

# Inhaltsverzeichnis

<b>1</b>	<b>Einstieg</b>	<b>17</b>
1.1	Was ist Gambas? . . . . .	18
1.2	Warum Gambas? . . . . .	19
1.2.1	Zitate . . . . .	19
1.3	Wie kann man Gambas lernen? . . . . .	19
1.4	Gambas herunterladen . . . . .	20
1.5	Gambas Installation . . . . .	20
1.6	Ausführliche Beschreibung . . . . .	20
1.6.1	Download . . . . .	21
1.6.2	Entpacken . . . . .	22
1.6.3	Konfiguration überprüfen . . . . .	22
1.6.4	Kompilation mit make . . . . .	23
1.6.5	Installation als root make install . . . . .	23
1.7	Starten von Gambas . . . . .	24
1.8	Hallo Welt! . . . . .	27
1.8.1	Legen wir zunächst ein neues Projekt an: . . . . .	28
1.8.2	Nun erstellen Sie ein neues Formular: . . . . .	28
1.8.3	Jetzt wird programmiert: . . . . .	29
1.8.4	Es geht los . . . . .	29
1.9	Zweites Programm in Gambas . . . . .	31
1.10	Drittes Programm: Programme im Terminalfenster ohne Grafik . . . . .	32
1.11	Wie kann man den Programmcode aus diesem Kurs kopieren? . . . . .	34
1.12	Ausführbare Datei erstellen . . . . .	35
1.13	Das Programm auf dem Desktop platzieren . . . . .	36
1.14	Dem Programm ein eigenes Icon (Programmsymbol) verpassen . . . . .	37
1.15	Programmstart . . . . .	37
1.15.1	Besonderheiten . . . . .	38
1.16	Programmende . . . . .	39
1.16.1	Programmende mit ME.Close . . . . .	39
1.16.2	Ende mit der ESC Taste . . . . .	40

---

1.16.3	Probleme . . . . .	40
1.17	Programm abwürgen . . . . .	40
<b>2</b>	<b>Hilfe</b>	<b>43</b>
2.1	Mitgelieferte Hilfe-Funktion . . . . .	43
2.2	Hilfe im Internet . . . . .	46
2.2.1	gambasdoc.org . . . . .	46
2.2.2	Diskussionsforen . . . . .	47
2.2.3	Gambas Wikibook FAQ . . . . .	47
2.2.4	Gambas Index . . . . .	47
2.3	Offene Fragen . . . . .	47
2.4	Cursor an das Ende eines Textes? . . . . .	48
2.5	Wie kann ich mehrere Variablen in eine Datei schreiben und bei Bedarf (z.B. Programmstart) wieder einlesen? . . . . .	48
2.5.1	Das Speichern in eine Datei . . . . .	48
2.5.2	Das Laden der Werte aus der Datei . . . . .	49
2.6	Wie kann man das Clipboard = Zwischenablage für eine Grafikspeicherung nutzen? . . . . .	49
2.7	Wie kann ich abfragen welcher Tab in in einem Tabstrip aktiv ist? . . . . .	49
2.8	Was bedeutet das h in vielen GambasCode Beispielen? . . . . .	49
2.9	Wie kann ich die Größe von Fonts feststellen ? . . . . .	50
2.10	Wie kann man eine Zeile in einem Editor teilen und wieder zusammenfügen? . . . . .	50
2.11	Wie kann man eine Funktion im Terminal Modus programmieren? . . . . .	51
2.12	Wie kann man in eine Textbox nur mit Grossbuchstaben schreiben? . . . . .	51
2.13	Direktausgabe Fenster verkleinern . . . . .	53
2.14	Neues Verzeichnis anlegen, Verzeichnis kopieren . . . . .	53
2.14.1	Neues Verzeichnis anlegen . . . . .	53
2.14.2	Verzeichnis kopieren . . . . .	54
2.15	Wie kann man das Home Verzeichnis des aktuellen Benutzers abfragen? . . . . .	57
2.16	Wie nutzt man den DateDiff Befehl? . . . . .	58
2.17	Anführungszeichen um einen Text herum . . . . .	58
2.18	Gambas Editor, nicht dokumentierte Funktionen . . . . .	59
2.19	Tabindex in gambas . . . . .	59
2.20	Zip Files . . . . .	61
2.21	Processabfrage mit Gambas . . . . .	61
2.22	Wie funktionieren Hotkeys? . . . . .	61
2.23	Image in Picture umwandeln und umgekehrt . . . . .	62

2.24	Wie kann man den Inhalt einer DrawingArea wieder einem Image zuordnen? . . . . .	62
2.25	Eurozeichen in Gambas . . . . .	62
2.26	Kann man in Gambas wie in VB mit den Zeichen \$ etc deklarieren? . . . . .	63
2.27	IF zeile compiliert nicht in einer Zeile . Wieso? . . . . .	63
2.28	Replacebefehl . . . . .	63
2.29	Programmierbares Hilfesystem? . . . . .	63
2.30	Dialog Pfad merken . . . . .	64
2.31	Datenbank: Variablen mit Feldname statt Feldname? . . . . .	64
2.32	Gambas . . . . .	65
2.33	Lernprogramme Basic . . . . .	65
2.34	Ich würde gern einen Index erstellen. . . . .	67
2.35	Zeilen oder Zeichenkette aus Datei ausschneiden . . . . .	67
2.36	Das Gambas Wikibook als Html Buch auf dem eigenen Rechner . . . . .	68
	2.36.1 Verzeichnis /wiki/ einrichten . . . . .	69
	2.36.2 Alle Dateien aus dem Wikibook Gambas kopieren . . . . .	69
2.37	Gambas-Programm . . . . .	69
2.38	Aufruf über den Konqueror . . . . .	70
<b>3</b>	<b>Die Arbeitsumgebung (IDE)</b>	<b>73</b>
3.1	Fenster in Gambas . . . . .	73
	3.1.1 Projektfenster . . . . .	73
	3.1.2 Grafisches Formfenster . . . . .	74
	3.1.3 Codefenster . . . . .	75
	3.1.4 Werkzeugsammlung . . . . .	76
	3.1.5 Eigenschaftsfenster . . . . .	77
	3.1.6 Direktfenster (Terminalfenster) . . . . .	78
	3.1.7 Ausführungsfenster . . . . .	79
	3.1.8 Hilfefenster . . . . .	79
	3.1.9 und einige mehr . . . . .	79
3.2	Wechseln zwischen Fenstern mit ALT + TAB . . . . .	79
3.3	Werkzeugsammlung . . . . .	79
	3.3.1 Inhalt der Werkzeugsammlung . . . . .	80
	3.3.2 Nachladbare Elemente . . . . .	82
3.4	Das Gambas Menü . . . . .	82
	3.4.1 Menü „Datei“ . . . . .	83
	3.4.2 Menü „Projekt“ . . . . .	83
	3.4.3 Menü „Ansicht“ . . . . .	84
	3.4.4 Menü „Werkzeuge“ . . . . .	84
	3.4.5 Menü „?“ . . . . .	85
3.5	Die Funktionstasten in Gambas . . . . .	85

---

3.6	Die wichtigsten Steuerelemente (Werkzeugsammlung / Toolbox)	85
3.6.1	Eigenschaften	87
3.6.2	Änderung zur Laufzeit	91
3.6.3	Methoden	92
3.6.4	Ereignisse	92
3.7	Befehlsknopf (Commandbutton, Button)	94
3.8	Textarea	94
3.9	Progressbar = Fortschrittsbalken	95
3.9.1	Allein	95
3.9.2	Zusammen mit dem Timer	95
3.10	Der Timer (Zeitgeber, Zeituhr)	96
3.11	Die Spaltenansicht (ColumnView)	96
3.12	Der Schieberegler	97
3.13	Die Form	97
3.14	Das Gitter (Gridview)	97
3.15	Tableview (Tabellenansicht)	97
3.16	Listview (Listenansicht)	97
3.17	Der Command Button (Befehlsknopf)	97
3.17.1	Aufruf des Befehlsknopfes mit der Tastatur	99
3.17.2	Den Commandbutton verschwinden lassen	100
3.17.3	Den Commandbutton endgültig verschwinden lassen	100
3.17.4	Den Commandbutton verschieben	101
3.17.5	Einen Commandbutton zur Laufzeit neu erstellen	101
3.17.6	Commandbuttons zur Laufzeit in einer Schleife neu erstellen	102
3.17.7	Der wandernde Befehlsknopf	102
3.18	Zahlenraten bis 27	103
3.19	Die Eigenschaften, Methoden und Ereignisse der Textarea	104
3.19.1	Eigenschaften (Properties) der Textarea	104
3.19.2	Methoden der Textarea	105
3.19.3	Ereignisse der Textarea	105
3.20	Wie kann man mit dem Cursor an das Ende des Textes in der TextArea springen?	105
3.21	Gambastextart	105
3.22	Das einfachste Timerprogramm	106
3.23	Delay zur Laufzeit ändern	107
3.24	Das Delay mit Zufallszahlen verändern	108
3.25	Countdown	109
3.26	Reaktionszeit testen	111
3.27	Der Wait-Befehl, eine Alternative zum Timer	111
3.28	ScrollBar (Schieberegler)	112

---

3.29	Beispielprogramm	113
3.30	Wie bekommt man eine neue Form?	113
3.31	Einfaches zum Thema Form	114
3.32	Fragen	115
3.33	Startform	115
3.34	Modale Form	116
3.35	Eigenschaften eines Formulars	116
3.36	Methoden einer Form	117
3.37	Ereignisse einer Form	117
3.38	Konstanten der Form	117
3.39	Neue Form	117
3.40	Datenübergabe von einem Formular zu einem anderen Formular	118
3.41	Globale Variablen in Gambas	118
3.41.1	Programmbeispiel:	119
3.41.2	Vereinfachtes Programmbeispiel zum Datenaustausch zwischen Formen	120
3.42	Kopieren des Formlayouts	121
3.43	Resize einer Textarea an eine Form	123
3.43.1	Das Programm	123
3.43.2	Der Code:	124
3.44	Ändern der Breite eines Textfeldes beim Ändern der Breite des Formulares	125
3.45	Allgemeines	125
3.46	Beispielprogramm	125
3.47	Theorie des GridView	126
3.47.1	Eigenschaften des GridView	126
3.47.2	Methoden	126
3.47.3	Ereignisse	127
3.48	GridView mit Werten füllen	127
3.49	Hintergrundfarbe für eine beliebige Zelle setzen	129
3.50	TableView-Komponente dazuladen	130
3.51	TableView nutzen	130
3.52	Zellen bearbeiten	131
3.53	Farbige Zellen	132
3.54	Navigieren mit den Cursorstasten	132
3.55	Wiki-Tabelle	132
3.56	Einfache Meldung	134
3.57	Meldung mit mehreren Antwortmöglichkeiten	135
3.57.1	Beispiel	135
3.57.2	Layout	136
3.57.3	Code	136

