9.8 Zeichenketten-Funktionen

Das Arsenal von Funktionen zur Manipulation von Zeichenketten (Strings) in Gambas ist vielfältig. Gambas besitzt zwei Sets von Funktionen zur String-Manipulation. Ein Set für die Manipulation von ASCII-Strings und ein anderes, das mit UTF-8-Strings umgehen kann.

Folgende Hinweise sind zu beachten, wenn Sie die Zeichenketten- oder String-Funktionen der beiden Sets einsetzen:

- Die Begriffe Zeichenkette und String werden synonym verwendet.
- Viele String-Funktionen können nur mit ASCII-Strings umgehen.
- Um UTF-8-Zeichenketten zu manipulieren, verwenden Sie die gleichwertigen Methoden der (statischen) String-Klasse, die in der folgenden Tabelle unter *UTF8* angegeben werden.
- Wenn Sie eine Zeichenfolge bearbeiten, die übersetzt werden soll, dann müssen Sie die UTF-8-Funktionen der String-Klasse benutzen.
- Die (UTF-8-)String-Klasse ist wohl zu unterscheiden vom nativen Datentyp String.
- Wenn keine UTF-8-äquivalente String-Funktion angegeben wird, dann manipuliert die String-Funktion sowohl ASCII-Strings als auch UTF-8-Zeichenketten.
- Seien Sie vorsichtig! Gambas verwendet intern den UTF-8-Zeichensatz, so dass ein Zeichen-Code größer als 128 nicht die gleiche Bedeutung hat wie im Zeichensatz *ISO 8859-1*.

9.8.1 Übersicht zu den String-Funktionen

Funktion	UTF8	Beschreibung
Asc (String [, Position])	String.Code	Gibt den ASCII-Code des Zeichens in einem String an der vorgegebenen Position zurück. Ist der optionale Parameter 'Position' nicht angegeben, dann wird der ASCII-Code des *ersten* Zeichens als Funktionswert zurückgegeben.
Chr (Code AS Integer) Chr$ (Code AS Integer)	String.Chr	Gibt das Zeichen zu einem ASCII-Code zurück. Diese Funktion arbeitet nur mit ASCII-Zeichen. Wenn Sie Nicht-ASCII-Zeichen, wie zum Beispiel deutsche Umlaute erzeugen müssen, so müssen Sie die Funktion String.Chr verwenden.
Comp (String1, String2 [, Mode AS Integer])	String.Comp	Vergleicht zwei Strings und gibt folgende Funktionswerte zurück: -1, wenn String1 kleiner als String2 ist; 0, wenn die beiden Strings gleich sind und eine +1, wenn String1 größer als String2 ist. Über das optionale Argument 'Mode' können Sie die Vergleichs-Methode einstellen.
LCase$ (String) AS String Lower$ (String) AS String	String.LCase	Gibt den String in kleinen Buchstaben zurück. Diese Funktion arbeitet nicht mit UTF-8-Strings! Für diesen Fall nutzen Sie mit Erfolg die äquivalente Funktion String.LCase.
UCase$ (String) Upper$ (String)	String.UCase	Gibt den String in großen Buchstaben zurück. Diese Funktion arbeitet nicht mit UTF-8-Strings! In diesem Fall setzen Sie die Funktion String.UCase ein.
Len (String)	String.Len	Len() gibt die Länge eines Strings zurück. Da jedes ASCII-Zeichen ein Byte groß ist, entspricht dies genau der Anzahl von Bytes im String. Da UTF-8-Zeichen wie ä, ö, ü, ß zwei oder mehr Bytes benötigen, greifen Sie zur Ermittlung der Anzahl der 'Zeichen' im String zu String.Len(). Die Anzahl belegter Bytes kann trotzdem über Len() ermittelt werden.
Left$ (String AS String [, k AS Integer]) Left (String AS String [, k AS Integer])	String.Left	Gibt die ersten k Zeichen des Strings zurück. Wenn k nicht nicht angegeben ist, wird nur das erste Zeichen des Strings zurück gegeben. Wenn k negativ ist, dann werden die -k letzten Zeichen nicht beachtet.
Mid (String , iStart AS Integer [, iLength AS Integer]) Mid$ (String , iStart AS Integer [, iLength AS Integer])	String.Mid	Gibt einen String als Teil-String zurück, der bei Position iStart beginnt und die Länge iLength hat. Wenn iLength (optional) nicht angegeben ist, wird alles von der Position iStart bis zum Ende von String zurückgegeben. Bei negativem Wert von iLength wird der String von iStart ohne die letzten iLength Zeichen zurück gegeben.
Right (String [, k AS Integer])	String.Right	Gibt die letzten k Zeichen als Teil-String von String zurück.

