine(1000000 + 0.0000001)        ' = 1000000,0000001
System.Console.WriteLine(1000000 + 0.000001)         ' = 1000000,000001
System.Console.WriteLine(1000000 + 0.00001)          ' = 1000000,00001
System.Console.WriteLine(1000000 + 0.0001)           ' = 1000000,0001
System.Console.WriteLine(1000000 + 0.001)            ' = 1000000,001
System.Console.WriteLine(1000000 + 0.01)             ' = 1000000,01
System.Console.WriteLine(1000000 + 0.1)              ' = 1000000,1
System.Console.WriteLine(1000000 + 1)                ' = 1000001
```


Value types, Floating point types

```
' implicit cast
```

Dim i As Integer

$$f = 1UL$$
$$\mathbf{i} = \mathbf{f}$$

```
' explicit cast
```

```
i = System.Convert.ToInt32(1.0F)
```

' Ariane V, 4.Juni 96, 36.7s self destruction after take-off

```
' Reason: Conversion float(64) to int(16)
```

`i = 9999.999R` ' = 10000

```
System.Console.WriteLine(i)
```

```
' breaking a floating point number into its binary significand (a floating point
```

' mantissa with an absolute value $0.5 \leq m < 1$, $0.00\dots01 \rightarrow 0.10\dots00$) and an in-

'tegral exponent for 2

$$f = -13.5F$$

```
Dim i64 As System.Int64 = System.BitConverter.DoubleToInt64Bits(f)
```

```
System.Console.WriteLine(System.Convert.ToString(i64, 2))
```

```
' = 1 10000000010 10110000000000000000000000000000000000000000
```

End Sub

End Module

Übung - IEEE-754 Floating-Point Conversion

- a) Berechnen Sie die floating-point Darstellung der Zahl 13.5
Verwenden Sie bei Bedarf die folgende Tabelle:

IEEE 754, Mantisse 24 Bit

Bit	Exponent	Wert	Wert dezimal
1	-1	1/2	0,500000000000000000000000
2	-2	1/4	0,250000000000000000000000
3	-3	1/8	0,125000000000000000000000
4	-4	1/16	0,062500000000000000000000
5	-5	1/32	0,031250000000000000000000
6	-6	1/64	0,015625000000000000000000
7	-7	1/128	0,007812500000000000000000
8	-8	1/256	0,003906250000000000000000
9	-9	1/512	0,001953125000000000000000
10	-10	1/1024	0,000976562500000000000000

Lösung - IEEE-754 Floating-Point Conversion

a)

$$13.5 \triangleq 1.6875 \times 2^3 \rightarrow 13.5 = 1.6875 \times 2^3$$

Exponent: $130-127=3 \ (\triangleq 2^3)$, $130(10) = 10000010(2)$

Mantissa: $1.6875(10) = 1.1011000000000000000000(2)$

Signed : 0

$$\rightarrow 13.5(10) = 0_10000010_101100000000000000000000 = 01000001010110000000000000000000(2)$$

Value types

```
Module BasicDatatypes
Sub valueTypes()
    '-----
    ' boolean
    '-----

    Dim b As Boolean = True ' default False
    ' equivalent to
    Dim bool As New Global.System.Boolean()

    ' corresponding .NET Framework type: System.Boolean structure
    Dim bStruct As System.Boolean
    Global.System.Diagnostics.Debug.Print(b.GetType.ToString) ' System.Boolean
    Debug.Print(bStruct.GetType.ToString) ' System.Boolean
    Debug.Print(b.CompareTo(bStruct)) ' 0, equal

    Dim ready, found As Boolean

    ready = found = False ' ⇔ ready = (found = False), assignment of a comparison
    Debug.Print(ready) ' True
    Debug.Print(found) ' False
    Debug.Print(ready.CompareTo(found)) ' 1, >0, not equal

    ready = found = True
    Debug.Print(ready.CompareTo(found)) ' 0, equal

    ready = True
    found = False
    Debug.Print(ready.CompareTo(found)) ' 1, >0, not equal

    ready = False
    found = True
    Debug.Print(ready.CompareTo(found)) ' -1, <0, not equal
End Sub
```

Value types

```
' automatic promotion
```

```
bool = 22
```

```
Debug.Print(bool) ' True (<>0 ⇔ True)
```

```
bool = 0
```

```
Debug.Print(bool) ' False
```

```
' bool = "yes" ' System.InvalidCastException
```

```
' bool = "no" ' System.InvalidCastException
```

```
bool = False * True
```

```
Debug.Print(bool) ' False
```

```
bool = True * -1
```

```
Debug.Print(bool) ' True
```

```
Debug.Print(CInt(bool)) ' -1
```

```
Dim ib As Integer
```

```
ib = True * -1
```

```
Debug.Print(ib) ' 1
```

```
Debug.Print(1 > 5 And 3 - 2 > 1 Or 4 <> 3) ' True
```

```
' =
```

```
Debug.Print(((1 > 5) And ((3 - 2) > 1)) Or (4 <> 3))
```

Value types

```
' parsing
bool = Boolean.Parse("true")
System.Console.WriteLine(bool) ' True
' bool = Boolean.Parse("1") ' runtime error

' converting cbool, inlined
bool = CBool(1) ' True
bool = CBool(1.0) ' True
bool = CBool(-1) ' True
bool = CBool(-10) ' True
bool = CBool(1D) ' True
bool = CBool(0) ' False
bool = CBool("1") ' True
bool = CBool("True") ' True
' bool = CBool("Yes") ' error

' converting convert class
Dim strBool As String = bool.ToString()
bool = System.Convert.ToBoolean(123)
' bool = System.Convert.ToBoolean("123") ' error
' bool = System.Convert.ToBoolean("-1") ' error
bool = System.Convert.ToBoolean("False")
bool = System.Convert.ToBoolean("tRuE")
' bool = System.Convert.ToBoolean("No") ' error
bool = System.Convert.ToBoolean(1.23)
' bool = System.Convert.ToBoolean(#1/1/2010#) ' error
bool = System.Convert.ToBoolean(123D)
```

Value types

' customized output

```
Dim bigCustomer As Boolean = True
```

```
System.Console.WriteLine("you get " +  
                          If(bigCustomer, "20%", "5%") + " discount")
```

' -----

' decimal

' -----

' The binary representation of a Decimal value consists of a 1-bit sign,
' a 96-bit integer number, and a scaling factor used to divide the 96-bit
' integer and specify what portion of it is a decimal fraction. The scaling
' factor is implicitly the number 10, raised to an exponent ranging from 0
' to 28.

- ' • not a floating-point data type
- ' + highest precision, up to 29 significant digits
- ' - performance, slowest of all numeric types

```
Dim de As Decimal
```

```
de = 0D
```

```
de = 1D
```

```
System.Console.WriteLine(Decimal.MaxValue) ' 79228162514264337593543950335
```

```
System.Console.WriteLine(Decimal.MinValue) '-79228162514264337593543950335
```

```
System.Console.WriteLine(Decimal.op_Increment(de)) ' 2
```

```
de += 1
```

Value types

' GetBits, returns an integer array with four elements.
' The first, second, and third elements of the returned array contain the
' low, middle, and high 32 bits of the 96-bit integer number, the fourth
' element of the returned array contains the scale factor and sign.
' Forth element: bits 0 to 15 unused and must be zero, bits 16 to 23 must
' contain an exponent between 0 and 28, which indicates the power of 10
' to divide the integer number, bits 24 to 30 are unused and must be zero,
' bit 31 contains the sign: 0 mean positive, 1 means negative.