---

<b>4</b>	<b>Beispielprogramme</b>	<b>137</b>
4.1	Einleitung	137
4.2	Rechenausdrücke	137
4.2.1	Addy der Summierer	139
4.2.2	Potenzen	141
4.2.3	Modulo oder Modulus (Rest beim Teilen)	142
4.3	Übersicht der mathematischen Zeichen in Gambas	144
4.4	Zahlendarstellung	144
4.4.1	Einfache Rechnungen mit ganzen Zahlen	144
4.4.2	Exponential Darstellung mit E	146
4.4.3	Rechenfehler	147
4.5	Hochzählen, Runterzählen, Abschneiden, Runden etc INC, DEC, INT, FRAC	148
4.5.1	INC Hochzählen, 1 dazuzählen	148
4.5.2	DEC Herunterzählen	148
4.6	Vorzeichen und Absolutwert Sgn, Abs	149
4.6.1	Sgn (Signum) Vorzeichen einer Zahl	149
4.6.2	Absolutwert einer Zahl ABS	151
4.7	Der Val Befehl - Aus einem Text etwas Sinnvolles machen	152
4.7.1	Theorie	153
4.8	Eulersche Zahl berechnen	154
4.9	Fibonaccifolge berechnen	154
4.10	Lucasfolge berechnen	155
4.11	Das Summenzeichen $\Sigma$ programmieren	156
4.11.1	Beispiel 1	156
4.11.2	Beispiel 1a	157
4.11.3	Beispiel 2	158
4.12	Zahlen zählen	160
4.13	Funktionsdarstellung	162
4.14	Trigonometrische Funktionen in Gambas	162
4.14.1	Beispiel Programm für den <b>sin und cos</b> ohne Grafik	163
4.14.2	Grafische Ausgabe der Sinusfunktion	163
4.15	Logik	164
4.16	Primzahlen berechnen	164
4.17	Logarithmus	165
4.17.1	Anwendung von Logarithmen:	166
4.17.2	Beispielprogramm Log Befehl	166
4.17.3	Beispielprogramm Logbefehl mit beliebiger Eingabe	168
4.17.4	Beispielprogramm Dualer Logarithmus	168
4.17.5	Beispielprogramm Dekadischer Logarithmus	169
4.17.6	Beispielprogramm Logarithmusrechner	169

---

4.18	Zinsrechnung . . . . .	172
4.18.1	Beispielprogramm Einfache Zinsen ohne Zinseszinsen (lineare Verzinsung) . . . . .	172
4.18.2	Beispielprogramm Zinseszinsrechnung (exponentielle Verzinsung) . . . . .	173
4.19	Beschreibende Statistik . . . . .	176
4.19.1	Mittelwert . . . . .	176
4.19.2	Varianz und Standardabweichung . . . . .	177
4.19.3	Der Median . . . . .	180
4.20	Gauss'sche Normalverteilung . . . . .	182
4.21	Tests . . . . .	183
4.21.1	Vierfeldertest . . . . .	183
4.21.2	Runtest . . . . .	184
4.21.3	Runtest 2 . . . . .	193
4.21.4	Runtest 3 . . . . .	197
4.22	Entropie bei einfachen 01 Folgen . . . . .	199
4.22.1	Vorübung1 . . . . .	200
4.22.2	Vorübung2 . . . . .	201
4.22.3	10Statistik . . . . .	202
4.22.4	Vorübung3: Programmierung des Problems mit dem Runtest	210
4.23	Zufallsberechnungen mit der Randomfunktion . . . . .	214
4.23.1	Münzwurf . . . . .	214
4.23.2	Münze ganz oft, 01 Zufallsfolge . . . . .	214
4.23.3	Würfeln bis 6 . . . . .	214
4.23.4	6er Würfel überprüfen . . . . .	216
4.23.5	Würfeln bis 12 . . . . .	217
4.23.6	Lotto . . . . .	217
4.23.7	Zufallsbuchstaben . . . . .	219
4.23.8	Zufallspunkte . . . . .	220
4.23.9	Zufallsquadrate . . . . .	221
4.23.10	Münzwurf Statistik . . . . .	223
4.24	DrawingArea, Image . . . . .	223
4.25	Die Koordinaten der DrawingArea1 . . . . .	224
4.26	Bilder . . . . .	225
4.27	Farben . . . . .	225
4.28	Mathematische Funktionen . . . . .	225
4.29	Punkt,Linie,Kreis . . . . .	226
4.30	Punkte . . . . .	226
4.31	Linien . . . . .	227
4.31.1	Linienbreite . . . . .	228
4.31.2	Noch ein paar Linien . . . . .	229

---

4.31.3	Farbe der Linie . . . . .	229
4.32	Rechteck . . . . .	230
4.32.1	Rechteck mit Farbe füllen . . . . .	231
4.33	Kreis . . . . .	231
4.33.1	Die Röhre . . . . .	234
4.33.2	Konzentrische Kreise . . . . .	236
4.34	Tortengrafik . . . . .	238
4.35	Ellipse . . . . .	238
4.35.1	Layout . . . . .	239
4.35.2	Der Code . . . . .	240
4.36	Der Seestern . . . . .	243
4.37	Layout . . . . .	244
4.38	Der Code . . . . .	244
4.39	Picture,Image,Drawingarea,Picturebox etc . . . . .	246
4.39.1	Was ist der Unterschied zwischen einem Image und einem Picture ? . . . . .	246
4.40	Grafikfeld ( DrawingArea) . . . . .	247
4.41	Bildfeld (Picturebox) . . . . .	247
4.42	Scrollviewfeld . . . . .	247
4.43	Laden Sie ein Bild in eine PictureBox . . . . .	249
4.44	Umwandeln eines Pictures in ein Image . . . . .	250
4.45	Laden Sie ein Bild mit Dialog in eine PictureBox . . . . .	251
4.46	Laden Sie ein Bild ohne Dialog in die PictureBox . . . . .	251
4.47	Bild speichern ohne Dialog . . . . .	252
4.48	Bild speichern mit Dialog . . . . .	252
4.49	Bild aus dem Clipboard holen . . . . .	252
4.50	Bild in den Zwischenspeicher laden . . . . .	252
4.51	Resize . . . . .	252
4.52	Cached . . . . .	253
4.53	Refresh . . . . .	253
4.54	Screenshot . . . . .	253
4.55	Farben . . . . .	254
4.56	Farbspektrum . . . . .	256
4.57	Wie man MouseDown, MouseMove und MouseUp benutzt . . . . .	258
4.58	Die Koordinaten der Maus . . . . .	260
4.59	Drag and Drop , Nehmen und Ziehen mit der Maus. . . . .	262
4.60	Maustasten abfragen . . . . .	264
4.61	Eine andere Cursorform über die Eigenschaften festlegen . . . . .	264
4.62	Die Zufallsmaus . . . . .	266
4.63	Ein anderes Cursorbild mit dem Cursorbefehl . . . . .	267
4.63.1	Theorie . . . . .	268



---

4.64	Frage	268
4.65	Textausgabe	268
4.65.1	Print	269
4.65.2	String\$	271
4.66	Texteingabe	271
4.67	Text Löschen	272
4.68	Text zusammenkleben mit &	272
4.69	Neue Zeile mit & Chr\$(10) & Chr\$(13)	272
4.70	Textlabel Fett, Kursiv und farbig	273
4.71	Der Split Befehl	274
4.71.1	Einen Satz in seine Worte zerlegen	275
4.71.2	Beispiel2: Eine Zahlreihe summieren	275
4.71.3	Beispiel 3 des Splitbefehls	275
4.72	Der Replace Befehl - Zeichenersetzen	277
4.72.1	Kleine Beispiele aus der Gambas Hilfe	277
4.72.2	Beispielprogramm	277
4.72.3	Layout	278
4.72.4	Der Code	279
4.73	Instr An welcher Position steht der Teil ?	279
4.74	RInStr	280
4.75	Subst Platzhalter im Text	280
4.76	Länge eines Strings ( einer Textfolge) feststellen Len	281
4.77	<b>CHR()</b> ASCII-Zahlencode in Zeichen umwandeln	281
4.78	<b>ASC()</b> Wandelt alphanumerische Zeichen in ASCII-Zahlencode um	282
4.79	<b>Str()</b> wandelt andere Variablen in einen Textstring um	283
4.80	Zeichensatz	283
4.81	Schriftart (Font) über einen Standarddialog wählen	283
4.82	Alle verfügbaren Schriftarten aufzählen	284
4.83	Text in die Zwischenablage	284
4.84	Theorie der Zwischenablage	286
4.85	Wie kann man Grafik in die Zwischenablage kopieren und wieder heraus holen ?	287
4.86	Übersicht der Befehle	287
4.87	Wait-Befehl (Pause)	288
4.88	Zeit	288
4.88.1	Time Befehl	288
4.88.2	Timer-Funktion	289
4.88.3	Format Befehl für die Zeit und Datumsausgabe	290
4.88.4	Problem mit Null Uhr Zeit rückwärts	292
4.88.5	Problem mit Greenwich Time	293

---

4.88.6	Gambas-Buch > Zeit und Datum > "Problem mit Null Uhr Zeit rückwärts" und "Problem mit Greenwich Time"	293
4.89	Datum . . . . .	294
4.89.1	Heute . . . . .	294
4.89.2	Wochentag . . . . .	295
4.89.3	Schaltjahr . . . . .	296
4.89.4	IsDate . . . . .	297
4.89.5	CDate . . . . .	297
4.89.6	Year . . . . .	297
4.90	Fragen . . . . .	298
4.91	Datei speichern . . . . .	300
4.92	Datei speichern ohne Dialog . . . . .	300
4.93	Datei laden 1 . . . . .	301
4.94	Datei laden 2 . . . . .	302
4.95	Datei laden mit Standarddialog . . . . .	302
4.96	Datei laden und speichern mit Addyplus . . . . .	303
4.97	Dir Befehl . . . . .	305
4.98	Neues Verzeichnis anlegen . . . . .	305
4.99	Datei kopieren . . . . .	306
4.100	Wie kann man das Home Verzeichnis des aktuellen Benutzers ab- fragen ? . . . . .	307
4.101	Existiert eine Datei ? . . . . .	308
4.102	Datei laden und speichern in einem kleinen kombinierten Programm	308
4.103	Datei schreiben mit WRITE . . . . .	310
4.104	Daten einlesen mit READ . . . . .	310
4.105	Standarddialoge . . . . .	312
4.106	Theorie . . . . .	313
4.107	Größe eines Dialogwindows ändern . . . . .	313
4.108	Erstes Menü . . . . .	314
4.108.1	Untermenüs . . . . .	316
4.108.2	Codeeingabe . . . . .	316
4.109	Optionsmenü . . . . .	317
4.110	Dynamisches Menü . . . . .	318
4.111	Kontextmenü . . . . .	320
4.112	Datenbankprogrammierung nur mit Gambas . . . . .	321
4.113	CSV Datei . . . . .	322
4.114	Datenbankprogrammierung mit Gambas und MySQL . . . . .	324
4.114.1	Vorarbeiten Mysql . . . . .	324
4.114.2	Can't connect to local MySQL server through socket ...	325
4.114.3	Mysql von der Konsole . . . . .	325
4.114.4	Arbeiten mit dem Datenbankmanager von Gambas . . . . .	327

---

4.114.5 MySQL Nutzerrechte . . . . .	328
4.114.6 MySQL, Apache und phpMyAdmin . . . . .	328
4.115 Datenbankprogrammierung mit Gambas und SQLite . . . . .	328
4.116 Erstellen einer Datenbankverbindung . . . . .	329
4.117 Verfügbare Datenbanken anzeigen . . . . .	329
4.118 Links . . . . .	330
4.119 Ein paar Zeilen im Direktausdruck . . . . .	330
4.120 Drucken mit Dialog . . . . .	331
4.121 Drucken im Querformat . . . . .	332
4.122 Drucken einer Grafik . . . . .	332
4.123 Drucken in eine Postscript PS Datei . . . . .	332
4.124 Drucken Abbrechen . . . . .	332
4.125 Printer Klasse Theorie . . . . .	332
4.125.1 Eigenschaften von Printer . . . . .	333
4.125.2 Methoden von Printer . . . . .	334
4.125.3 Konstanten von Printer . . . . .	334
4.126 Printer.Print . . . . .	334
4.127 Html Browser . . . . .	335
4.128 Erweiterter Browser . . . . .	337
4.129 Gambas Web Browser . . . . .	338
4.130 Codebeispiel für die RS 232 Schnittstelle . . . . .	338
4.131 Ls unter Gambas . . . . .	339
4.131.1 Noch ein Beispiel: Unterverzeichnisse anschauen . . . . .	339
4.132 wc = wordcount unter Gambas . . . . .	340
4.133 Laden der Kompressions-Komponente . . . . .	341
4.134 Beispielprogramm . . . . .	341
4.135 In den mitgelieferten Beispielen finden sich mehrere Spiele . . . . .	343
4.136 Die Türme von Hanoi . . . . .	343
4.137 Sprint eine Minispiel für 2 . . . . .	343
4.137.1 Verbesserungen . . . . .	344
4.138 TicTacToe . . . . .	345
4.138.1 Verbesserungen . . . . .	346
4.139 Links . . . . .	347
4.140 Konverter (Text zu HTML-Code) . . . . .	347
4.141 Youtube Downloader . . . . .	349
4.142 Lernprogramm zum Addieren ( Zusammenzählen) . . . . .	351
4.143 Lernprogramm zum Subtrahieren ( Abziehen) . . . . .	353
4.144 Lernprogramm zum Multiplizieren . . . . .	354
4.145 Lernprogramm für die Vergleichszeichen >< und ist gleich . . . . .	356
4.146 Harry's Rechentrainer . . . . .	358
4.146.1 Form und Steuerelemente . . . . .	358