Funktion	UTF8	Beschreibung
Right$ (String [, k AS Integer])		Wenn k nicht angegeben, wird das letzte Zeichen von String zurück gegeben. Bei negativem Wert von k wird der String ohne die ersten -k Zeichen zurück gegeben.
InStr (String, Substring AS String [, Start AS Integer , Comparison AS Integer])	String.InStr	Gibt die Position des 1. Auftretens von 'Substring' in String zurück oder 0, wenn der 'Substring' *nicht* gefunden wurde. Wird das optionale Argument 'Start' angegeben, dann beginnt die Suche ab der Start-Position. Die Vergleichsmethode kann eine der folgenden sein: gb.Binary (Standard-Voreinstellung) oder gb.IgnoreCase für einen case-insensitiven Vergleich.
RInStr (String, Substring AS String [, Start AS Integer , Comparison AS Integer])	String.RInStr	Gibt die Position des letzten Vorkommens von 'Substring' im String zurück. Ist 'Start' angegeben, so beginnt die Suche an der Position 'Start'. Die Vergleichsmethode kann eine der folgenden sein: gb.Binary (Standard-Voreinstellung) oder gb.IgnoreCase für einen case-insensitiven Vergleich. Wenn der Teilstring 'Substring' nicht gefunden wird, gibt die Funktion 0 zurück.
Scan (String, Pattern AS String)	-	Gleicht einen String mit Pattern ab und gibt ein Array aller Strings zurück, die mit dem * Zeichen verknüpft sind. Pattern ist ein Muster, wie es im Kapitel → 8.3.3 Operator LIKE beschrieben ist.
Space$ (k AS Integer)	-	Gibt einen String aus k Leerzeichen zurück.
Replace (String , Pattern , ReplaceString [, Comparison]) Replace$ (String , Pattern , ReplaceString [, Comparison])	-	Ersetzt *jedes* Vorkommen von 'Pattern' durch 'ReplaceString' und gibt das Ergebnis zurück. Wenn String Null ist, dann wird ein Null-String zurück gegeben. Wenn 'Pattern' Null ist, so wird der Original-String zurückgegeben. Die Vergleichsmethode (optional) kann eine der folgenden sein: gb.Binary (Standard-Voreinstellung) oder gb.IgnoreCase für einen case-insensitiven Vergleich.
Base64 (String) Base64$ (String)	-	Kodiert einen String in das Base64-Format.
UnBase64 (Base64String) UnBase64$ (Base64String)	-	Dekodiert einen String im Base64-Format.
LTrim (String) LTrim$ (String)	-	Entfernt Leerzeichen *am Anfang* des Strings. Als Leerzeichen wird jedes Zeichen mit einem ASCII-Code kleiner als 32 aufgefasst. Die Funktion ist so optimiert, dass Leerzeichen *im String* nicht beachtet werden.
Trim$ (String) Trim (String)	-	Entfernt Leerzeichen am Anfang und am Ende des Strings.
RTrim (String) AS String RTrim$ (String) AS String	-	Entfernt Leerzeichen am Ende des Strings.
Html (String)	-	Zeichnet einen String so aus, dass er sicher in eine HTML-Seite eingefügt werden kann. Sie müssen in Ihrer HTML-Seite die UTF-8-Codierung verwenden. Mit der Angabe des Charset-Parameters <meta charset="utf-8" /> im Content-Type-Header sind Sie stets auf der sicheren Seite.
String$ (k AS Integer , Pattern AS String)	-	Gibt einen String zurück, der genau *k* mal den Pattern-String enthält.
Subst$ (Pattern , ReplaceString [, ReplaceString ...]) Subst (Pattern , ReplaceString [, ReplaceString ...])	-	Ersetzt die Teil-Strings &1, &2, ... in einem Muster-String durch den ersten, zweiten und die nachfolgenden Replace-Strings und gibt den resultierenden String zurück. Wenn der Muster-String Null ist, dann wird ein Null-String zurückgegeben.
Split (String [, Separators , Escape , IgnoreVoid , KeepEscape])	-	Zerlegt String in Teil-Strings. Die Trennstellen werden durch Separator-Zeichen (aus einer Liste von Separatoren) gekennzeichnet. Die Teil-Strings können durch Escape-Zeichen begrenzt sein. Diese Funktion liefert ein *String-Array* zurück, das jeden er-