```
Dim iArr(4) As Integer ' → arrays
de = -65535D
iArr(0) = System.Decimal.GetBits(de)(0)
iArr(1) = System.Decimal.GetBits(de)(1)
iArr(2) = System.Decimal.GetBits(de)(2)
iArr(3) = System.Decimal.GetBits(de)(3)
System.Console.WriteLine(Global.Microsoft.VisualBasic.Constants.vbCrLf +
    "{0:X} {1}", iArr(0), System.Convert.ToString(iArr(0), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(1), System.Convert.ToString(iArr(1), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(2), System.Convert.ToString(iArr(2), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(3), System.Convert.ToString(iArr(3), 2))
System.Console.WriteLine(
    "Bit 31 is " + If(iArr(3) And (1 << 31), "set (neg.)", "not set (pos.)"))
```


Value types

```
de = 255D
iArr(0) = System.Decimal.GetBits(de)(0)
iArr(1) = System.Decimal.GetBits(de)(1)
iArr(2) = System.Decimal.GetBits(de)(2)
iArr(3) = System.Decimal.GetBits(de)(3)
System.Console.WriteLine(vbCrLf +
    "{0:X} {1}", iArr(0), System.Convert.ToString(iArr(0), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(1), System.Convert.ToString(iArr(1), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(2), System.Convert.ToString(iArr(2), 2))
System.Console.WriteLine(
    "{0:X} {1}", iArr(3), System.Convert.ToString(iArr(3), 2))
System.Console.WriteLine(
    "Bit 31 is " + If(iArr(3) And (1 << 31), "set (neg.)", "not set (pos.)"))

' constructing a decimal
Dim lo, mid, hi As Integer, isNegative As Boolean, scale As Byte
lo = 0
mid = 100
hi = 1000
isNegative = True
scale = 3 ' decimal places
Dim conDec As New Decimal(lo, mid, hi, isNegative, scale)
System.Console.WriteLine(
    "constructed decimal=" & conDec) ' -18446744074139048345,600
```

Übung - Einfache Datentypen

- a) Definieren Sie für die folgenden Wertebereiche adäquate Variablen bzw. Konstanten.

Achten Sie darauf, den kleinstmöglichen Datentyp festzulegen.

Bezeichner	Wertebereich, Wert
Stückzahl	[0..1Mio]
ArtikelNummer	[1..10000]
Rating	["A"]
Smiley	["☺"], Codepage 850, \001
colorIndex	[0..10]
BetriebsZustand	[on/off]
Elementarladung	[1,60217733·10 ⁻¹⁹]
Milchstraße	[7000000000000000000]
RGB_Red	[FF0000]
Kostenfaktor	[35%]
MountEverest	[8848]
MilesToKm	[1,60934]
SerialPort16bit	[1100110011110000]

Übung - Einfache Datentypen

- b) Welche Werte haben die Variablen im folgenden Programmstück und wie viele Bytes werden von ihnen jeweils belegt?

```
Sub test()  
    Dim a, b, c As Integer  
    Dim d, e, f As Long  
    Dim g, h, i As Single  
    Dim j, k, l As Double  
    Dim m, n, o, p As SByte  
  
    a = 1.0E+10F * (1.0E+10F + 1.0E-10F - 1.0E+10F)  
    a = 3.75  
    b = 2 / 3  
    d = 1000000L * 100000  
    c = CInt(d)  
    c = System.Convert.ToInt32(d)  
    c = CType(d, Integer)  
    g = 1 / 3  
    h = 1D / 3D  
    j = 1 / 3  
    b = j  
    m = 127  
    n = m + 1  
    n = System.Convert.ToSByte(m + 1)  
    o = 255  
    p = -150  
    p = h  
End Sub
```

Übung - Einfache Datentypen

c) Transformieren Sie die folgenden mathematischen Formeln in gültige VB-Ausdrücke.

Verwenden Sie hierzu die [System.Math](#)-Klasse und die folgenden Funktionen:

- Die Wurzel eines Ausdrucks kann dann mittels der Sqrt-Funktion berechnet werden:
 $\text{Sqrt}(4) = 2$
- Die Potenz eines Ausdrucks:
 $\text{Pow}(2,3) = 2^3 = 8$
- Der Betrag $|a|$ eines Ausdrucks:
 $\text{Abs}(-1) = 1$
- Die Exponentialfunktion:
 $\text{Exp}(2) = e^2$
- π , Pi

I. $f(A, B) = \frac{|A - B|}{|A| + |B|}$

II. $f(C, L, R) = \sqrt{\frac{1}{LC} - \left(\frac{R}{2L}\right)^2}$

III. $f(N) = e^{-N} N^N \sqrt{2N\pi}$

Übung - Einfache Datentypen

d) Überprüfen Sie die folgenden Ausdrücke auf Richtigkeit.

```
Sub test()  
  Const q As Integer = 1  
  Const r As Integer = 7.77  
  Const s As Double  
  Const t As Single = 1000UI  
  Const u As UInt32 = 333L  
  Const v As Char = 1  
  Const fire As Short = 112  
  Const high As Short = 1  
  Const low As Short = OFF  
  Const w As Char = "1"  
  Const error As Char = "error"  
  Const x As Integer = "2"  
  Const y As Single = 1.0F  
  Const z As Integer = w + x  
  Const four As Char = "f" - "a"  
  Const A As Integer = x + y  
  Const B As Integer = 0  
  Dim C As Integer = 1  
  Const D As Single = C  
End Sub
```

Lösung - Einfache Datentypen

```
Module DataTypesEx
Sub Main()
    ' a)
    Dim Stückzahl As UInt32 = 0
    Dim ArtikelNummer As UShort = 555
    Dim Rating As Char = "A"c
    Rating = CChar("A")
    Dim Smiley As Char = System.Convert.ToChar(1)
    Smiley = Global.Microsoft.VisualBasic.Chr(1)
    Dim colorIndex As SByte = 0
    Dim BetriebsZustand As Boolean = True
    Const [ON] As Byte = 1
    Const OFF As Byte = 0
    Dim yBetriebsZustand As Byte = [ON]
    Const Elementarladung As Decimal = 0.000000000000000000160217733D
    Dim Milchstraße0 As ULong = 700000000000000000UL
    Milchstraße0 = System.Convert.ToInt64(7.0E+17)
    Milchstraße0 = CULng(7.0E+17)
    Const RGB_Red As UInt32 = &HFF0000
    Const Kostenfaktor As Single = 0.35F
    Const MountEverest As UShort = 8848
    Const MilesToKm As Double = 1.60934
    Dim SerialPort16bit As System.UInt16 = System.Convert.ToUInt16("1100110011110000", 2)

    System.Console.WriteLine(Stückzahl)
    System.Console.WriteLine(ArtikelNummer)
    System.Console.WriteLine(Rating)
    System.Console.WriteLine(Smiley)
    System.Console.WriteLine(colorIndex)
    System.Console.WriteLine(BetriebsZustand)
    System.Console.WriteLine(yBetriebsZustand)
    System.Console.WriteLine(Elementarladung)
    System.Console.WriteLine(Milchstraße0)
    System.Console.WriteLine(RGB_Red)
    System.Console.WriteLine(Kostenfaktor)
    System.Console.WriteLine(MountEverest)
    System.Console.WriteLine(MilesToKm)
    System.Console.WriteLine(SerialPort16bit)
```

Lösung - Einfache Datentypen

```
' b)
Dim a, b, c As Integer
Dim d, e, f As Long
Dim g, h, i As Single
Dim j, k, l As Double
Dim m, n, o, p As SByte