---

4.146.2 Code	359
4.147 Eine Datei herunterladen	359
4.148 <b>Erweiterter Taschenrechner</b>	360
4.149 <b>Berechnung der Miete</b>	364
4.150 Einfache Schleife	367
4.151 Erweiterte Schleife	367
4.152 Blutgasanalyse	368
4.153 EKG-Befundung	371
4.154 Röntgen Thoraxbefundung	375
4.155 Langzeit EKG-Befundung	375
4.156 Endoskopiebefundung	375
4.157 Ultraschallbefundung	375
4.158 Aufnahmebefund	375
4.159 Kurzarztentlassbrief	375
4.160 CT-Befundung	375
4.161 QT-Zeit	375
4.162 Umrechnung Herzfrequenz - RR Abstand	375
4.163 Gambas und SVG	375
4.163.1 Quellen	376
4.164 Blights	376
4.164.1 Wie bindet man eine Grafik ein?	377
4.164.2 Wie schaltet man eine Grafik an und aus?	378
4.165 Snake	378
4.166 Webbrowser	378
4.167 Datenbankanbindung	378
<b>5 Codierung</b>	<b>379</b>
5.1 Apostroph als Kommentarzeichen	379
5.2 Programmtests mit Kommentaren	380
5.3 Einfache Meldung	381
5.4 Meldung mit mehreren Antwortmöglichkeiten	381
5.4.1 Beispiel	382
5.4.2 Layout	382
5.4.3 Code	382
5.5 Theorie der Zuweisung (Assignment)	383
5.6 Einfache Datentypen	384
5.7 Zusammengesetzte Datentypen	385
5.7.1 Klassen	385
5.7.2 Objekte	385
5.7.3 Arrays	385
5.7.4 Collections	385

---

5.8	Umwandlung von Datentypen	385
5.9	Ganz einfach	385
5.10	Links	386
5.10.1	Gambas	386
5.10.2	Allgemein	386
5.11	Variablen	386
5.11.1	Was sind Variablen?	386
5.11.2	Deklaration (Dimensionierung, Zuweisung des Datentyps)	387
5.11.3	Assignment (Wertzuweisung)	388
5.11.4	Wie kann man Variablen begrenzen	389
5.11.5	Public - Private	389
5.11.6	Variablen allgemein definieren	389
5.11.7	Static (Statische Variable)	390
5.11.8	Globale Variablen in Gambas	391
5.12	val	395
5.13	str\$	397
5.14	asc	398
5.15	chr\$	399
5.16	int	400
5.17	fix	400
5.18	sgn	401
5.19	bst	402
5.20	cbool	402
5.21	cbyte	403
5.22	cdate	403
5.23	cfloat	403
5.24	cstr	403
5.25	cint	403
5.26	Einfache Arrays	403
5.27	Schneller füllen mit dem Arraybefehl	405
5.28	Einfache Arrayroutinen	406
5.28.1	Erzeugung	406
5.28.2	Operationen mit Elementen	407
5.28.3	Index	407
5.28.4	Mehrere Dimensionen	408
5.29	Fehler mit Arrays	408
5.30	Split » Zerlegt einen Text in ein Stringarray mit lauter einzelnen Elementen	410
5.31	Join Der Befehl zum zusammenfügen	411
5.32	For each	411
5.33	Arrayroutinen Add,Remove,Resize,Revert,Clear, Sort	412

---

5.33.1	Arrayroutinen . . . . .	412
5.33.2	Sie brauchen . . . . .	413
5.33.3	Layout . . . . .	414
5.33.4	Code . . . . .	414
5.34	Array mit Zufallszahlen füllen . . . . .	416
5.35	Lottozahlen ( 7 aus 49 ) . . . . .	418
5.36	Doppelgänger aus dem Array herauswerfen . . . . .	419
5.37	Theorie . . . . .	419
5.38	Arten von Arrays . . . . .	420
5.39	Zweidimensionales Array anlegen . . . . .	421
5.40	Array Routinen . . . . .	421
5.41	Stoffsammlung und Fragen zu Arrays . . . . .	421
5.42	Alphabetischer Befehlsindex . . . . .	422
5.43	AND (Und) . . . . .	423
5.44	OR (Oder) . . . . .	424
5.45	XOR . . . . .	425
5.46	NOT . . . . .	426
5.47	Bitweise Operatoren . . . . .	426
5.47.1	Bitweises AND . . . . .	426
5.47.2	Bitweises OR . . . . .	426
5.47.3	Bitweises XOR . . . . .	427
5.48	Übersicht der Strukturen von Gambas . . . . .	427
5.48.1	Reihenfolge . . . . .	427
5.48.2	Auswahl . . . . .	428
5.48.3	Schleifen . . . . .	428
5.49	Verzweigungen . . . . .	429
5.49.1	if - then . . . . .	429
5.49.2	if then else . . . . .	430
5.49.3	GOTO Sprungadresse . . . . .	430
5.49.4	Select Case . . . . .	431
5.50	Schleifen . . . . .	432
5.50.1	For next Schleife . . . . .	432
5.50.2	Do Schleifen . . . . .	435
5.50.3	For Each Schleife . . . . .	435
5.50.4	Schleife und Verzweigung kombiniert . . . . .	435
5.51	Sortierungen . . . . .	436
5.52	Allgemeines . . . . .	436
5.53	Ein ganz einfaches Beispiel . . . . .	436
5.54	Beispiele von Funktionen: . . . . .	437
5.54.1	Beispiel TIME: ( in die Funktion geht kein Parameter ein , es kommt trotzdem ein Wert zurück) . . . . .	437

5.54.2	Beispiel UCASE : ( in die Funktion geht ein Parameter ein und es kommt ein umgewandelter Wert zurück) . . . . .	438
5.54.3	Beispiel 3: (mehrere Parameter) . . . . .	439
5.54.4	Befehle zur Stringbearbeitung = Zeichenfolgenmanipulation . . . . .	441
5.55	Funktionen selber programmieren . . . . .	441
5.55.1	Hoch3 . . . . .	442
5.55.2	Dritte Wurzel ( hoch 1/3 ) . . . . .	442
5.55.3	Texte Zusammenkleben mit einer Funktion . . . . .	443
5.55.4	Fakultaet . . . . .	443
5.55.5	Binominalkoeffizient . . . . .	445
5.55.6	Pascalsche Dreieck . . . . .	446
5.56	Vordefinierte Konstanten . . . . .	448
5.56.1	Datei Typen . . . . .	448
5.56.2	String Konstanten . . . . .	448
5.56.3	Sortierungen . . . . .	448
5.56.4	Wochentage . . . . .	449
5.56.5	Vorgegebene Zahlenformate . . . . .	449
5.56.6	Vorgegebene Datums- und Zeitformate . . . . .	449
5.56.7	Verschiedene Formate . . . . .	449
5.57	Ripplesort . . . . .	450
5.57.1	Ripplesort mit Zufallszahlen . . . . .	452
5.57.2	Zahlenliste beliebiger Länge durchsortieren . . . . .	453
5.58	Ereignisse der Steuerelemente . . . . .	454
5.59	Ereignisse programmieren mit EVENT, STOP EVENT, RAISE . . . . .	455
5.60	Fehler abfangen mit Catch . . . . .	457
5.60.1	Beispiel 1 . . . . .	457
5.60.2	Der Code: . . . . .	457
5.60.3	Beispiel 2 . . . . .	458
5.60.4	Beispiel 3 . . . . .	459
5.61	Fehler abfangen mit TRY . . . . .	459
5.61.1	Beispiel . . . . .	459
<b>6</b>	<b>Anhang</b>	<b>461</b>
6.1	F11 . . . . .	461
6.2	Direkteingabe in das Direktfenster ohne Programm . . . . .	461
6.3	Ausgabe im Direktfenster mit dem PRINT Befehl . . . . .	461
6.4	Programmierung ohne Grafik nur im Direktfenster . . . . .	462
6.5	Frage: . . . . .	463
6.6	Datentypen . . . . .	464
6.7	Dateitypen . . . . .	464

---

6.8	String (Text) Konstanten . . . . .	464
6.9	Sortierungsarten . . . . .	464
6.10	Wochentage . . . . .	465
6.11	Vordefinierte Zahlenformate . . . . .	465
6.12	Vordefinierte Datums und Zeitformate . . . . .	465
6.13	Verschiedene Formate . . . . .	466
6.14	Weitere Konstanten . . . . .	466
6.14.1	Farben . . . . .	466
6.14.2	Tasten . . . . .	466
6.14.3	Rahmen . . . . .	466
6.15	Konstanten . . . . .	466
6.16	Escape Sequenzen . . . . .	467
6.17	GB Komponenten . . . . .	467
6.17.1	Zusätzliche Komponenten laden . . . . .	469
6.17.2	Liste der derzeitigen Gambaskomponenten . . . . .	469
6.17.3	Experimentelle Komponenten . . . . .	470
6.17.4	Programmdateien . . . . .	471
6.17.5	Eigene Projekte . . . . .	471
6.17.6	Konfigurationsdateien . . . . .	471
6.17.7	Projektdateien . . . . .	471
6.18	Wie arbeitet Gambas ? . . . . .	473
6.19	Unterschiede zwischen Visual Basic und Gambas . . . . .	475
6.20	Übersicht Dateieindungen im Vergleich: . . . . .	476
6.21	VB Form Import . . . . .	476
6.22	Globale Variablen in Gambas . . . . .	476
6.23	IsNumber . . . . .	476
6.24	Unload Befehl unter Gambas nicht verfügbar . . . . .	477
6.25	Die Inflation der Befehle . . . . .	477
6.26	Funktionen und Prozeduren . . . . .	478
6.27	Die Objektorientierte Programmierung ( OOP ) . . . . .	479
6.28	Public, Private, Static . . . . .	480
6.29	Es gibt (keine) globale Variablen wie unter VB . . . . .	480
6.30	Wünschenswert wäre ein interaktives Lernprogramm . . . . .	480
6.31	Gambas light . . . . .	481
6.32	Nachteile von Gambas . . . . .	481
<b>7</b>	<b>Autoren</b>	<b>483</b>
<b>8</b>	<b>GNU Free Documentation License</b>	<b>485</b>
<b>9</b>	<b>Bildnachweis</b>	<b>489</b>



# Kapitel 1

## Einstieg

**Nach einigen Jahren Entwicklungszeit ist Gambas die erste VB ähnliche Entwicklungsumgebung für Linux, die eine ausgereifte 2.0 Version bietet.**

Die Basis dieses Buches bilden die Gambas Miniprogramme, die ich für [meine Website](#) erstellt habe. Nachdem ich die Wikibooks entdeckt habe, entschied ich mich, sie auch hier zu veröffentlichen.

Diese **Miniprogramme** haben sich schon seit Visual Basic Zeiten sehr bewährt. Das Lernen der Programmiersprache wird so stark vereinfacht. Man hat auch ein einfaches Nachschlagewerk zur Hilfe. Jeder Interessierte sollte Gambas ohne großes Vorwissen einfach lernen können. Es ist natürlich allen gestattet, neue Kapitel einzufügen, bestehende zu verändern und verbessern. Auch ich kann noch viel in Gambas lernen und bin dankbar dafür!