Funktion	UTF8	Beschreibung
		kannten Teilstring enthält. Die Separator-Zeichen werden nicht mit zurück gegeben.
Quote (String) Quote$ (String)	-	Setzt den String unter Nutzung der Gambas-String-Syntax in Anführungszeichen.
Unquote (Quoted string) Unquote$ (Quoted string)	-	Entfernt *Anführungszeichen-Paare* um den String.

Tabelle 9.8.1.1 : Übersicht zu den String-Funktionen

9.8.2 Hinweise zu ausgewählten String-Funktionen

9.8.2.1 Funktion Scan (String, Pattern AS String)

Die Scan-Funktion teilt einen String mit Hilfe eines regulären Ausdrucks (LIKE-Format) auf. Im folgenden Beispiel wird der Befehl 'df' (disk free) ohne Parameter benutzt und liefert Informationen zum Speicherplatz auf allen gemounteten Dateisystemen:

```
Dim sResult, sLine, sElementAs String
Shell "df" To sResult
For Each sLine In Split(sResult, "\n")
  For Each sElement In Scan(sLine, "* * * * *")
    Print sElement; " | ";
  Next ' sElement
  Print
Next ' sLine
```

Ausgabe in der Konsole der IDE:

```
Dateisystem | 1K-Blöcke | Benutzt   | Verfügbar | Verw% Eingehängt auf |
|-------------------------------------------------------------------------|
/dev/sda6   | 50395844  | 10974228 | 36861616  | 23%   /              |
udev        | 4037720   | 4        | 4037716   | 1%    /dev           |
tmpfs       | 809308    | 944      | 808364    | 1%    /run           |
none        | 5120      | 0        | 5120      | 0%    /run/lock      |
none        | 4046520   | 156      | 4046364   | 1%    /run/shm       |
/dev/sda7   | 218656644 | 57277676 | 150271848 | 28%   /home          |
```

9.8.2.2 Funktion Subst(Pattern , ReplaceString [, ReplaceString ...]))

Ruft man ein Programm mit der Shell-Instruktion auf, kann die Kommandozeile schnell kompliziert werden, wenn man viele Argumente übergibt, die ihrerseits Variablen im Gambas-Prozess sind. Wir betrachten als Beispiel das Programm "tar", wie es zum Beispiel in den Patch-Dialogen der Gambas-IDE aufgerufen wird.