System.Console.WriteLine("Size of Integer: " &
    System.Runtime.InteropServices.Marshal.SizeOf(a) & " Bytes") ' = 4
System.Console.WriteLine("Size of Long: " &
    System.Runtime.InteropServices.Marshal.SizeOf(d) & " Bytes") ' = 8
System.Console.WriteLine("Size of Single: " &
    System.Runtime.InteropServices.Marshal.SizeOf(g) & " Bytes") ' = 4
System.Console.WriteLine("Size of Double: " &
    System.Runtime.InteropServices.Marshal.SizeOf(j) & " Bytes") ' = 8
System.Console.WriteLine("Size of SByte: " &
    System.Runtime.InteropServices.Marshal.SizeOf(m) & " Bytes") ' = 1

a = 1.0E+10F * (1.0E+10F + 1.0E-10F - 1.0E+10F) ' = 0
a = 3.75 ' = 4
b = 2 / 3 ' = 1
d = 100000L * 100000 ' = 10.000.000.000
' c = CInt(d) ' error, overflow
' c = System.Convert.ToInt32(d) ' error, overflow
' c = CType(d, Integer)
c = Bin_To_Dec(
    Global.Microsoft.VisualBasic.Right(' convert only 32 bit
        System.Convert.ToString(d, 2), 32)) ' = 1410065408
' or
c = Convert.ToInt32(Right(Convert.ToString(d, 2), 32), 2)
g = 1 / 3 ' = 0,333...43
h = 1D / 3D ' = 0,333...43
j = 1 / 3 ' = 0.333...31
b = j ' = 0
m = 127 ' = 127
' n = m + 1 ' error, overflow
' n = System.Convert.ToSByte(m + 1) ' error, overflow
Try
    n = System.Convert.ToSByte(m + 1)
Catch
    n = n.GetType().GetField("MaxValue").GetValue(Nothing)
    ' or
    n = If(TypeName(n) = "SByte", SByte.MaxValue, 0)
    ' VarType has no SByte enumeration member
End Try
' o = 255 ' error, overflow
' p = -150 ' error, overflow
p = h ' = 0
```

Lösung - Einfache Datentypen

```
' c)
System.Console.WriteLine(System.Math.Sqrt(4)) ' = 2
System.Console.WriteLine(System.Math.Pow(2, 3)) ' = 8
System.Console.WriteLine(System.Math.Abs(-1)) ' = 1
System.Console.WriteLine(System.Math.Exp(2)) ' = 7,38905609893065
System.Console.WriteLine(System.Math.PI) ' = 3,14159265358979

Dim _A, _B, _L, _C, _R, _N
Dim f_i = System.Math.Abs(_A - _B) / (System.Math.Abs(_A) + System.Math.Abs(_B))
Dim f_ii = System.Math.Sqrt(1 / (_L - _C) - System.Math.Pow(_R / (2 * _L), 2))
Dim f_iii = System.Math.Exp(-_N) *
            System.Math.Pow(_N, _N) *
            System.Math.Sqrt(2 * _N * System.Math.PI)

' d)
Const q As Integer = 1
Const r As Integer = 7.77 ' = 8
' Const s As Double ' error, missing initialization
Const s As Double = 0
Const t As Single = 1000UI
Const u As UInt32 = 333L
' Const v As Char = 1 ' error, no implicit conversion
' Const v As Char = System.Convert.ToChar(1) ' error, not constant
' Const v As Char = CChar(1) ' error, can't be converted
Const v As Char = Global.Microsoft.VisualBasic.Chr(1)
Const fire As Short = 112
Const high As Short = 1
Const low As Short = OFF
Const w As Char = "1"c
' Const error As Char = "error" ' error, keyword
Const [error] As Char = "error"
' Const x As Integer = "2" ' error, no conversion from string to integer
' Const x As Integer = Integer.Parse("2") ' error, not
Const x As Integer = Global.Microsoft.VisualBasic.Asc("2")
Const y As Single = 1.0F
Const z As Integer = Asc(w) + x
' Const four As Char = "f" - "a" ' error no conversion from double to char
Const four As Char = Chr(Asc("f") - Asc("a"))
Const A_ As Integer = x + y
' B_ const As Integer = 0 ' error
Const B_ As Integer = 0
' Dim C_ As Integer = 1 ' error prone, C_ is not const → 'Failed to emit module'
Const C_ As Integer = 1
Const D_ As Single = C_
End Sub
```


Lösung - Einfache Datentypen

```
Function Bin_To_Dec(ByVal Bin As String)
    Dim dec As Double = Nothing
    Dim length As Integer = Len(Bin)
    Dim temp As Integer = Nothing
    Dim x As Integer = Nothing
    For x = 1 To length
        temp = Val(Mid(Bin, length, 1))
        length = length - 1
        If temp <> "0" Then
            dec += (2 ^ (x - 1))
        End If
    Next
    Return dec
End Function
End Module
```

Value types

```
' -----  
' datetime structure  
' -----  
  
Dim dateVal As System.DateTime = System.DateTime.Today  
Dim timeVal As System.DateTime = System.DateTime.Now  
System.Console.WriteLine("Date: " + dateVal) ' date  
System.Console.WriteLine("Time: " + timeVal) ' date + time  
System.Console.WriteLine("Time: " + timeVal.ToLocalTime)  
System.Console.WriteLine("Time: " + timeVal.ToShortTimeString)  
System.Console.WriteLine("Time: " + timeVal.TimeOfDay.ToString)  
System.Console.WriteLine("Time: " + timeVal.ToString("HH:mm:ss"))  
System.Console.WriteLine(dateVal.ToLongDateString) ' weekday  
  
' time measuring  
Dim stopW As New System.Diagnostics.Stopwatch  
stopW.Start()  
Dim counter  
start:  
Math.Sqrt(Math.Sin(counter) * Math.Cos(counter))  
counter += 1  
If counter < 20000000.0 Then GoTo start  
stopW.Stop()  
System.Console.WriteLine("elapsed milliseconds=" & stopW.ElapsedMilliseconds)
```

Value types

```
dateVal = Today.AddDays(40)
System.Console.WriteLine("future date: " & dateVal)

Dim duration As System.TimeSpan = Today.AddDays(20).AddHours(10) -
    Today.AddDays(14).AddHours(3)

System.Console.WriteLine("difference: " & duration.Days & " days " &
    duration.Hours & " hours")
```

Value types

```
' -----  
' enum  
' -----  
' Enum MyEnum : ValueOne : ValueTwo : End Enum  
' error, enums can't appear within a method body  
  
' -----  
' struct  
' -----  
' Structure MyStruct : Dim ItemOne : Dim ItemTwo : End Structure  
' error, structs can't appear within a method body  
End Sub ' !  
  
Enum ColorType : RED : GREEN = &HFF00 : BLUE : BLACK : WHITE : YELLOW : End Enum  
  
Public Enum CurrencyType As Integer ' only integral types  
    EUR = 1  
    USD = 110  
    GBP = 80  
    JPY = 11000  
    CHF  
    AUD  
    [TRY]  
    BRL  
    CAD  
    HKD = 850  
    DKK  
    KWD = 33  
    THB  
    ILS  
    NOK  
End Enum
```

Value types

```
Structure CustomerType
    Dim customerCompany As String
    Dim contactPerson As String
    Dim id As UShort
    Dim customerAddress As String
    Dim customerSince As Date
    Dim customerSales As Decimal
    Structure phone
        Dim countryCode As String
        Dim areaCode As String
        Dim number As String
        Dim directDial As String
    End Structure
End Structure
```

Value types

```
Sub enumStructTypes()  
    '-----  
    ' iterate through enumeration  
    '-----  
    ' elements of the array are sorted by the binary values of the enumeration constants  
    Dim enumItems As System.Array  
    enumItems = System.Enum.GetNames(GetType(CurrencyType)) ' GetNames type string  
    For Each item As String In enumItems  
        System.Console.WriteLine("enum id: " + item)  
    Next  
  
    Dim enumValues As System.Array  
    ' or (GetType → Type Currency)  
    System.Console.WriteLine(GetType(CurrencyType)) ' → Intro.BasicDatatypes+Currency  
    Dim enumIntValues() As Integer = ' conversion to int array  
        CType(System.Enum.GetValues(GetType(CurrencyType)), Integer())  
    enumValues = System.Enum.GetValues(GetType(CurrencyType))  
    For Each item As Integer In enumValues  
        System.Console.WriteLine("enum value: " & item)  
    Next
```

Value types