Versuchen Sie für ein Problem **immer die absolut einfachste Lösung** und **die didaktisch eingängigste** anzugeben. Erst danach sollten etwas kompliziertere Lösungen als weitere Beispiele angegeben werden. Dies ist für Anfänger sehr hilfreich, da sie so den Kern einer Programmierlösung begreifen und nicht von unnötigem Ballast außen herum abgelenkt werden. Für erfahrene Programmierer steht dann vielleicht einiger banaler Code in dem Buch. Was dem einen banal erscheint, ist für andere der Einstieg, etwas wirklich zu verstehen.

Um dem angehenden Programmierer die Sprache Gambas schmackhaft zu machen, fängt dieses Buch gleich mit kleinen **Anwendungsprogrammen** an. Die kompliziertere Theorie kommt weiter hinten. Man braucht am Anfang ein paar sichtbare Erfolgserlebnisse und nicht ellenlange Theorieabhandlungen.

Um es dem Anfänger einfacher zu machen, werden die **Namen der Steuerelemente** und Klassen meistens bei den vorgegebenen Bezeichnungen belassen.

Die Befehlstasten heißen dann also immer *Button1* oder *Button2*, die Formulare *Form1*, *Form2* etc.

Suchen Sie sich ein Anwendungsgebiet aus, das Ihnen zusagt, und fangen Sie in diesem an. Hoffentlich sind die ersten Beispiele einfach genug.

Das ganze Buch steht unter der GNU Free Documentation License und ist somit frei verfügbar und darf weiterverbreitet werden, solange die [Lizenzbestimmungen](#) eingehalten werden.

Ich wünsche allen denselben Spaß beim Lernen und eventuell auch Verbessern, den ich beim Schreiben hatte!

[Rho](#) im Okt 2004 (UTC)

Zum Schluss noch ein Zitat. Es ist nicht ganz ernst gemeint: **Je dümmer der Programmierer, desto dicker die Programme.** Autor Unbekannt.

## 1.1 Was ist GambaS?

GambaS ist eine Programmiersprache zur Softwareentwicklung.

Es bezeichnet eine visuelle Open Source Programmierumgebung (IDE) für Basic unter Linux. Sie macht die **Vorteile von Visual Basic auch unter Linux KDE und Gnome verfügbar**. Es gibt eine Windows Version, die im Gegensatz zu Visual Basic lizenzrechtlich frei ist.

GambaS wird von Benoît Minisini entwickelt, der in der Nähe von Paris wohnt und dafür sorgt das GambaS immer besser und ausgefeilter wird.

Die derzeitig aktuelle Version ist GambaS 2.0. Stabile Versionen wurden mit der Linux Professional Distribution 9.x von Suse ausgeliefert und automatisch installiert. Auch für Debian (Sarge) ist ein Paket vorhanden.

Mit GambaS kann man leicht

- ein Programm mit grafischer Bedienoberfläche GUI entwickeln
- auf Datenbanken wie MySQL oder PostgreSQL zugreifen
- KDE Applikationen mit DCOP entwickeln
- das eigene Programm in viele Sprachen übersetzen.
- Visual Basic Programme unter Linux verfügbar machen.

- Netzwerklösungen erstellen.

Gambas befindet sich in einer stabilen Entwicklungsphase. Die IDE-Oberfläche ist selber in Gambas-BASIC geschrieben.

Nach dieser [Weltkarte](#) zu schließen, wird es weltweit weiterentwickelt und genutzt.

## 1.2 Warum Gambas?

Vielen Visual-Basic-Programmierern macht es Spaß, in Gambas zu programmieren.

Hat man die Microsoftwelt hinter sich gelassen und ist erfolgreich auf Linux umgestiegen, sucht man verzweifelt nach einer Programmiersprache, die ähnlich gut zu handhaben ist wie Visual Basic. Mit Gambas hat man eine gute Wahl getroffen, denn Gambas ist sehr stabil und komfortabel. Außerdem bügelt Gambas einige VB Ungereimtheiten aus und erzieht einen konsequent zur Objektorientierten Programmierung. Man lernt also noch dazu.

### 1.2.1 Zitate

„This project aims at making a graphical development environment based on a Basic interpreter, so that we have a language like Visual Basic under Linux.“

„Still, this Qt-based software is one of the most advanced open source RAD tools available.

It is nearly complete and very usable and stable. Benoit's goal is to avoid the development failures of Microsoft Visual Basic. The modern concept of Gambas is available as a graphic at the Web site. Gambas is translated into many languages. To run Gambas applications you need the Gambas interpreter component installed on the user machine.“

## 1.3 Wie kann man Gambas lernen?

Für alle, die mit Basic oder Visual Basic Programmiererfahrung gewonnen haben, ist der Umstieg auf Gambas ziemlich einfach.

Für die reine Programmierung kann man Bücher über Visual Basic zu Rate ziehen. Viele der Programme bekommt man auch unter Gambas in Gang.

Ein gutes Einsteiger-Buch ist wohl: VISUAL BASIC 6 FÜR DUMMIES. Klingt zwar etwas seltsam, ist aber ganz brauchbar. Es wird sicher sehr schnell ein Buch GAMBAS FÜR DUMMIES geben.

Oder Sie arbeiten dieses Wikibuch durch und helfen mit, es zu verbessern.

Einige Visual-Basic-Programmierkurse sind auch recht hilfreich. Siehe Links am Ende dieses Buches.

## 1.4 Gambas herunterladen

Gambas kann man als gezippte Tarfiles herunterladen und selbst kompilieren wobei jedoch fertige Packages komfortabler zu handhaben sind.

Auf [gambas.sourceforge.net](http://gambas.sourceforge.net) findet man hierzu alles, was man benötigt.

## 1.5 Gambas Installation

Am einfachsten bekommt man Gambas, wenn man es aus der eigenen Distribution installieren kann. So hat Suse ab 9.1. eine brauchbare Gambasversion schon dabei. Man muss Sie nur als root über *Yast2* und *Software* ins aktuelle System holen.

Hat man keine fertige Gambasversion in der eigenen Linux Distribution dabei oder will man die eigene Version auf den neuesten Stand bringen, kann man Gambas auch aus dem Quellcode erzeugen. Das geht leichter als man denkt.

Die Gambas Installation ist auf [dieser Seite](#) genau beschrieben.

Im Folgenden habe ich die Compilation für die Version 1.0 ins Deutsche übersetzt:.

## 1.6 Ausführliche Beschreibung

Vor der Compilation sollten Sie überprüfen, ob wichtige Komponenten auf ihrem System verfügbar sind:

- Die X11 Entwicklerpakete.

- Die Qt 3 Entwicklerpakete.
- Die KDE 3 Entwicklerpakete (so Sie denn die KDE Komponenten ebenso verwenden möchten).
- Die PostgreSQL, MySQL oder SQLite Endwicklerpakete (so Sie denn Datenbanktreiber kompilieren möchten).
- Die libcurl development packages (version 7.10.7 or greater) if you want to compile de network-curl component.
- Die SDL und SDL\_mixer Entwicklerpakete (so Sie denn die SDL Komponenten verwenden möchten).
- Die libxml und libxslt Entwicklerpakete (so Sie denn die XML Komponenten verwenden möchten).

Anschließend sollten Sie ins Internet gehen.

Sie müssen die folgenden development packages (Entwicklerpakete) installiert haben:

- X11,
- QT3,
- KDE3,
- PostgreSQL,
- MySQL.

### 1.6.1 Download

Wie man das macht, hängt von der Distribution ab.

Laden Sie den Gambas Quellcode herunter. Dazu können Sie im Konqueror die Downloaddatei von Gambas anklicken und in ihr home-Verzeichnis kopieren, oder Sie starten von der Konsole ihres Homeverzeichnisses den Befehl **wget**.

```
[linux@home ~]$ wget http://gambas.sourceforge.net/gambas-1.0.tar.bz2
```

Das Paket hat in etwa eine Größe von 4MB. Je nach der Art Ihrer Internetverbindung kann dies einige Zeit in Anspruch nehmen, bis es auf Ihrem Rechner geladen ist.

## 1.6.2 Entpacken

Entpacken Sie das Paket am besten in home:  
Hiermit wechseln Sie in Ihr home Verzeichnis:

```
[linux@home ~] cd ~
```

Entpacken:

```
[linux@home ~]$ bunzip2 gambas-1.0.tar.bz2
```

Gambas 1.0 in ein neues Verzeichnis kopieren:

```
[linux@home ~]$ tar xf gambas-1.0.tar
```

```
[linux@home ~]$ ls  
... gambas-1.0 ...
```

Gehen Sie jetzt in dieses neue Verzeichnis

```
[linux@home ~]$ cd gambas-1.0
```

Kompilieren Sie nun Gambas:

## 1.6.3 Konfiguration überprüfen

Mit `configure` wird Ihr System untersucht, um den Quellcode an Ihr System anzupassen. Es läuft dabei ein Scriptprogramm, welches in Ihrem Gambasverzeichnis steht. Deswegen geben Sie `./` vor dem Compile Befehl ein:

```
[linux@home gambas-1.0]$ ./configure  
...
```

Viele Meldungen spulen jetzt auf Ihrer Kommandozeile ab. Wenn etwas nicht funktioniert, gibt es Fehlermeldungen. Sie müssen dann zur Fehlerbeseitigung gehen.

Man kann Gambaskomponenten während der Compilation ausschließen, wenn man folgende Optionen wählt:

`--disable-kde-component` um die KDE Komponente abzuschalten  
`--disable-db-component` um die Datenbankkomponente abzuschalten

Und so weiter: geben Sie `./configure --help` um eine komplette Liste aller Option zu bekommen.

Beispiel:

```
[linux@home gambas-1.0]$ ./configure --disable-db-component
```

Bedenken Sie auch, dass automatisch Komponenten abgeschaltet werden, wenn die entsprechenden Entwicklerpakete nicht auf Ihrem Rechner zu finden sind, es erscheint dann ein Warnhinweis in der Configure Output Anzeige.

Wenn die Konfiguration ohne Fehler durchläuft, dann können Sie die Kompilation mit `make` starten:

## 1.6.4 Kompilation mit `make`

```
[linux@home gambas-1.0]$ make
...
```

Die Kompilation braucht etwa 20 Minuten auf einem alten Pentium-II 400 Mhz und mehr als 5 Minuten auf einem neuen Athlon XP 2000+. Deswegen sollte man Geduld bewahren. Sie können diese Zeit nutzen, um schon einmal ein wenig in der Dokumentation zu stöbern.

Falls ein Fehler während der Compilation auftritt, gibt es natürlich wieder eine Fehlermeldung. Da so ein Fehler ungewöhnlich ist, sollten er auf der Gambas Mailingliste gemeldet werden.

## 1.6.5 Installation als `root` `make install`

Wenn das Kompilieren ohne Probleme beendet wurde, kann man Gambas fertig installieren. Um das zu tun, muss man Rootrechte haben. Deswegen nutzen Sie das Linuxkommando `su` mit ihrem Root-Passwort danach.

```
[linux@home gambas-1.0]$ su -c "make install"
Password:
...
```

Geben Sie Ihr Root-Passwort ein und die Installation wird gestartet. Normalerweise erfolgt die Installation in `/opt/gambas`. Wenn Sie ein anderes Installationsverzeichnis nutzen wollen, dann müssen Sie die `configure`-Option `-prefix` nutzen. Lesen Sie in der `INSTALL`-Datei im Verzeichnis der Quelldatei nach, um darüber noch mehr Einzelheiten zu erfahren.

Um Gambas z. B. in `/usr` zu installieren, müssen Sie folgendes eingeben:

```
[linux@home gambas-1.0]$ ./configure --prefix=/usr
```

Beachten Sie, dass Gambas symbolische Links im `/usr/bin` Verzeichnis anlegt, so dass Sie die Gambas-Programme immer in Ihrem Pfad haben und sie so von der Entwicklungsumgebung leicht gefunden werden können.

Wenn Sie Gambas direkt in `/usr` installieren, werden diese symbolischen Links nicht erzeugt.