Der "Patch-generieren"-Dialog erfragt den Pfad eines alten Quelltext-Archivs, von dem aus ein Patch zum Zustand des aktuellen Projekts generiert werden soll. Damit das "patch"-Programm auf den Quellen arbeiten kann, muss das Archiv entpackt werden. Dies wird mittels "tar" so umgesetzt:

```
Shell "tar -" & sType & "xf "  & Shell$(sOldSource) & " -C " &sOld &  " --strip-components=1"
```

Wenn Sie statt der &-Verkettungen nun die Funktion *Subst$()* benutzen, werden Sie der Zeile auch noch später ihre Bedeutung ansehen:

```
Shell Subst$("tar -&1xf &2 -C &3 --strip-components=1", sType, Shell$(sOldSource), sOld)
```

Die Funktion *Subst$()* ist das Mittel der Wahl, wenn Sie Strings in Ihrem übersetzbaren Projekt aus Variablen und String-Konstanten zusammensetzen, wie beispielsweise in diesem Fall:

```
Message(("Der Server ") & sServer & (" meldet: alles in Ordnung!"))
```

Durch die getrennten String-Konstanten und die Variable *sServer* in der Mitte legen Sie die Syntax des Satzes fest. Was ist aber, wenn jemand Ihr Programm in eine Sprache übersetzen möchte, in der das Subjekt "Der Server X" am Ende des Satzes steht? Da *sServer* immer an das Ende des ersten Strings verkettet wird, müsste der Übersetzer alle Informationen in der Übersetzung des ersten Strings unter-

bringen und den zweiten String in den leeren String übersetzen. Mit dem Einsatz von *Subst$()* kann man diese Situation vermeiden:

```
Message(Subst$(("Der Server &1 meldet: alles in Ordnung!"), sServer))
```

Hier kann der Übersetzer selbst die für seine Sprache natürliche Syntax wählen, in dem er den Platzhalter &1 geeignet verschiebt.

Beachten Sie: Wenn Sie mehr als 9 Teil-Strings verwenden, dann müssen Sie zum Beispiel statt &10 den (vor-definierten) Teil-String in geschweifte Klammern {&10} setzen.

9.8.2.3 Funktion Split(String [, Separators , Escape , IgnoreVoid , KeepEscape])

Mit Hilfe der Funktion *Split(String [, Separators , Escape , IgnoreVoid , KeepEscape])* können Sie eine Zeichenkette in einzelne Zeichenketten zerlegen, wenn die Teil-Strings durch Separatoren getrennt werden und (optional) durch Escape-Zeichen begrenzt sind. Diese Funktion liefert ein String-Array zurück, das jeden erkannten Teil-String enthält.

- *String* ist der String, der an allen angegebenen Trennzeichen aus der Liste der Trennzeichen in Teil-Strings aufgeteilt wird.
- *Separators* ist eine komma-separierte Liste der Trennzeichen. Standard ist das 'Komma' als Trennzeichen.
- *Escape* ist ein Escape-Zeichen. Es dürfen nur 1-Byte-ASCII-Zeichen verwendet werden. Standardmäßig wird kein Escape-Zeichen angegeben. Maximal können 2 Escape-Zeichen verwendet werden. Werden 2 Escape-Zeichen eingesetzt, dann ist das erste das Start-Escape-Zeichen und das zweite Zeichen das Ende-Escape-Zeichen.
- Alle Trennzeichen, die zwischen zwei Escape-Zeichen eingeschlossen sind, werden beim Splitten ignoriert. Das ermöglicht zum Beispiel in einem Teil-String eine Zahl mit einem Komma zu verwenden, auch wenn das Trennzeichen ein Komma ist!
- *IgnoreVoids* ist vom Daten-Typ Boolean. Die Standard-Vorgabe ist False. Ist der Wert True, dann werden leere Teil-Strings beim Splitten ignoriert.
- *KeepEscape* ist auch vom Typ Boolean. Die Standard-Vorgabe ist False. Ist der Wert True, so werden die Escape-Zeichen im Teil-String mit zurückgegeben.
- Die Separator-Zeichen werden im Teil-String nicht mit zurückgegeben. Die Escape-Zeichen werden nur dann im Teil-String nicht mit zurückgegeben, wenn der Wert von KeepEscape mit False vorgegeben ist.
- Wenn der String selbst Escape-Zeichen enthält, dann müssen diese Escape-Zeichen dupliziert werden.
- Sie können die Split-Funktion nicht verwenden um eine Zeichenfolge aus Nicht-ASCII-Zeichen zu splitten.