```
' -----  
' declare variable of  
' enum type  
' -----  
  
Dim color As ColorType  
color = ColorType.GREEN  
System.Console.WriteLine(color)  
System.Console.WriteLine(color.ToString)  
  
' enum value formatting  
System.Console.WriteLine(System.Enum.Format(GetType(ColorType), color, "d"))  
' or  
System.Console.WriteLine([Enum].Format(GetType(ColorType), color, "x"))  
  
Dim currency As CurrencyType  
currency = CurrencyType.HKD  
Dim invoice As Double = 15980.88  
System.Console.WriteLine("invoice total in eur={0:##0.00 EUR}", _  
                           invoice / currency * 100)
```

Value types

```
'-----  
' declare variable of  
' structure type  
'-----
```

```
Dim customer As CustomerType  
customer.customerSales = 380500.45  
customer.id = 720  
Dim customerPhone As CustomerType.phone  
customerPhone.areaCode = "(214)"
```

```
End Sub
```

```
End Module
```


Value types, variant types

```
Sub variantTypes()  
    ' -----  
    ' defining mixed types  
    ' -----  
    Dim v1, v2, v3 As Integer, v4, v5 As Single, v6  
    ' explicit initialization not allowed when declaring several variables  
  
    ' -----  
    ' determine type of variant  
    ' -----  
    Debug.Print("v1 is of type " &  
        Global.Microsoft.VisualBasic.VarType(v1)) ' 3  
    Debug.Print("v1 is of type " &  
        Global.Microsoft.VisualBasic.VarType(v1).ToString) ' Integer  
    Debug.Print("v2 is of type " + VarType(v2).ToString) ' Integer  
    Debug.Print("v3 is of type " + VarType(v3).ToString) ' Integer  
    Debug.Print("v4 is of type " + VarType(v4).ToString) ' Single  
    Debug.Print("v5 is of type " + VarType(v5).ToString) ' Single  
    Debug.Print("v6 is of type " + VarType(v6).ToString) ' Object  
    ' VB.NET does not support the Variant data type. The Object data type is an universal  
    ' datatype, that can hold data of any other data type
```

Value types, variant types

```
' VarType function returns the type of a variant object defined  
' by the VariantType enumeration
```

```
' Enum VariantType  
'     Empty = 0  
'     Null = 1  
'     [Short] = 2  
'     [Integer] = 3  
'     [Single] = 4  
'     [Double] = 5  
'     Currency = 6  
'     [Date] = 7  
'     [String] = 8  
'     [Object] = 9  
'     [Error] = 10  
'     [Boolean] = 11  
'     [Variant] = 12  
'     DataObject = 13  
'     [Decimal] = 14  
'     [Byte] = 17  
'     [Char] = 18  
'     [Long] = 20  
'     UserDefinedType = 36  
'     Array = 8192  
' End Enum
```

Value types, variant types

```
' check
If VarType(v6) = Global.Microsoft.VisualBasic.VariantType.Object Then _
    Debug.Print("v1 is an Object")
' alternatively
If VarType(v6) = Global.Microsoft.VisualBasic.Constants.vbObject Then _
    Debug.Print("v1 is an Object")

' -----
' nothing
' -----

Dim var
' check for 'nothing' (literal)
' nothing represents the default value of any data type
' reference type is set to a null reference
' (not associated with any object)
Debug.Print(var Is Nothing) ' True
Debug.Print(IsNothing(var)) ' True
Debug.Print(var = Nothing) ' True
```

Value types, variant types

```
var = New String("test")
Debug.Print(var Is Nothing) ' False
Debug.Print(IsNothing(var)) ' False
Debug.Print(var = Nothing) ' False

var = ""
Debug.Print(var Is Nothing) ' False
Debug.Print(IsNothing(var)) ' False
Debug.Print(var = Nothing) ' True! → avoid '= Nothing' or '<> Nothing'
Debug.Print(var.Equals(Nothing)) ' False

' check for 'nothing' with value types
Dim i As Integer
i = Nothing ' → value type is set to its default value
Debug.Print(i) ' 0
' Debug.Print(i Is Nothing) ' error, operator can't be applied to value types
Debug.Print(IsNothing(i)) ' False
Debug.Print(i = Nothing) ' True → contradiction, error prone

Dim varT As VariantType ' numeric
Debug.Print(varT) ' 0 ⇔ Empty
' Debug.Print(varT Is Nothing) ' error, operator can't be applied
Debug.Print(IsNothing(varT)) ' False
Debug.Print(varT = Nothing) ' True → contradiction, error prone

' negation
If Not IsNothing(var) Then _
    Debug.Print("var points to an object")
' or
If Not var Is Nothing Then _
    Debug.Print("var points to an object")
' or
If var IsNot Nothing Then _
    Debug.Print("var points to an object")
```

Value types, variant types

```
'-----  
' nullable types  
'-----  
' • declaring a variable as nullable expands the applicable methods  
'   with.HasValue and Value to indicate whether a value has been assigned  
' • underlying type must be a value type  
' • 'is' operator can be used  
' • assigning nothing to a nullable variable sets the value to null  
' • nullable type is constructed from the generic Nullable(Of T) structure
```

```
Dim size As Integer  
size = Nothing  
Debug.Print(size) ' 0 (default value)  
' If size.HasValue Then Debug.Print("size has been assigned") ' error  
' size is not of type nullable  
If IsNothing(size) Then _  
    Debug.Print("not assigned") ' no, error prone  
If size = Nothing Then _  
    Debug.Print("not assigned") ' yes  
If size <> Nothing Then _  
    Debug.Print("assigned") ' no  
  
size = 0  
System.Console.WriteLine(size = Nothing) ' true, still default value
```

Value types, variant types

```
size = 10
If IsNothing(size) Then _
    Debug.Print("not assigned") ' no
If size = Nothing Then _
    Debug.Print("not assigned") ' no, <> default value
If size <> Nothing Then _
    Debug.Print("assigned") ' yes

' example
Dim clientGroup As String
Dim newClient As Boolean
newClient = True
If newClient Then clientGroup = ""
If clientGroup = Nothing Then MsgBox("error, group not initalized") ' yes
If IsNothing(clientGroup) Then
    MsgBox("error, group not initalized") ' but clientGroup was assigned
Else
    MsgBox("already initialized")
End If

' nullable (only for value types)
Dim numClients As System.Nullable(Of UShort)
Dim numOrders? As Integer ' shorthand
numOrders = Nothing
If numOrders.HasValue Then
    Debug.Print("size has been assigned")
    Debug.Print(numOrders.Value)
Else
    Debug.Print("error, not assigned")
End If
```

Value types, variant types

```
numOrders = 10
```

```
If numOrders.HasValue Then
```

```
    Debug.Print("size has been assigned")
```

```
    Debug.Print(numOrders.Value)
```

```
End If
```

```
' -----
```

```
' GetType, GetTypeCode,
```

```
' VarType TypeName, TypeOf
```

```
' -----
```

```
' • GetType returns an instance of Type class
```

```
' • TypeOf works only with reference/variant types (option infer off)
```

```
' • TypeName returns a string
```

```
' • VarType returns a member of the VariantType enumeration
```

```
var = 1
```

```
Debug.Print(var.GetType.ToString) ' System.Int32, qualified name
```

```
Debug.Print(var.GetTypeCode) ' 9
```

```
Debug.Print(VarType(var)) ' 3
```

```
Debug.Print(VarType(var).ToString) ' Integer
```

```
Debug.Print(TypeName(var)) ' Integer, shortname
```

```
If TypeOf var Is Integer Then Debug.Print("var is type of integer") ' yes
```

Value types, variant types

```
var = "test"  
Debug.Print(var.GetType.ToString) ' System.String  
Debug.Print(var.GetTypeCode)      ' 18  
Debug.Print(VarType(var))          ' 8  
Debug.Print(VarType(var).ToString) ' String  
Debug.Print(TypeName(var))         ' String  
If TypeOf var Is String Then Debug.Print("var is type of string")
```