Wenn alles fertig ist, dann können Sie Ihre Gambas-GUI starten:

```
[linux@home gambas-1.0]$ gambas
```

Aus der Erfahrung heraus ist es am leichtesten, eine Programmiersprache zu lernen, wenn man einfache Beispiele nachvollziehen kann.

Suchen Sie sich ein Gebiet aus, das Sie interessiert, sei es nun die Grafik, die Textbearbeitung oder die Datenbankanwendung. Wenn Sie in diesem Gebiet zu programmieren anfangen, werden sie am erfolgreichsten starten, denn Sie bringen ja schon einige Vorkenntnisse mit.

Für alle Mathematiker sind kleine Rechenprogramme der geeignete Anfang. Für einen Künstler sollte es ein kleines Malprogramm sein. Ein Internetfan braucht einen eigenen Browser, den er selbst programmieren kann. Zu all dem finden sich in Gambas gute Beispiele und Möglichkeiten.

## **1.7 Starten von Gambas**

Starten Sie Gambas nach der Installation mit Alt und F2 aus dem Direktbefehlsfenster. Geben Sie einfach `gambas` ein.

Danach erscheint folgender Begrüßungsbildschirm:



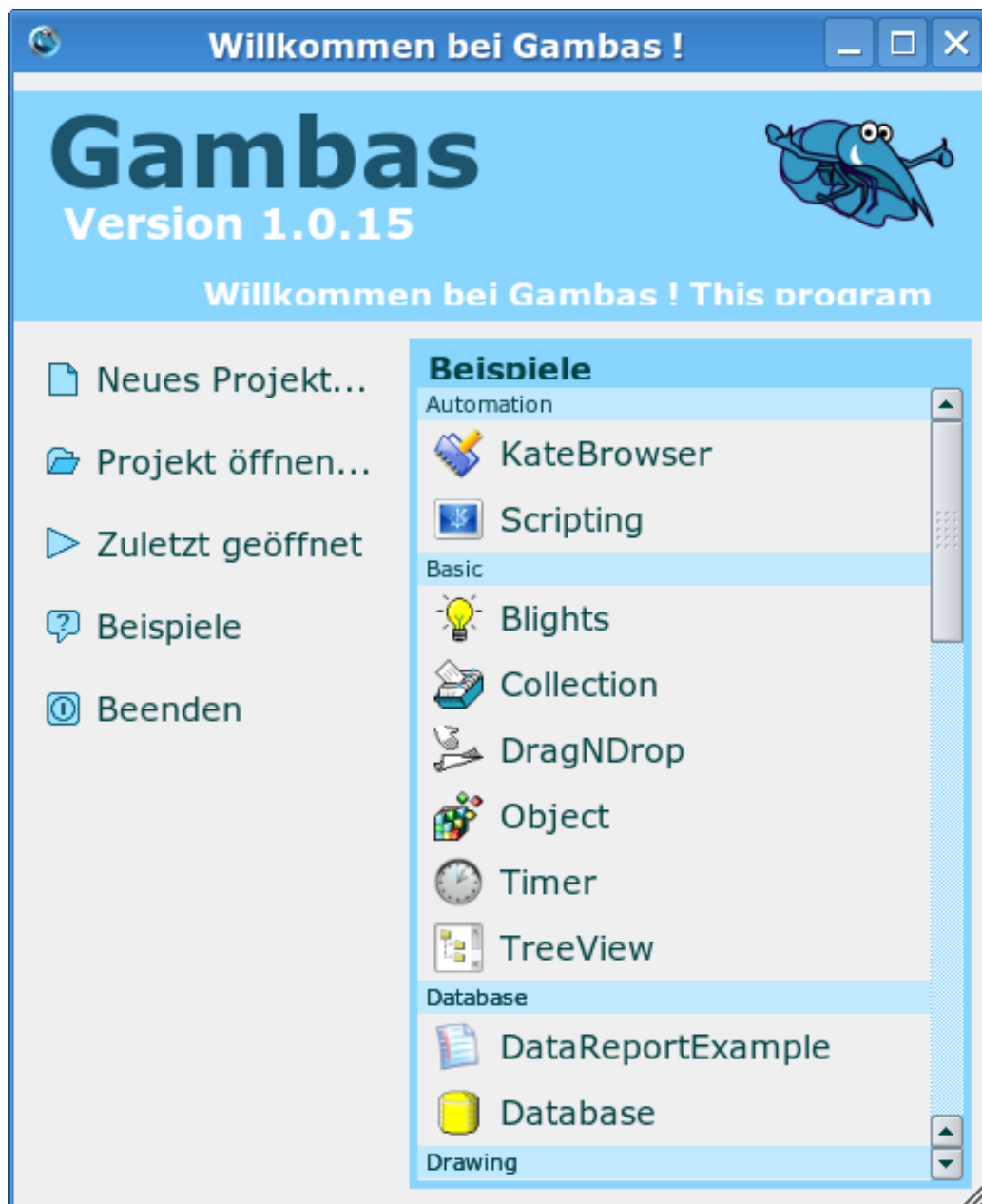


Abbildung 1: Der Begrüßungsbildschirm

Sie können Gamas auch über den KDE Menüknopf links unten starten. Dann gehen Sie zu *Entwicklung* und *Entwicklungsumgebung*. Wenn Sie Gamas z. B. als Suse-rpm-Datei installiert haben erscheint dort *Gamas* als Eintrag.

Sie können Gambas auch auf Ihren Desktop legen, Klicken sie in die freie Fläche ihres Desktops mit der rechten Maustaste. Dann wählen Sie *Neu erstellen, Datei* und *Verknüpfung zu einem Programm herstellen*. Bei den Eigenschaften müssen Sie einiges ergänzen.

So sieht ein Projekt unter Gambas aus:



Abbildung 2: Ein Projekt unter Gambas

Wenn Sie ein Gambas-Icon suchen, finden Sie dies hier:



Abbildung 3: Das Gamba-Icon

Bis jetzt ist noch kein Gamba-Icon in der Linux - KDE - Iconsammlung zu finden.

## 1.8 Hallo Welt!

Wie jede Einleitung zu einer Programmiersprache kommt diese auch nicht ohne *Hallo Welt!* aus.

### 1.8.1 Legen wir zunächst ein neues Projekt an:

- Starten Sie Gambas
- Wählen Sie in dem erscheinenden Startfenster **Neues Projekt**
- Es erscheint das Assistentenfenster, klicken Sie auf **Nächster**
- Wählen Sie den Punkt **Grafisches Projekt erstellen** aus und klicken Sie wieder auf **Nächster**
- Geben Sie dem Projekt den Namen *Hallo\_Welt* – das Titel-Feld können Sie frei lassen – klicken Sie wieder auf **Nächster**
- Wählen Sie nun ein Verzeichnis aus, in dem ihr Projekt abgespeichert werden soll. (Sie können jedoch auch die Voreinstellung lassen, da jedes Projekt in ein neues Unterverzeichnis bekommt.)
- Klicken Sie auf **Nächster**
- Klicken Sie auf **OK** um den Assistenten zu beenden und mit der Entwicklung zu beginnen

### 1.8.2 Nun erstellen Sie ein neues Formular:

- Es öffnen sich 3 Fenster
  - Das Projektfenster
  - Das Werkzeugfenster
  - Das Eigenschaftsfenster
- In dem Fenster **Projekte** klicken Sie **mit der rechten Maustaste auf Formulare** und wählen **Neu** und dann **Formular**
  - Lassen Sie den Formularnamen wie vorgeschlagen bei **Form1** und klicken Sie auf **Nächster**.
- Jetzt öffnen sich weitere Fenster.
  - das Formular-Fenster, auf dem die Oberfläche erstellt wird
  - das Code-Fenster, in das der Programmcode eingegeben wird

### 1.8.3 Jetzt wird programmiert:

- Geben Sie ins Codefenster folgendes Programm ein:

```
PUBLIC SUB Form_Open()  
    PRINT "Hallo Welt!"  
END
```

### 1.8.4 Es geht los

- Drücken Sie die Taste **F5** um das Programm zu kompilieren und es auszuführen
- Nun sehen Sie ein Fenster in dem *Hallo Welt!* steht

Im Gegensatz zu VB erfolgt die Ausgabe des Print Befehls nicht in das Grafikformular sondern ins Direktfenster.

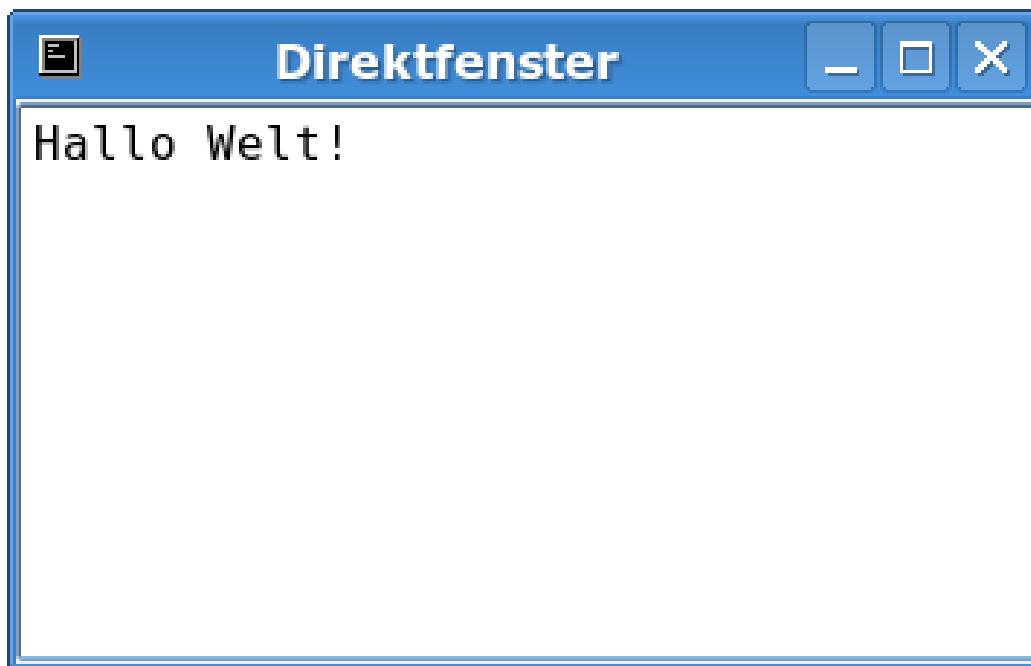


Abbildung 4: Bild:Gambas HalloWelt Direktfenster.png

*Wenn Sie das Direktfenster nicht sehen, müssen Sie das Formularfenster mit der Maus verschieben oder mit ALT und Tab dorthin wechseln.*

Stoppen Sie das Programm durch einen Klick auf die rechte obere Ecke des Formfensters.

**Versuchen Sie folgende Varianten , die ohne Direktfenster auskommen:**

```
PUBLIC SUB Form_Open()  
    ME.Caption = "Hallo Welt!"  
END
```



Abbildung 5: Bild:Gambas HalloWelt.png

**Oder als Meldung:**

```
PUBLIC SUB Form_Open()  
    Message.Info("Hallo Welt!", "OK")  
END
```



Abbildung 6: Bild:Gambas HalloWelt Message.png

Beenden Sie Gambas mit **Datei Beenden** oder **Strg + Q**. Ihr Programm wird automatisch gespeichert.

Siehe auch: [madeasy.de](http://madeasy.de)

## 1.9 Zweites Programm in Gambas

Machen Sie alles wie beim ersten Programm. Aber vergeben Sie einen anderen Namen für das Projekt, z.B. kurz und bündig **2**.

Erstellen Sie ein neues Formular wie im Abschnitt *Erstes Programm* beschrieben und nennen Sie es wie Sie wollen. Sie können auch die Voreinstellung Form1 stehen lassen.

Jetzt drücken Sie auf **F6**.

Die **Werkzeugkiste** erscheint.

Wählen Sie aus der Werkzeugkiste einen Befehlsbutton (Button OK) und eine Textbox (abc Feld) aus. Mit Doppelklick auf die Symbole werden diese auf das Formular kopiert.