Dieser Quelltext:

```
sInput = "(12,3 )*()+(789),(    def)*(45,76)+     (a b c)"

Print "String  : (12,3 )*()+(789),(    def)*(45,76)+     (a b c)"
Print "Funktion: Split(sInput, \"+,*\", \"()\", IgnoreVoid, KeepEscape)"
Print String$(61, "-") & gb.NewLine
Print "IgnoreVoid = False, KeepEscape = False"
aStringFieldArray = Split(sInput, "+,*", "()", False, False)
For k = 0 To aStringFieldArray.Max
   Print "k = "; k; "  ~>  "; Trim(aStringFieldArray[k])
Next
Print "IgnoreVoid = True, KeepEscape = False"
aStringFieldArray = Split(sInput, "+,*", "()", True, False)
For k = 0 To aStringFieldArray.Max
  Print "k = "; k; "  ~>  "; Trim(aStringFieldArray[k])
Next
Print "IgnoreVoid = False, KeepEscape = True"
aStringFieldArray = Split(sInput, "+,*", "()", False, True)
For k = 0 To aStringFieldArray.Max
  Print "k = "; k; "  ~>  "; Trim(aStringFieldArray[k])
Next
Print "IgnoreVoid = True, KeepEscape = True"
aStringFieldArray = Split(sInput, "+,*", "()", True, True)
For k = 0 To aStringFieldArray.Max
  Print "k = "; k; "  ~>  "; Trim(aStringFieldArray[k])
Next
```

produziert diese Ausgaben in der Konsole der Gambas-IDE:

```
IgnoreVoid = False, KeepEscape = False
k = 0  ~>  12,3
k = 1  ~>
k = 2  ~>  789
k = 3  ~>  def
k = 4  ~>  45,76
k = 5  ~>  a b c

IgnoreVoid = True, KeepEscape = False
k = 0  ~>  12,3
k = 1  ~>  789
k = 2  ~>  def
k = 3  ~>  45,76
k = 4  ~>  a b c

IgnoreVoid = False, KeepEscape = True
k = 0  ~>  (12,3 )
k = 1  ~>  ()
k = 2  ~>  (789)
k = 3  ~>  (    def)
k = 4  ~>  (45,76)
k = 5  ~>  (a b c)

IgnoreVoid = True, KeepEscape = True
k = 0  ~>  (12,3 )
k = 1  ~>  ()
k = 2  ~>  (789)
k = 3  ~>  (    def)
k = 4  ~>  (45,76)
k = 5  ~>  (a b c)
```