```
var = Nothing  
' Debug.Print(var.GetType.ToString) ' error  
' Debug.Print(var.GetTypeCode)      ' error  
Debug.Print(VarType(var))          ' 9  
Debug.Print(VarType(var).ToString) ' Object  
Debug.Print(TypeName(var))         ' Nothing  
If TypeOf var Is Object Then  
    Debug.Print("var is type of object") ' yes  
Else  
    Debug.Print("var is nothing")  
End If  
' if objectexpression is null, then TypeOf...Is returns False,  
' and ...IsNot returns True
```

End Sub

End Module

Optionen

Option Explicit On ' forces explicit declaration of all variables

Option Strict Off ' forces declaration of datatype → 'as' is required

' speeds up the execution of code, compiled code might have to convert back and forth between Object and other data types, which reduces performance, implicit data type conversions are disallowed (only 'widening' conversions)

Option Infer On ' declare local variable without explicitly stating a data type, compiler infers the data type of a variable from the type of its initialization
' expression: `dim i = 1` → i is type of integer

Option Compare Binary ' case sensitive

' **Option Compare Text** ' not case sensitive

Type Conversion

- **Parse** bzw. **TryParse**-Methoden

- Parse throws Exception

- Prüfausdruck muss in einem Try/Catch-Statement eingebettet sein
Schema:

```
Try
```

```
...
```

```
.Parse(Expression)
```

```
Catch Exception
```

```
Exception Handling
```

```
End Try
```

- TryParse returns Boolean

- Rückgabewert wird auf True/False geprüft
 - zusätzliche Definition einer Variablen, die das geparste Ergebnis speichert

- Methoden der **Convert**-Klasse

- automatische Typumwandlung

Type Conversion

```
Module Conversions
Sub conversion()
    '=====
    ' string → int
    '=====

    '-----
    ' Parse/TryParse
    '-----

    ' Parse throws exception on failing
    ' TryParse returns a bool

    Dim i? As Integer = Nothing ' Nothing for indicating a parser error

start: ' Parse
    Try ' exception handling
        Select Case Cint(
            Global.Microsoft.VisualBasic.Interaction.InputBox("select 1..6"))
            Case 1
                i = Integer.Parse("true") 'error
            Case 2
                i = Integer.Parse("123")
            Case 3
                i = Integer.Parse("-000123")
            Case 4
                i = Integer.Parse("1.23") 'error
            Case 5
                i = Integer.Parse("000123ABC") 'error
            Case 6
                Exit Sub
        End Select
    End Try
End Sub
```

Type Conversion

```
Catch e As System.Exception
    System.Console.WriteLine("error, can't parse expression")
End Try

' check for nothing
System.Console.WriteLine("i=" + If(i.HasValue, i.ToString(), "null value"))
' Or
System.Console.WriteLine("i=" + If(i IsNot Nothing, i.ToString(), "null value"))
' Or
System.Console.WriteLine("i=" +
    If(Global.Microsoft.VisualBasic.Information.IsNothing(i),
        "null value",
        i.ToString()))

If i.HasValue Then
    Microsoft.VisualBasic.MsgBox("i=" & i)
End If

i = Nothing
GoTo start
End Sub
End Module
```

Type Conversion

```
Module Conversions
```

```
Sub conversion()
```

```
    ' TryParse
```

```
    Dim i? As Integer = Nothing ' nullable as indicator
```

```
    Dim outInt As Integer ' storage can't be nullable
```

```
    If Integer.TryParse("true", outInt) Then ' throws no exception, check  
                                              ' for true or false
```

```
        i = outInt
```

```
        System.Console.WriteLine("successfully parsed, i=" & i)
```

```
Else
```

```
    System.Console.WriteLine("error, can't parse expression")
```

```
End If
```

Type Conversion

```
i = Nothing ' reset
```

```
If Integer.TryParse(" 123 ", outInt) Then
```

```
    i = outInt
```

```
    System.Console.WriteLine("successfully parsed, i=" & i)
```

```
Else
```

```
    System.Console.WriteLine("error, can't parse expression")
```

```
End If
```

```
Dim v As Nullable(Of Boolean) ' alternative definition
```

```
Dim outBool As Boolean
```

```
If Boolean.TryParse(" true ", outBool) Then
```

```
    v = outBool
```

```
    System.Console.WriteLine("successfully parsed, v=" & v)
```

```
Else
```

```
    System.Console.WriteLine("error, can't parse expression")
```

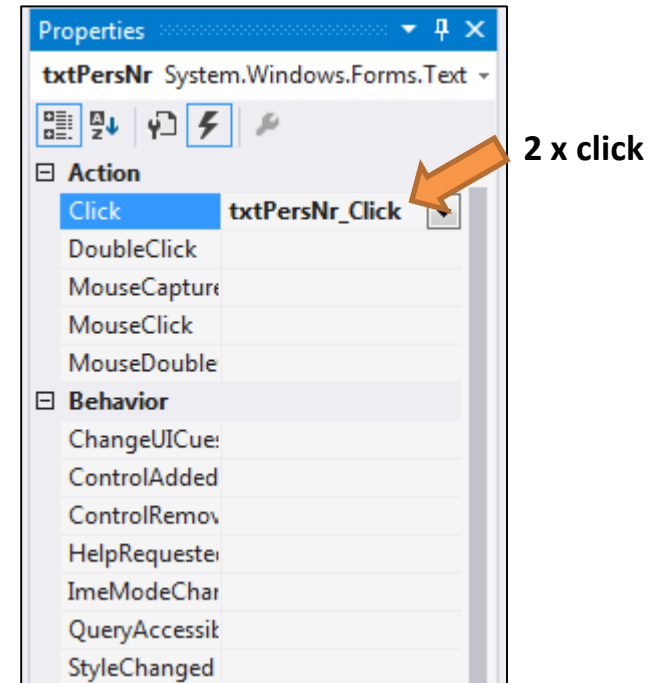
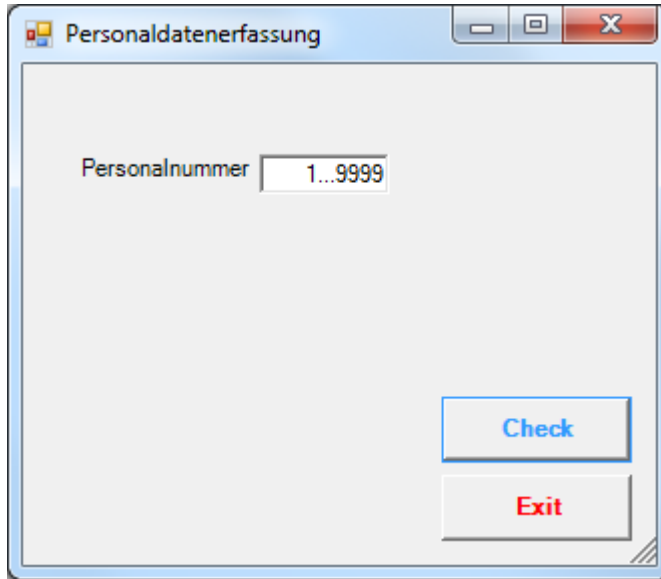
```
End If
```

```
End Sub
```

```
End Module
```

Type Conversion

- Beispielanwendung Formular
 - Prüfroutine Textfeld



- Properties
 - Properties
 - Text, TextAlign, (Name), Font Bold, ForeColor 255;0;0
 - Events
 - Click
 - Inhalt löschen nach erstmaligem Klick
 - Keypress
 - mittels Enter-Taste Check-Routine aufrufen

Type Conversion

```
Public Class frmPersonalDaten
```

```
    Private Sub txtPersNr_Click(sender As Object, e As EventArgs) Handles txtPersNr.Click
        Static changed As Boolean
        If Not changed Then txtPersNr.Clear()
        changed = True
    End Sub
```