Klicken Sie jetzt auf den Befehlsbutton und geben Sie folgenden Code ein:

```
PUBLIC SUB Button1_Click()  
    Textbox1.Text = "Es geht"  
END
```

Die obere und die untere Zeile stehen schon dort. Sie brauchen also nur die mittlere Zeile kopieren

```
    Textbox1.Text = "Es geht"
```

Dann drücken Sie **F5**.

Das Programm sollte starten.

Klicken Sie auf ihren Befehlsbutton.

Der Text erscheint in der Textbox.



Abbildung 7: Bild:Gambas EsGeht.png

Stoppen Sie das Programm mit dem Kontrollkästchen **X** rechts oben am Formular.

Mit **F6** können Sie die **Eigenschaften** ihrer 3 Objekte noch ändern. Probieren Sie dazu etwas mit verschiedenen Eigenschaften herum.

## 1.10 Drittes Programm: Programme im Terminalfenster ohne Grafik

Sie können Gambas auch für Programme ohne die grafische Oberfläche nutzen. Diese Programme laufen im Direktfenster (= Terminalfenster) auf der Konsole ab. Mit Visual Basic gibt es diese Möglichkeit nicht mehr.

Starten Sie Gambas wie gewohnt.

Wählen Sie **Neues Projekt**.

Folgen Sie dem Wizard und drücken Sie **Nächster**.

Bei *Typ des Projektes auswählen* wählen Sie **Terminal-Projekt erzeugen**.

Folgen Sie weiter dem Wizard und drücken Sie **Nächster**.

Geben Sie ihrem Projekt einen Namen z.B. **1mal1**.

Folgen Sie dem Wizard und drücken Sie **Nächster**.



Wählen Sie das Verzeichnis, in dem ihr Projekt als Unterverzeichnis abgespeichert werden soll.

Folgen Sie dem Wizard und drücken Sie **Nächster**.

Die Daten Ihres Projektes werden zusammengefasst dargestellt.

Drücken Sie **OK**.

Jetzt erscheint das Projektfenster. Drücken Sie mit der rechten Maustaste auf das Verzeichnis **Klassen**.

Drücken Sie dann mit der linken Maustaste auf **Neu** und **Klasse**.

Lassen Sie den Namen bei **Class1** als Startklasse und Drücken Sie **OK**.

Geben Sie als Beispiel folgendes Programm ein:

```
STATIC PUBLIC SUB Main()  
DIM x AS Integer  
DIM y AS Integer  
FOR x = 1 TO 10  
  FOR y = 1 TO 10  
    PRINT x;" mal ";y; " = "; x*y  
  NEXT  
NEXT  
END
```

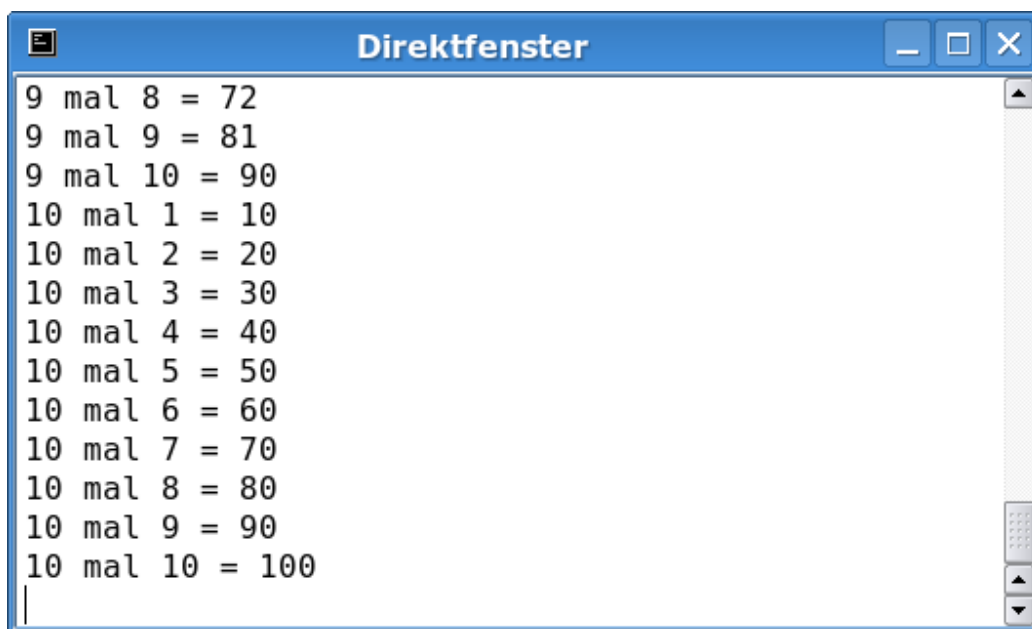


Abbildung 8: Ausgabe im Direktfenster

Drücken Sie **F5** um es zu starten.

Schauen Sie sich im Direktfenster das Ergebnis an. Eventuell mit **Alt - Tab** zwischen den Fenstern hin- und herschalten.

Beenden Sie das Programm mit einem Klick auf das **rote Quadrat** im Projektfenster. Oder mit **ALT - F4**

Beenden Sie Gambas mit **Datei Beenden** oder mit **Strg + Q** oder programmieren Sie noch etwas weiter.

Frage: Wie kann man ein nicht-grafisches Programm als ausführbare Datei auf dem Desktop zugänglich machen ?

## 1.11 Wie kann man den Programmcode aus diesem Kurs kopieren?

Hinweise zum Download der Programme aus dem Internet. So geht es:

- Markieren sie den Code auf diesen Seiten mit der Maus.
- Kopieren Sie ihn in die Zwischenablage mit **Strg + C**.
- Starten Sie ihre Gambas Programmierungsumgebung.
- Starten Sie ein **neues Projekt**.
- Starten Sie mit einer neuen Form.
- Holen Sie sich die notwendigen Steuerelemente und Befehlsfelder aus der Werkzeugleiste mit **F6**.
- Benennen Sie die Steuerelemente so, wie sie auch im Code heißen. Mit **F4** über die Eigenschaften.
- Klicken Sie das wichtigste Befehlsfeld im Entwurfsmodus an.
- Kopieren Sie den Code mit **Strg + V** in das Unterprogramm des Befehlsfeldes.
- Starten Sie das Programm mit **F5**.

## 1.12 Ausführbare Datei erstellen

Unter dem Menüeintrag *PROJEKT* wählen Sie *Ausführbare Datei erstellen*. Dies ist nur möglich, wenn das Programm auch korrekt funktioniert. Probieren Sie das vorher mit **F5** aus.

Das Programm wird dann in dem Verzeichnis abgespeichert, in dem auch ihr Quelltext steht.

Mit Gambas2 Version 1.9.49 wird eine ausführbare Datei mit der Endung `.gambas` erzeugt. Diese Datei kann zum Beispiel mit dem Terminal und dem Befehl `/home/xxx/gambas2/programm.gambas` gestartet werden. `xxx` ist der Name des Homesverzeichnisses. Es wird voraus gesetzt, dass das zu startende Programm im dem Verzeichnis `/home/xxx/gambas2` abgelegt ist und der Name der ausführbaren Datei `programm.gambas` ist.

Bei der älteren Gambas Version 1.0.15 (die mit Ubuntu 7.04 installiert wird) ist es nicht ganz so einfach, das Programm zu starten. Nachstehend ist eine Lösung geschildert, dass es auch mit Ubuntu und Gnome und vermutlich auch mit dem KDE-Desktop klappt:

Unter Ubuntu 6.10 und 7.04 und mit Gnome habe ich beim Start eines Gambas-Programmes immer folgende Fehlermeldung:

```
X Error: BadDevice, invalid or uninitialized input device 169
Major opcode: 147
Minor opcode: 3
Resource id: 0x0
Failed to open device
X Error: BadDevice, invalid or uninitialized input device 169
Major opcode: 147
Minor opcode: 3
Resource id: 0x0
Failed to open device
```

Programme (eigene und auch Beispielprogramme) funktionieren unter der Gambas-Entwicklungsumgebung, trotz Fehlermeldung, einwandfrei. Auch der Gambas Befehl „alles kompilieren“ und „ausführbare Datei erstellen“ funktionieren ohne Fehlermeldung, dennoch ist es mir nie gelungen, die ausführbare Datei zu starten. Es gab verschiedene Fehlermeldungen beim Start des Programmes. Auch mit dem KDE-Desktop ( ich installierte das Mega-Paket Kubuntu-Desktop nach), stellte sich der Erfolg nicht ein.

Nach vielen Experimenten und per Zufall kam ich doch an das Ziel: Ich gab der ausführbaren Datei den Namen „irgendwas.exe“, wobei `.exe` der entscheidende Faktor ist. Mit einem Doppelklick auf die Datei in Nautilus startete das Programm außerhalb der Entwicklungsumgebung von Gambas. Das Ganze ist mir rätselhaft, ich dachte zunächst, das Wine eventuell zum Start benötigt wird. Dies ist nicht

der Fall, da ich mit einem zweiten Rechner experimentierte, auf dem Wine nicht installiert ist.

Mit dem Hexeditor ghex habe ich die von Gambas erstellten ausführbaren Dateien „irgendwas“ und „irgendwas.exe“ miteinander verglichen. Die Dateien sind tatsächlich etwas verschieden, „irgendwas“ ist um einige Bytes kürzer als „irgendwas.exe“

## 1.13 Das Programm auf dem Desktop platzieren

Sie haben mit Gambas ein kleines Programm erstellt und ausführbar gespeichert. Jetzt wollen Sie es direkt vom KDE Desktop starten.

- Klicken Sie dazu mit der rechten Maus an einer leeren Stelle auf die KDE Oberfläche.
- Wählen Sie dann den Menüpunkt *Neu erstellen, Datei und Verknüpfen mit Programm ...*
- Im folgenden Eigenschaften-Menü wählen Sie den Punkt *Allgemein* und geben der Verknüpfung einen Namen.
- Dann wählen Sie den Punkt *Programme* und in der Zeile *Befehl* drücken Sie die Taste *Auswählen*.
- Suchen Sie das richtige Verzeichnis in dem ihre ausführbare Datei steht, z.B. `home/name/gambas99/program1`
- Wählen Sie das richtige Programm aus diesem Verzeichnis und bestätigen Sie Ihre Eingaben mit *OK*.

Jetzt wird ihr Programm auf dem Desktop erscheinen und zwar mit einem blauen Zahnrad als Symbol (= Icon).

Siehe dazu auch: <http://www.madeasy.de/7/prgkde.htm>

Um unter KDE ein nicht grafisches Programm als ausführbare Datei auf dem Desktop zugänglich machen, müssen Sie nach dem Auswählen des Programms unten rechts auf den Button *Erweiterte Optionen ...* klicken und dort *In Terminal starten* anwählen.

## 1.14 Dem Programm ein eigenes Icon (Programmsymbol) verpassen

Icons sind Bitmapdateien mit maximal 64\*64 Bildpunkten. Gut geeignet zur Erstellung eines Icons ist das Programm KIconEdit in einer neueren Version. Haben Sie ihr Programmsymbol gemalt, dann speichern Sie es am besten im selben Verzeichnis wie die ausführbare Datei ab. Dann müssen Sie noch die Desktopdatei mit dem neuen Icon verknüpfen. Dazu klicken Sie mit der rechten Maustaste auf das blaue Zahnrad ihres Programmes und wählen Sie den untersten Punkt *Eigenschaften*. Klicken Sie auf das blaue Zahnrad und suchen Sie sich ihren eigenen Icon unter *Sonstige Symbole* und *Auswählen*.

Siehe dazu auch: <http://www.madeasy.de/7/icon.htm>

## 1.15 Programmstart

Ein fertiges Programm kann man auf verschiedene Arten starten:

1. Aus der Programmierumgebung GUI mit **F5**
2. Aus der Programmierumgebung GUI über die Menüleiste
3. Vom Desktop her mit Klick auf das Programm, wenn man es kompiliert hat und im Desktop gespeichert hat.
4. Mit **Alt + F2** und Eingabe des Programmnamens, wenn man es kompiliert hat .