9.8.3 Beispiele

Der nächste Quelltext-Ausschnitt gibt Einsicht in die Verwendung ausgewählter String-Funktionen:

```
Public Sub btnStringManipulationen_Click()
  Dim iCount, k As Integer
  Dim sCommand As String = "Open"
  Dim sZK As String = "Heute ist der "
  Dim sLine, sInput, sText, sMessage1, sMessage2 As String
  Dim aStringFieldArray, aDateArray As String[]
  Dim aVariantFieldArray, vElement As Variant[]
  Dim vVariant As Variant
  Dim aCSV As New Variant[][]
  Dim hFile As File

  Print "ASC(\"Tobias\") = "; Asc("Tobias"), "ASC(\"Tobias\",2) = "; Asc("Tobias", 2)
  Print "ASC(\"ö\") = "; Asc("ö")
  Print "STRING.CODE(\"Ärger\") = "; String.Code("Ärger"); " ("; Hex(String.Code("Ärger")); " hex)"
  Print "BASE64(\"Tobias\") = "; Base64("Tobias"), "UNBASE64(\"VG9iaWFz\") = "; UnBase64("VG9iaWFz")
  Print "CHR(66) = "; Chr(66), "STRING.CHR(246) = "; String.Chr(246)
  Print "Zeile 1"; Chr(10); "Zeile 2"
  If Comp(Upper(sCommand), "OPEN") = 0 Then Print "Die Datei wird geöffnet!"
  Print "Html(\"Ärger im Büro...\")  "; Html("Ärger im Büro...")

  Print "Der String 'bas' wurde an Pos "; InStr("Gambas als Basic-Dialekt.", "bas"); " erstmalig gefunden."
  Print IIf(Comp(Lower(sCommand), "open") = 0, "Die Datei wird geöffnet!", "Fehler!")
  Print "Lower(\"Ärger?\") = "; Lower("Ärger?"); "  String.Lower(\"Ärger!\") = "; String.Lower("Ärger!")
  If Comp(Upper(sCommand), "OPEN") = 0 Then Print "Die Datei wird geöffnet!"
  Print Left("Hans-Joachim", 4), Left("Hans-Joachim"), Left("Hans-Joachim", -8)
  Print Mid("Gambas-Buch", 8), Mid("Gambas-Buch", 4, 5), Mid("Gambas-Buch", 2, -5)
  Mid$(sZK, Len(sZK) + 1) = Format(Now(), "d. mmmm yyyy!")
  Print sZK; gb.NewLine

  sInput = "-3.88+4.275i"
  With Scan(sInput, "*+*i")
    Print "    Realteil =", .[0]; " (z = "; sInput; ")"
    Print "Imaginärteil = ", .[1]; " (z = "; sInput; ")"
  End With
  Print
  Print "Acht Leerzeichen zwischen < und >: "; "<"; Space(8); ">"
  Print "Original: "; "\"   Das ist - nicht nur bei Gambas - Standard!   \""
  Print "LTRIM: "; "<"; LTrim("   Das ist - nicht nur bei Gambas - Standard!   "); ">"
  Print "TRIM: "; "<"; Trim("   Das ist - nicht nur bei Gambas - Standard!   "); ">"
  Print "RTRIM: "; "<"; RTrim("   Das ist - nicht nur bei Gambas - Standard!   "); ">"
  Print "Byte-Anzahl 'Gambas' = "; Len("Gambas"); "  Byte-Anzahl 'Äöü' = "; Len("Äöü")
```

```
  Print "Position von 'bas' in 'Gambas is basic' mit RInstr(..) = "; RInStr("Gambas is basic", "bas")
  Print "Pos von 'bas' in 'Gambas is basic' mit RInstr(..) ab Pos 10="; RInStr("Gambas is basic","bas", 10)

  sText = "Gambas is basic"
  sMessage1 = "Text 'BAS' in '" & sText & "' gefunden!"
  sMessage2 = "Text 'BAS' in '" & sText & "' nicht gefunden!"
  Print IIf(RInStr(sText, "BAS", gb.IgnoreCase), sMessage1, sMessage2)
  Print "Ersetzt + in '+123.45 durch NULL-String: "; Replace("+123.45", "+", "")
  Print String$(11, "-----")
  Print Subst("Heute ist &1. &2", Format(Now(), "dddd"), "Ist das nicht schön?")
  Print "Zeilen-Anzahl (TextArea) = "; Split(TextArea1.Text, gb.NewLine).Count
  Wait 3
  Print
  Print Quote("This is a 'multi-Line' string." & gb.NewLine)

  TextArea1.Clear
  hFile = Open "example.csv" For Input
  While Not Eof(hFile)
    Line Input #hFile, sLine
    TextArea1.Insert(sLine & gb.NewLine)
    aStringFieldArray = New String[] ' Für jede Zeile wird ein neues Array genutzt
    aStringFieldArray = Split(sLine, ",", "\"", False, False)

    aVariantFieldArray = New Variant[]
    For iCount = 0 To aStringFieldArray.Max
      If IsFloat(aStringFieldArray[iCount]) Then
         aStringFieldArray[iCount] = Replace(aStringFieldArray[iCount], ",", ".")
         aVariantFieldArray.Add(CFloat(aStringFieldArray[iCount]))
      Else If IsBoolean(aStringFieldArray[iCount]) Then
         aVariantFieldArray.Add(aStringFieldArray[iCount])
      Else If IsDate(aStringFieldArray[iCount]) Then
         aDateArray = Split(aStringFieldArray[iCount], ".")
         aVariantFieldArray.Add(Date(aDateArray[2], aDateArray[1], aDateArray[0]))
       ' aVariantFieldArray.Add(Val(aStringFieldArray[iCount])) ' Schnelle Alternative
      Else
         aVariantFieldArray.Add(aStringFieldArray[iCount])
      Endif
    Next
    aCSV.Add(aVariantFieldArray)
   Wend

   Print
   For Each vElement In aCSV
        For Each vVariant In vElement
            If IsDate(Str(vVariant)) Then
               Print Format(vVariant, "dd.mm.yyyy")
            Else
               Print vVariant,
            Endif
        Next ' vVariant
   Next ' sElement
End ' StringManipulationen
```