```
    Private Sub cmdCheck_Click(sender As Object, e As EventArgs) Handles cmdCheck.Click
        Dim persNr As Integer
        If String.IsNullOrEmpty(txtPersNr.Text) Then
            MsgBox("Personalnummer eintragen")
        ElseIf Not Integer.TryParse(txtPersNr.Text, persNr) Then
            MsgBox("Keine gültige Personalnummer")
            txtPersNr.Clear()
        ElseIf persNr < 1 Or persNr > 9999 Then
            MsgBox("Personalnummer zu klein/gross")
        Else
            MsgBox("accepted")
        End If
        txtPersNr.Focus()
    End Sub
```


Type Conversion

```
Private Sub txtPersNr_KeyPress(  
    sender As Object,  
    e As System.Windows.Forms.KeyPressEventArgs) Handles txtPersNr.KeyPress  
    If e.KeyChar = Microsoft.VisualBasic.ChrW(System.Windows.Forms.Keys.Return) Then  
        cmdCheck_Click(Nothing, Nothing)  
        e.Handled = True  
    End If  
End Sub
```

```
Private Sub cmdExit_Click(sender As Object, e As EventArgs) Handles cmdExit.Click  
    persForm.Close()  
End Sub  
End Class
```

```
Module Personalverwaltung  
    Public persForm As New Global.Intro.frmPersonalDaten  
    Sub main()  
        persForm.ShowDialog()  
    End Sub  
End Module
```

Type Conversion

- Schema Try/Catch/End Try

Try

Catch more special error

Catch less special error

...

Catch general error

End Try

Module Parsing

Sub fileReadWrite()

' -----

' Example Parse file input

' -----

Dim filePath As String = "C:\Users\" + System.Environment.UserName +
 "\Desktop\order_dispatch.txt"

' write file

Const length As Integer = 100

Try

Dim fs As New System.IO.FileStream(filePath, System.IO.FileMode.Create)

' filestream for writing bytes or block of bytes

Dim sw As New System.IO.StreamWriter(fs)

' StreamWriter implements abstract class TextWriter

sw.WriteLine("{0,7} {1,8}", "OrderID", "Quantity")

Dim randomQuant As String = "0123456789AB", quantity

Dim rnd As New System.Random() ' not in loop!

Type Conversion

```
For k As Integer = 1 To length
    quantity = ""
    For m As Byte = 0 To 5 ' generate random quantity
        quantity += randomQuant.ToCharArray()(
            rnd.Next(randomQuant.Length))
            ' length: 12, index: 0..11
    Next
    sw.WriteLine("{0,7:000000} {1,8}", k, quantity)
Next
sw.Close()
Catch e As System.Exception
    System.Console.WriteLine(e.Message)
End Try

' read file
Dim s As String, i, j As Integer
Try
    Dim fr As New System.IO.FileStream(filePath, System.IO.FileMode.Open)
    Dim sr As New System.IO.StreamReader(fr)
```

Type Conversion

```
s = sr.ReadLine
While s IsNot Nothing
    System.Console.WriteLine("s=" + s)
    Try

        ' tokenize line, default delimiter blank
        ' split-method: _XX__XXX_X → _, XX_, _, _, XXX_, X
        i = Integer.Parse(s.Split()(4)) ' jump to catch if parse or split fails
        System.Console.WriteLine("i=" & i)
    Catch e As Exception
        System.Console.WriteLine("splitting or parsing error")
        System.Console.WriteLine(e.Message)
    End Try
```

Type Conversion

```
' Or
Try
    s = s.Split()(4) ' first check split for preparing tryparse
Catch
    System.Console.WriteLine("splitting error")
End Try
If Integer.TryParse(s, j) Then
    System.Console.WriteLine("j=" & j)
Else
    System.Console.WriteLine("parsing error")
End If
s = sr.ReadLine
End While
sr.Close()
Catch e As System.IO.DirectoryNotFoundException
Catch e As System.IO.FileNotFoundException
Catch e As Exception
    System.Console.WriteLine(e.Message)
End Try
```

Type Conversion

```
' Or
For Each item As String In System.IO.File.ReadAllLines(filePath)
    System.Console.WriteLine("item=" + item)
    ' the first dataset contains only 2 delimiters (blanks)
    If item.Split().Length > 2 AndAlso Integer.TryParse(item.Split()(4), j) Then
        ' lazy evaluation, else splitting error
        System.Console.WriteLine(j)
    End If
Next
' automatic close of ReadAllLines-method
End Sub
End Module
```

Type Conversion

```
Sub main()  
    'persForm.ShowDialog()  
    '-----  
    ' Convert  
    '-----  
    Dim str As String = "123", i As Integer  
    i = System.Convert.ToInt32(str)  
  
    Dim nBase As Integer ' Convert with numerical base  
    nBase = 2  
    str = "1001010010111011"  
    i = System.Convert.ToInt32(str, nBase)  
    System.Console.WriteLine("1001010010111011(2)=" + i)  
  
    nBase = 8  
    str = "77770000"  
    i = System.Convert.ToInt32(str, nBase)  
    System.Console.WriteLine("77770000(8)=" + i)  
  
    nBase = 16  
    str = "FFFFFF"  
    i = System.Convert.ToInt32(str, nBase)  
    System.Console.WriteLine("FFFFFF(16)=" + i)  
  
    '=====  
    ' int → string  
    '=====  
    i = 1;  
    ' str = (string)i; ' error, no conversion from int to string  
    i = 1;  
    str = i.ToString();  
    System.Console.WriteLine("str=" + str);  
  
    i = 123;  
    Str = System.Convert.ToString(i);  
    System.Console.WriteLine("str=" + str);
```

Type Conversion

```
Sub main()
    persForm.ShowDialog()
    '-----
    ' Convert
    '-----
    Dim str As String = "123", i As Integer
    i = System.Convert.ToInt32(str)

    Dim nBase As Integer ' Convert with numerical base
    nBase = 2
    str = "1001010010111011"
    i = System.Convert.ToInt32(str, nBase)
    System.Console.WriteLine("1001010010111011(2)=" & i)

    nBase = 8
    str = "77770000"
    i = System.Convert.ToInt32(str, nBase)
    System.Console.WriteLine("77770000(8)=" & i)

    nBase = 16
    str = "FFFFFF"
    i = System.Convert.ToInt32(str, nBase)
    System.Console.WriteLine("FFFFFF(16)=" & i)

    '-----
    ' int → string
    '-----
    i = 1;
    ' str = (string)i; ' error, no conversion from int to string
    i = 1;
    Str = i.ToString();
    System.Console.WriteLine("str=" & Str());

    i = 123;
    Str = System.Convert.ToString(i);
    System.Console.WriteLine("str=" & Str());

    '-----
    ' char → int
    '-----
    Char c = 'A';
    System.Console.WriteLine((Int())c); ' = 65
    System.Console.WriteLine(System.Convert.ToInt32('A'));

    ' int value of char digits
    i = (Int())Char.GetNumericValue('7'); ' returns a double
    System.Console.WriteLine(i); ' = 7

    i = (Int())Char.GetNumericValue("9876543210", 3); ' zero-based index
    System.Console.WriteLine(i); ' = 6

    '-----
    ' int → char
    '-----
    i = 1;
    c = (Char)i;
    c = (Char)i;
    Char Space() = (Char)32;
    System.Console.WriteLine("blank" & Space() & "blank" & Space() & "blank");

    c = System.Convert.ToChar(1);

    '-----
    ' bool → int
    '-----
    bool b = False;
    ' i = (int)b; ' error, no conversion from bool to int
    System.Console.WriteLine(System.Convert.ToInt32(False)); ' = 0
    b = True;
    System.Console.WriteLine(System.Convert.ToInt32(True)); ' = 1
    System.Console.WriteLine(System.Convert.ToInt32(b)); ' = 1

    '-----
    ' int → bool
    '-----
    i = 0;
    ' b = (bool)i; error, no conversion from int to bool
    System.Console.WriteLine("int to bool, 0=" & System.Convert.ToBoolean(0)); ' = False
    i = 2;
    System.Console.WriteLine("int to bool, 2=" & System.Convert.ToBoolean(2)); ' = True
    i = -1;
    System.Console.WriteLine("int to bool, -1=" & System.Convert.ToBoolean(-1)); ' = True
End Sub
End Module
```

```
#If False Then
    '-----
    ' Example Parse file input
    '-----
    String filePath = @"C:\Users\" & System.Environment.UserName & @"\Desktop\order_dispatch.txt";

    /* write file */
    Try
        Int length = 100;
        System.IO.TextWriter tw =
            New System.IO.StreamWriter(
                (New System.IO.FileStream(filePath, System.IO.FileMode.Create)));

        tw.WriteLine("0,7 1,8", "OrderID", "Quantity");
        String randomQuant = "0123456789ABC", quantity;
        System.Random Rnd() = New System.Random(); ' Not in loop!