Spezielle Startmöglichkeiten:

- Bei Klick auf einen Befehlsbutton.
    - Diese Version ist für den Anfänger am einfachsten zu verstehen.
  - Mit `Form_Open()`
    - Automatischer Programmstart mit dem Laden einer Form
- Bei Doppelklick auf die Form (Event = `DoppelClick`)
- Beim Berühren der Form mit der Maus (Event = `Enter`)

## 1.15.1 Besonderheiten

### Ein Programm starten und sofort wieder beenden

Ob das Sinn macht sei dahingestellt. Wie es geht, zeigt folgendes Beispiel. Sie brauchen nur eine leere Form, um es in Gang zu bringen.

```
PUBLIC SUB Form_Open()  
DIM a AS String()  
DIM x AS Integer  
a = NEW String()  
FOR x = 0 TO 4  
    a.Add(Str(x))  
NEXT  
a[0] = "Anna"  
a[1] = "Anton"  
a[2] = "Toni"  
a[3] = "Max"  
a[4] = "Klaus"  
FOR x = 0 TO 4  
    PRINT a[x]  
NEXT  
ME.Close  
END
```

### Mit

```
PUBLIC SUB Form_Open()
```

wird das Programm direkt gestartet.

### Mit

```
ME.Close
```

in der vorletzten Zeile, wird es gleich wieder beendet.

Frage dazu: Wie kann man das Direktausgabefenster programmtechnisch verkleinern?

### Programmstart mit Argumentenübergabe

Noch zu bearbeiten

### zweiten Programmstart verhindern

Noch zu bearbeiten

## 1.16 Programmende

- Mit dem Schließen-Button, normalerweise ein Kreuz in der rechten oberen Fensterecke
- Ende mit *ME.Close* nicht mit *END* wie in VB!

```
PUBLIC SUB Button1_Click()  
    ME.Close  
END
```

### 1.16.1 Programmende mit ME.Close

In VB konnte man ein Programm mit dem Befehl *END* beenden. Das funktioniert in Gambas nicht. Auch der Befehl *Quit* produziert Fehlermeldungen. Nutzen Sie deswegen den Befehl *ME.Close*.

Das Beispielprogramm hat nur einen Befehlsbutton.

Starten Sie ein neues Projekt, holen Sie sich eine neue Form.

Holen Sie sich aus der Werkzeugkiste = Toolbox einen Befehlsbutton.

Drückt man auf den Button, soll das Programm sofort wieder beendet werden.

So schaut das Ganze aus:



Abbildung 9: Bild:Gambas ME Close.png

Hinter dem Button1 und hinter der Form steht folgender Code:

```
PUBLIC SUB Button1_Click()  
    ME.Close  
END
```

## 1.16.2 Ende mit der ESC Taste

```
PUBLIC SUB Form_KeyPress()  
    IF Key.Code = Key["Esc"] THEN ME.Close  
END
```

oder einfach in den Eigenschaften des Buttons für das Beenden die Eigenschaft `Cancel` auf `True` stellen. Dadurch wird beim Drücken der ESC-Taste das selbe bewirkt, als würde man den Button mit der Maus anklicken

## 1.16.3 Probleme

- Mit dem Befehl *STOP* gibt es Ärger.
- Mit dem Befehl *Quit* gibt es öfter Ärger.
- Mit `Window.Close` (ist nicht static).
- Mit der Tastenkombination **ALT + F4** kann man Fenster schließen.
- Ende und vorher Datenbank schließen

## 1.17 Programm abwürgen

- Das Programm beenden mit dem Klick auf den roten Punkt in der GUI
- Das Programm beenden mit dem Klick auf das X rechts oben am Bildschirm
- Ende mit `ME.Close`
- Programm abwürgen mit **Alt + F4**
- Programm abwürgen: Mit **Strg + ESC**, dann `kpm` aufrufen.
- Mit **Alt + F2** die Direkteingabe starten, dann `kpm` starten (`kpm + Return`)
  - Man erhält eine Aufstellung aller laufenden Prozesse
  - Den fehlerhaften Prozess markiert man und drückt unten die Taste `kill` = beenden.



Oder noch einfacher: Ein Konsolenfenster öffnen, *killall gambas* eingeben.

- Andere Möglichkeit:
  - Ein Konsolenfenster öffnen, *ps -d | grep gambas* eingeben und Prozessnummer merken.
  - Nun folgt der Befehl *kill -9 PROZESSNUMMER*. Damit wird das komplette Gambas abgeschossen.
  - Möchte man nur einen bestimmten Prozess (Dialog etc) abschießen, sucht man sich mit: *ps -ef | grep gambas* den richtigen Prozess heraus und tötet ihn wie oben beschrieben. Diese Varianten des Abschießens überlebt kein Prozess. Daher ist ein Reboot des Systems unnötig!

Programm abwürgen mit **Strg + Pause** oder **Strg + C** funktioniert in Gambas nicht.

- Programm abwürgen mit **Strg + Alt + Entf** funktioniert in Gambas nicht.



# Kapitel 2

## Hilfe

Was kann man tun, wenn man beim Programmieren mit Gambas nicht mehr weiter kommt?

### 2.1 Mitgelieferte Hilfe-Funktion

Bei jeder Gambas-Installation wird ein umfangreiches Hilfesystem mitinstalliert. Diesen Hilfebrowsers kann man gut nutzen, wenn man bei Programmierproblemen nicht mehr weiter weiß. Mit **F1** wird dieser Hilfe Browser aufgerufen:

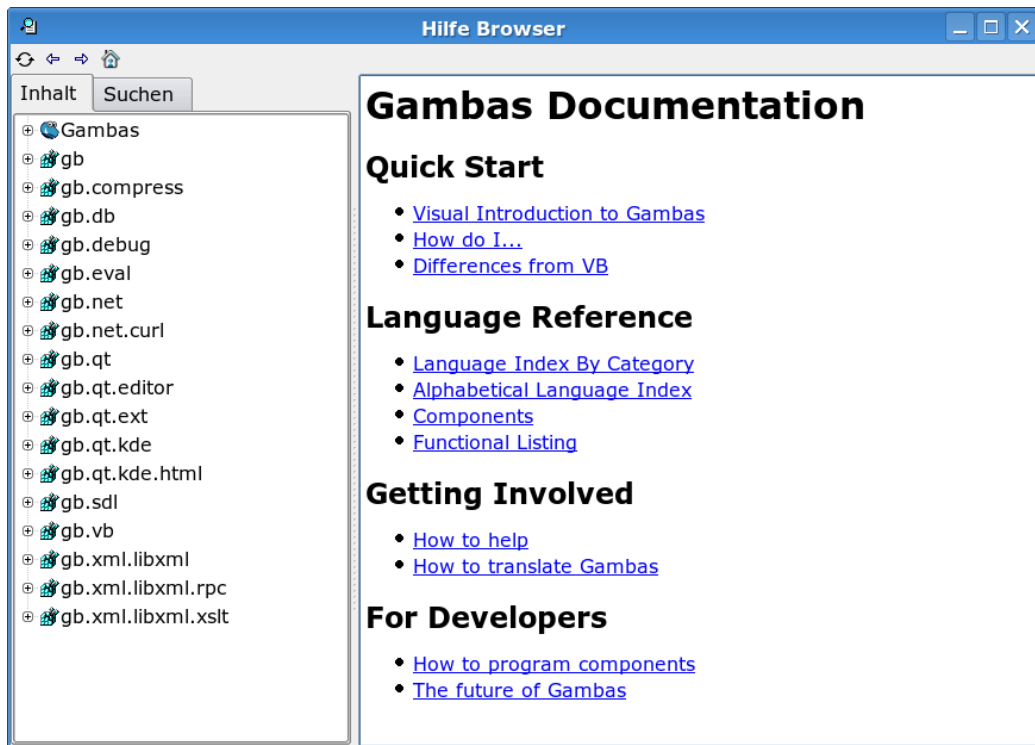


Abbildung 10: Der Hilfebrowsers von Gambas

Sehr brauchbar ist das alphabetische Verzeichnis der Gambas-Befehle. Oft findet sich zu einem Befehl auch ein kleines Beispielprogramm.

Sehr hilfreich ist auch der Aufruf der Komponenten im Hilfebrowsers.

Klickt man beispielsweise die Komponente **gb.qt** an, so erhält man eine Übersicht aller Klassen dieser sehr wichtigen Gambas-Grafikkomponente.

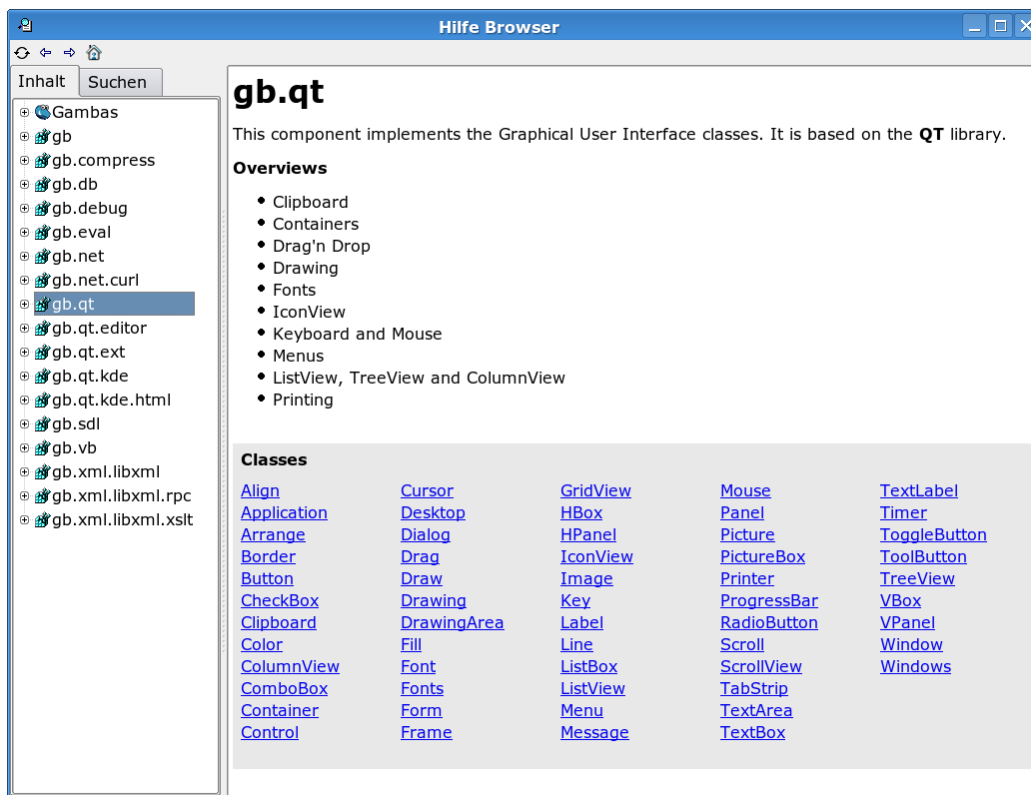


Abbildung 11: Die Komponente **gb.qt** im Hilfebrowser

Jetzt kann man wieder durch das Anklicken der dort angebotenen Klassen weitere Hilfe erhalten.

Hat man beispielsweise die Klasse **Clipboard** angeklickt erscheint eine Hilfetext zum Thema.

Dieser Text listet dann alle Eigenschaften (Properties), alle Methoden (Ereignisse) und alle Konstanten dieser Klasse auf.