Das ist der Inhalt der csv-Datei *example.csv*:

```
"8,03","False","ERSTER","24.06.2014"
"58,1","True","Peter","23.02.2015"
"66,3","True","Paula","08.08.2015"
"83,1","False","Anna","14.01.2015"
"13,8","True","Thomas","04.02.2014"
"-8,9","False","Peter","07.05.2015"
"8,95","True","Paul","21.09.2013"
"52,5","True","Peter","23.02.2015"
"16,7","True","LETZTER","08.04.2015"
```

Hier sehen Sie die erzeugten Ausgaben:

```
ASC("Tobias") = 84      ASC("Tobias",2) = 111
ASC("ö") = 195
STRING.CODE("Ärger") = 196 (C4 hex)
BASE64("Tobias") = VG9iaWFz     UNBASE64("VG9iaWFz") = Tobias
CHR(66) = B     STRING.CHR(246) = ö
Zeile 1
Zeile 2
Die Datei wird geöffnet!
Html("Ärger im Büro...")  Ärger im Büro...
Der Suchstring 'bas' wurde an der Position 4 erstmalig gefunden.
Die Datei wird geöffnet!
Lower("Ärger?") = Ärger?  String.Lower("Ärger!") = ärger!
Die Datei wird geöffnet!
Hans    H       Hans
Buch    bas-B   ambas
```

```
Heute ist der 29. Juni 2014!

    Realteil =  -3.88 (z = -3.88+4.275i)
Imaginärteil =  4.275 (z = -3.88+4.275i)

Acht Leerzeichen zwischen < und >: <        >
Original: "   Das ist - nicht nur bei Gambas - Standard!   "
LTRIM: <Das ist - nicht nur bei Gambas - Standard!   >
TRIM: <Das ist - nicht nur bei Gambas - Standard!>
RTRIM: <   Das ist - nicht nur bei Gambas - Standard!>
Byte-Anzahl 'Gambas' = 6  Byte-Anzahl 'ÄöÜ' = 6
Position von 'bas' in 'Gambas is basic' mit RInstr(..) = 11
Position von 'bas' in 'Gambas is basic' mit RInstr(..) ab Position 10 = 4
Text 'BAS' in 'Gambas is basic' nicht gefunden!
Ersetzt + in '+123.45 durch NULL-String: 123.45
------------------------------------------------------
Heute ist Sonntag. Ist das nicht schön?
Zeilen-Anzahl (TextArea) = 9

"This is a 'multi-Line' string.\n"

8,03    False   ERSTER  24.06.2014
58,1    True    Peter   23.02.2015
66,3    True    Paula   08.08.2015
83,1    False   Anna    14.01.2015
13,8    True    Thomas  04.02.2014
-8,9    False   Peter   07.05.2015
8,95    True    Paul    21.09.2013
52,5    True    Peter   23.02.2015
16,7    True    LETZTER 08.04.2015
```