        For (Int() k = 1; k <= length; k++)
            quantity = "";
```


VBA-Projekt - Aufbau

- Hierarchische Gliederung
 - "TopLevel-Objekt": **VBAProjekt** (\triangleq Dateiname)
 - **Tabellenobjekte**
(jd. Tabelle in der Arbeitsmappe entspricht einem Tabellenobjekt)
 - **Module**
Container für Makros, benutzerdefinierte Prozeduren und Funktionen
 - Gültigkeiten
 - Procedure-level
 - Module-level
 - Global
- | Scope | Prefix | Example |
|--------------------|--------|--------------------|
| Global | g | gstrUserName |
| Module-level | m | mbInCalcInProgress |
| Local to procedure | None | dblVelocity |
- alternativ: g_, m_
 - **Formulare**
Eingabemasken für Datenerfassung, Darstellung von Daten, grafischen Objekten etc.
 - **Klassenmodule**
objektorientierte Modulvariante

ModulName

Deklarationsteil

Optionen: `Option Explicit`

Globale Variablen und Konstanten mit Gültigkeit auf Projektebene

```
Public gFirma
```

```
Global gOrt
```

```
Global Const gPLZ = 33333
```

Variablen und Konstanten mit Gültigkeit auf Modulebene

```
Private mLagerID
```

```
Dim mLagerBestandTotal
```

```
Const mLagerOrt = "Gütersloh"
```

Prozedur- und Funktionsteil

Globale Prozeduren u. Funktionen

```
Public Sub LagerInfo()
```

```
End Sub
```

Prozeduren mit Gültigkeit auf Modulebene

```
Private Sub Disposition()
```

```
End Sub
```

Modul - Aufbau

The screenshot displays the Microsoft Visual Basic for Applications environment for an Excel project named 'ExcelProjektMappe.xlsm'. The interface includes a menu bar, a toolbar, and several panes.

Project Explorer (Left): Shows the 'VBAProject (ExcelProjektMappe.xlsm)' structure. It includes 'Microsoft Excel Objekte' (Tabelle1, Tabelle2, Tabelle3) and 'Module' (Modul1, Modul2).

Properties Window (Bottom Left): Shows the 'Eigenschaften - Modul2' for 'Modul2 Modul'. The 'Name' property is set to 'Modul2'.

Code Windows:

- ExcelProjektMappe.xlsm - Modul1 (Code):** The 'Deklarationsteil' (Declaration Section) contains:

```
Public gLänderCode ' globale Variablen und Konstanten behalten ihren Wert auch nach  
                    Beendigung des Programms (außer bei Zurücksetzung des Projekts)  
  
Private PLZ  
  
Private Sub Postleitzahl()  
    PLZ = 12345  
End Sub  
  
Sub PrintPostleitzahl() ' default public  
    Call Postleitzahl ' initialisieren  
    MsgBox "Postleitzahl=" & PLZ  
End Sub
```
- ExcelProjektMappe.xlsm - Modul2 (Code):** The 'Deklarationsteil' contains:

```
Option Explicit  
  
Sub LänderCode()  
    gLänderCode = "DE"  
End Sub  
  
Sub NeueBestellung()  
    Call Postleitzahl ' nicht bekannt, da private  
    Call PrintPostleitzahl ' ok  
    MsgBox "Postleitzahl=" & PLZ ' Variable nicht definiert (nur bei Option Explicit)  
  
    MsgBox "LänderCode=" & gLänderCode ' ok, aber nicht initialisiert  
    Call LänderCode  
    MsgBox "LänderCode=" & gLänderCode ' ok  
End Sub
```

A yellow box labeled **Deklarationsteil** is positioned between the two code windows, with arrows pointing to the declaration sections of both modules.

Modul - Mehrdeutigkeiten



----- MODUL1 -----

Global gLändercode

Dim mKundenID

Dim mFirma

Dim mAdresse

Sub init()

mKundenID = 222

mFirma = "ABC GmbH"

mAdresse = "50999 Köln"

End Sub

Sub zeigeKundenInfo(typ)

If typ Then

Call init()

Else

Call Modul2.init

End If

Debug.Print "ID ="; mKundenID

Debug.Print "Firma = "; mFirma

Debug.Print "Adresse = "; mAdresse

End Sub

----- MODUL2 -----

Global gLändercode

Dim mKundenID

Dim mFirma

Dim mAdresse

Sub init()

mKundenID = 444

mFirma = "DEF GmbH"

mAdresse = "33333 Gütersloh"

End Sub

----- MODUL3 -----

Sub kundenverwaltung()

Call init

Call Modul1.init

Call Modul2.init

Call zeigeKundenInfo(False)

Modul2.mKundenID = 333

Modul2.gLändercode = "DE"

Debug.Print "Ländercode = "; Modul1.gLändercode

Debug.Print "Ländercode = "; Modul2.gLändercode

Debug.Print "Ländercode = "; gLändercode

End Sub

Makro Verteilung (Deployment) - Verweis

- Verweis
 - Bsp.: Quellmakromappe "ExcelVBA-Projekt.xlsm"
 5. Speichern und schließen
 6. Neue Makromappe
 7. Zum VBA-Editor wechseln, ALT+E+M neues Modul erstellen, ALT+E+P neue Prozedur erstellen "CallTimer"
 8. Menü Extras/Verweise, Durchsuchen, Dateityp: *.xlsm
 9. Quellmakromappe selektieren ("ExcelVBA-Projekt.xlsm")
 10. Verfügbare Verweise prüfen, ob Kontrollkästchen aktiviert

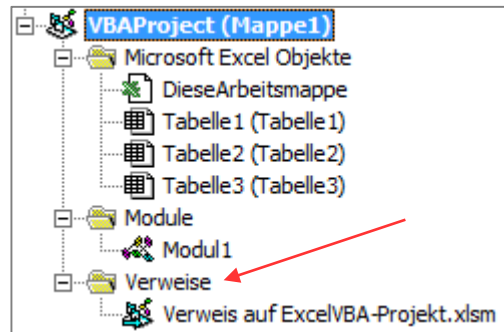


Makro Verteilung (Deployment) - Verweis

■ Verweis

■ Bsp.: Quellmakromappe "ExcelVBA-Projekt.xlsm"

11. Im Projektextplorer den Ordner "Verweise" prüfen



12. Prüfen ob "TimerCode" kennwortgeschützt ist

13. In der Prozedur "CallTimer" die Prozedur "Timer" aufrufen

```
Public Sub CallTimer()  
    Call TimerCode.Timer Call Timer würde Fehler liefern  
End Sub
```

14. In der Prozedur "CallTimer" die Prozedur "TimerSerial" aufrufen

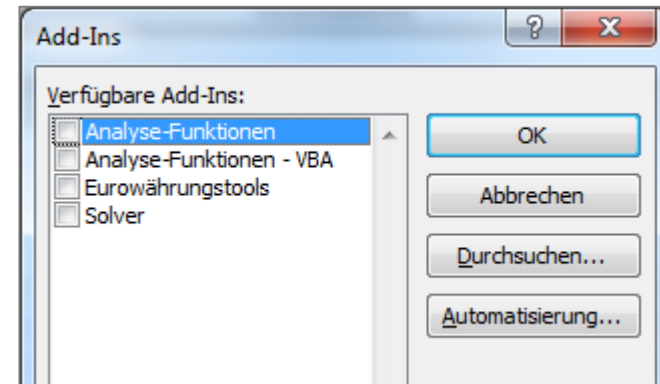
```
Public Sub CallTimer()  
    Call TimerSerial  
End Sub
```

Makro Verteilung (Deployment) - Add-In

- Variante Add-In

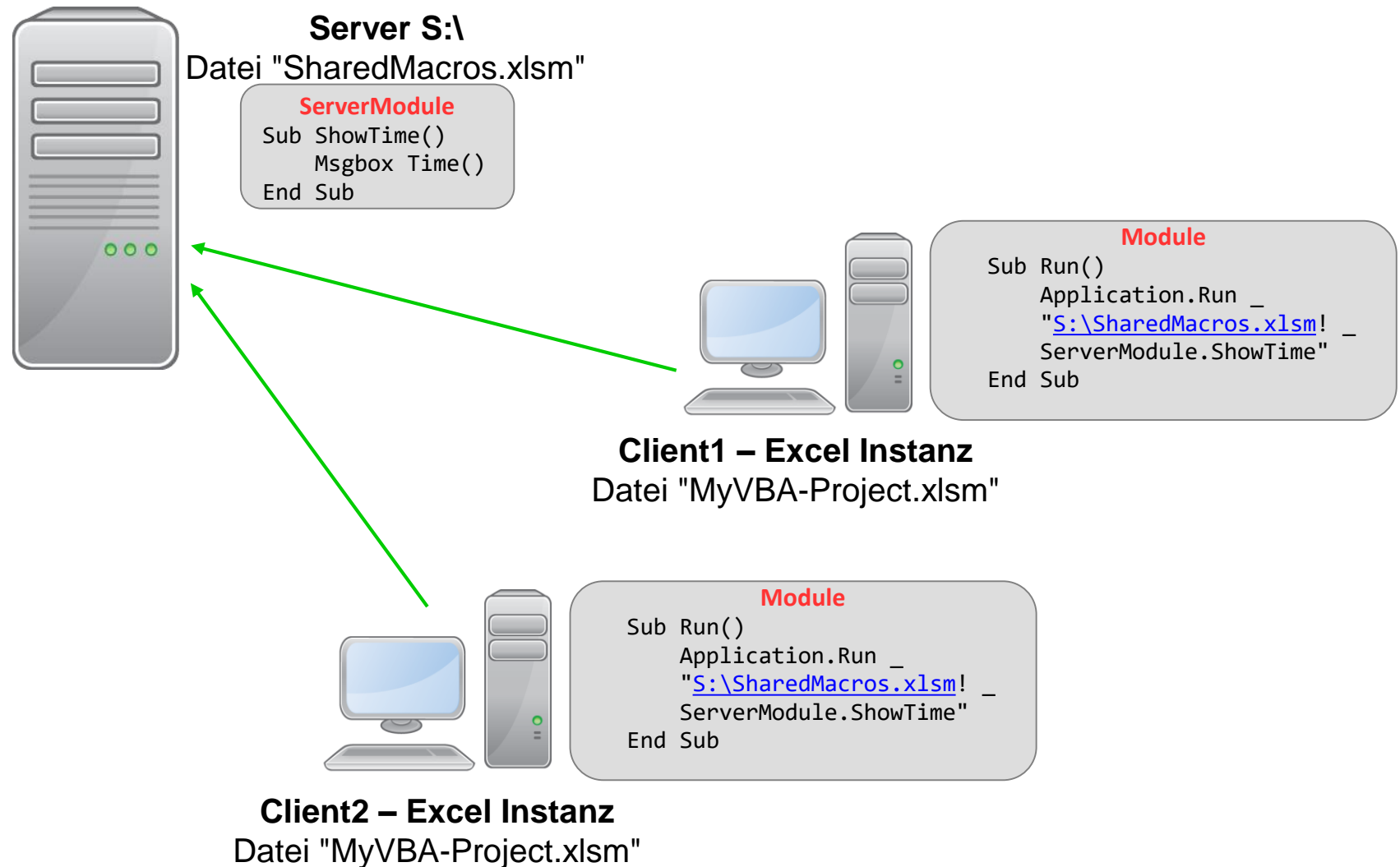
- Permanent eingebunden
- Menü Entwicklertools/Add-Ins/Add-Ins
bzw. Menü Datei/Optionen/Add-Ins
- Anwenderseitig aufrufbar via Funktionsassistent
- Programmierseitig nur über Fernbezug `Application.Run`
- Bsp. Quellcodemappe als Add-In-Datei speichern

1. Neue Mappe "Systeminfo"
 2. Zum VBA-Editor wechseln, ALT+E+M neues Modul erstellen
 3. Funktion "SystemInformationStart" als Hilfsfunktion,
ALT+E+P+Function neue Funktion hinzufügen
- ```
Public Function SystemInformationStart()
 Call SystemInformation
 SystemInformationStart = 1
End Function
```



# Makro Verteilung (Deployment) - Fernbezug

- Bsp.: Gemeinsam genutzte Quellcode-Datei auf Server





# Message Box

| ASCII   | Konstante | Englisch        | Deutsch        |
|---------|-----------|-----------------|----------------|
| Chr(9)  | vbTab     | Tabulator       | Tabulator      |
| Chr(10) | vbLf      | Line Feed       | Zeilenvorschub |
| Chr(13) | vbCr      | Carriage Return | Wagenrücklauf  |

| Konstante          | Wert    | Beschreibung                |
|--------------------|---------|-----------------------------|
| vbMsgBoxRight      | 524288  | Rechtsbündiger Text         |
| vbMsgBoxRtlReading | 1048576 | Von rechts nach links lesen |

# Enum

- Aufzählungstyp ("enumeration")
- Datentyp zur Definition einer Menge an vordefinierten Konstanten
- Verbesserung der Lesbarkeit, Prüfung durch Compiler
- typsicher

```
public enum Day
 SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
 THURSDAY, FRIDAY, SATURDAY
end enum
```

```
Day day; //defines a variable day of enumerated type Day
```

```
day = Day.SATURDAY; //set
```

# Exceptions

```
On Error GoTo ErrHandler:
Worksheets("NewSheet").Activate
Exit Sub
```

```
ErrHandler:
If Err.Number = 9 Then
 sheet does not exist, so create it
 Worksheets.Add.Name = "NewSheet"
 go back to the line of code that caused the
 problem
 Resume
End If
```

# Exceptions

```
On Error Goto 0
On Error Resume Next
On Error Goto <label>:
```

```
On Error Resume Next
N = 1 / 0
If Err.Number <> 0 Then
 Debug.print Err.Number
 N = 1
End If
```

```
Exit sub/function
```

```
Label:
 Resume Next / exit
```

```
On Error GoTo Err1:
Debug.Print 1 / 0
more code
Err1:
On Error GoTo Err2:
Debug.Print 1 / 0
more code
Err2:
```

# Klassenmodule

- **Klassenbezeichner**, ClassID
- **Attribute**, Eigenschaften, Properties
- **Methoden**, Operationen

| Artikel                                                                                          |
|--------------------------------------------------------------------------------------------------|
| Bezeichnung: String<br>Netto-Preis: double<br>Mehrwertsteuersatz: double<br>Brutto-Preis: double |
| Brutto_berechnen()                                                                               |

- **Sichtbarkeit** von Operationen und Attributen
  - + ***public***, unbeschränkter Zugriff
  - ***private***, nur die Klasse selbst kann es sehen

# Pause



# Mahlzeit!



 60