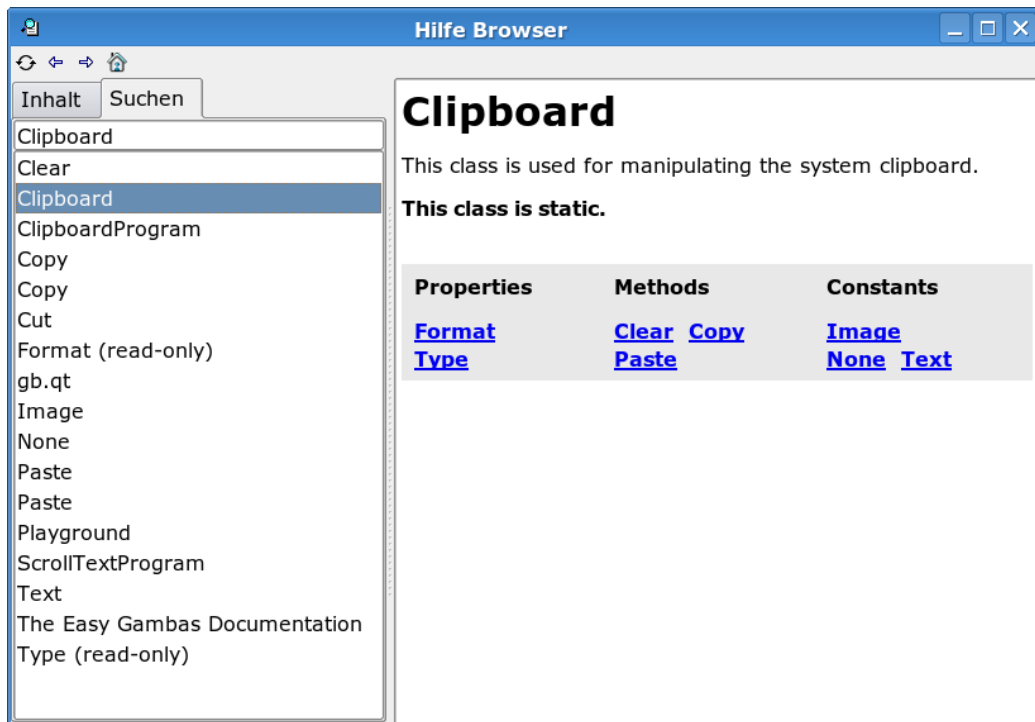


Abbildung 12: Die Klasse **Clipboard** im Hilfebrowser

Einen Nachteil hat der Hilfebrowser von Gamba's. Alle Texte sind in Englisch geschrieben.

## 2.2 Hilfe im Internet

### 2.2.1 gambasdoc.org

Der Hilfebrowser wird von den Gamba's Nutzern selbst gepflegt und verbessert. Man kann ihn über [gambasdoc.org](http://gambasdoc.org) im Internet aufrufen und alle Themen einsehen.

### 2.2.2 Diskussionsforen

Unter [dieser Webadresse](#) findet man ein sehr umfassendes englischsprachiges Diskussionsforum zum Thema Gambas. Auch Benoit Minisini ist immer wieder in diesem Forum vertreten.

[Hier](#) entsteht derzeit ein deutschsprachiges Forum.

### 2.2.3 Gambas Wikibook FAQ

Auch über das hier vorliegende Buch können Sie Hilfe zum Thema Gambas bekommen. Beschreiben Sie ihr Problem hier: [Gambas: Fragen](#)

### 2.2.4 Gambas Index

Eine Suchmöglichkeit nach Stichworten für das Gambas Wikibook findet sich hier: [Google Wikibook Gambas](#)

## 2.3 Offene Fragen

Wenn Sie Schwierigkeiten mit Gambas haben, schreiben sie ihr Problem einfach an den Ende dieses Textes. Aus Ihren Schwierigkeiten können andere lernen und man kann Lektionen und Beispiele entwickeln, die die Probleme klären. Lernen kann man am besten an überwundenen Problemen und in der Diskussion mit erfahrenen Nutzern, denen man beim Programmieren über die Schulter schauen kann.

Diskussionen zu Gambas finden sich auch hier im Netz: <http://sourceforge.net/mailarchive/forum.php?forum=gambas-user>

Einige der Fragen, die dort gestellt werden, sollen im folgenden Teil aufgearbeitet und dabei ins Deutsche übersetzt werden.

Gib es eine Möglichkeit mit Gambas eine Website zu steuern? Also z.B. mit Usernamen und Passwort einloggen und dort über einige Links weiter in ein feld text einfügen, eine natelnummer einfügen und anschließend den Link senden anklicken....

## 2.4 Cursor an das Ende eines Textes?

Wie kann ich in einem TextArea den Cursor an das Ende des Textes setzen? ( `TextArea1.Selection = Len(TextArea1.Text)` funktioniert nicht )

Antwort Siehe [Gambas: Textarea#Wie\\_kann\\_man\\_mit\\_dem\\_Cursor\\_an\\_das\\_-Ende\\_des\\_Textes\\_in\\_der\\_Textarea\\_springen\\_.3F](#)

```
PUBLIC SUB Button1_Click()  
    textareal.pos = Len(textareal.text)  
    'Der Cursor geht ans Ende des Textes  
END
```

## 2.5 Wie kann ich mehrere Variablen in eine Datei schreiben und bei Bedarf (z.B. Programmstart) wieder einlesen?

Hier ist eine komplette, funktionierende Lösung. Damit werden 5 Textteile, die mit Kommas getrennt abgespeichert wurden später wieder aufgerufen und den Feldern zugeordnet:

### 2.5.1 Das Speichern in eine Datei

”ZusFeld” ist eine verdeckt angeordnete TextArea zur Aufnahme des Array. Sie wird mit dem String ”ZUS” gefüllt. Der Inhalt des ersten Feldes ist die Variable für den Dateinamen ”NAME”. Im home-Verzeichnis des Nutzers ist danach die Datei ”NAME” gespeichert, die als Inhalt die durch Komma getrennten Werte der 5 Felder enthält.

```
Dim ZUS AS String  
DIM NAME AS String  
  
ZusFeld.Clear  
ZUS = Name.Text & Chr(44) & Feld2.Text & Chr(44) & Feld3.Text & Chr(44) & Feld4.Text & Chr(44) &  
Feld5.Text & Chr(10)  
ZusFeld.Text = ZUS  
NAME = Name.Text  
  
File.Save(system.home & "/" & NAME ,ZusFeld.Text)
```

Man kann die Felder auch mit `Chr(10)` trennen um sie nachher mit `LineInput` einzeln einzulesen und anderen Variablen zuzuordnen.



## 2.5.2 Das Laden der Werte aus der Datei

Die ComboBox enthält in diesem Falle die Namen mehrerer Wertlisten. "Name.Text = name" an Stelle von "Name.Text = Liste[0]" würde durch die Variablenübernahme aus der ComboBox an dieser Stelle auch funktionieren.

```
DIM name AS String
DIM Liste AS String[]
DIM i AS Integer

name = ComboBox2.Text
ZusFeld.Text = File.Load(system.home & "/" & name)
Liste = Split(ZusFeld.Text , ",")
FOR i = 0 TO 4
Name.Text = Liste[0]
Feld2.Text = Liste[1]
Feld3.Text = Liste[2]
Feld4.Text = Liste[3]
Feld5.Text = Liste[4]
NEXT
END
```

## 2.6 Wie kann man das Clipboard = Zwischenablage für eine Grafikspeicherung nutzen?

## 2.7 Wie kann ich abfragen welcher Tab in in einem Tabstrip aktiv ist?

Testformular besteht aus einen Button, einem Tabstrip und einer Textbox. In der Textbox steht nach dem klick auf den Button dann der Name des aktiven Tabs.

```
PUBLIC SUB Button1_Click()
    Textbox1.Text = Tabstrip1.Text
END
```

## 2.8 Was bedeutet das h in vielen GambasCode Beispielen?

```
'Gambas class file
PRIVATE $hProcess AS Process
```

First, I suggest putting comments before the code they explain. I think it is the tradition.

- "h" tells you that the variable will receive an object handle.
  - h sagt einem, dass die Variable ein Objekt handle bekommen wird
- "Process" tells you that it will be a process.
  - Process besagt, dass dieses handle von einem Prozess sein wird.

And the type of the variable confirms that it will be an object of Process class.

- Und der TYP der Variablen bestätigt, es wird ein Objekt einer Prozess-Klasse.

## 2.9 Wie kann ich die Größe von Fonts feststellen ?

how can i get the actuell pixel values of font.heigth and font.width (in same pixelvalue as ME.x and ME.y) so i can set an correct size to button or labels

```
x=font.width("This is an teststring to see how many pixels this string are long")
y=font.height("H")
```

That is device dependent (i.e. it"s different between the screen and the printer) so you use Draw.TextHeight and Draw.TextWidth to get those values. I think you might have to have a Draw.Begin active when you call those methods.

## 2.10 Wie kann man eine Zeile in einem Editor teilen und wieder zusammenfügen?

From: Maria Graciela Cesar <mgcesar@ya...> how cut a line in editor? Wie kann man eine Zeile in einem Editor teilen und wieder zusammenfügen? 2004-10-21 03:50

Hi everybody. I have been using gambas for a while and I still can find the way to do the next: I wanna write a sentence in several lines. In visual basic you can use the connectors + and &. So you write a long line in several lines. I dont know if you can do this in gambas (and if it"s a stupid question) Thanks

Ich möchte gern einen Satz aus mehreren Zeilen zusammenbauen. In VB kann man dazu die Befehle + und & benutzen. So kann man eine lange Zeile aus mehreren kurzen zusammenbauen. Kann man das auch in Gambas ?

Antwort: Im Prinzip ja , aber: Das Pluszeichen + wird in Gambas nur für mathematische Operationen benutzt. Alle String( = Text)additionen werden mit dem & Zeichen durchgeführt.

Beispiel:

```
Print "Dies ist ein Satz" & " der mit einem Nebensatz ergänzt wurde."  
Print 3 + 5
```

Siehe auch <http://www.madeasy.de/7/prgtxtplus.htm>

## 2.11 Wie kann man eine Funktion im Terminal Modus programmieren?

Bei der Programmierung von Funktionen im Terminal Modus kommt es immer wieder zu Fehlermeldungen. Im Grafikmodus funktioniert es einwandfrei.

Vergleiche [http://de.wikibooks.org/wiki/Gambas:\\_Erste\\_Schritte#Drittes\\_Programm:\\_Programme\\_im\\_Terminalfenster\\_ohne\\_Grafik](http://de.wikibooks.org/wiki/Gambas:_Erste_Schritte#Drittes_Programm:_Programme_im_Terminalfenster_ohne_Grafik)

## 2.12 Wie kann man in eine Textbox nur mit Grossbuchstaben schreiben?

Hallo ! Ich habe ein echtes Problem, an dem ich schon mehrere Stunden hänge und auch im Internet nichts gefunden habe.....

Ich möchte in eine Textbox nur mit Grossbuchstaben schreiben. Wenn ich also z.B. "a1b2c3d" eingebe, soll in der Textbox "A1B2C3D" erscheinen... Wie macht man sowas?

Erster Hinweis <http://www.madeasy.de/7/prgklein.htm>

Im Terminalmodus schaut ein Programmcode für Ucase und Lcase so aus:

```
STATIC PUBLIC SUB Main()  
DIM a AS String  
a = "Test"  
PRINT a  
PRINT LCase(a)  
PRINT UCase(a)  
END
```

Ergebnis im Direktfenster:

```
Test
test
TEST
```

Etwas schwieriger ist es möglicherweise, die Zahlen von den Buchstaben zu trennen. Aber Gambas bezieht Ucase und Lcase offensichtlich nur auf Buchstaben nicht auf Zahlen, wie folgendes Beispiel zeigt:

' Gambas class file

```
STATIC PUBLIC SUB Main()
DIM a AS String
a = "Test1234"
PRINT a
PRINT LCase(a)
PRINT UCase(a)
END
```

Ergebnis im Direktfenster:

```
Test1234
test1234
TEST1234
```

Es ist etwas schwieriger, wenn die Großschreibung direkt bei der Eingabe auftauchen soll. Leicht kann man es sich machen, wenn man erst die Eingabe auch mit kleinen Buchstaben erlaubt und dann den ganzen Text zb einer Textbox mit Ucase hochsetzt und weiterverarbeitet.

Man braucht einen Commandbutton und eine Textbox, um das Programm in gang zu bringen. Das Beispiel läuft im Grafikmodus.

```
PUBLIC SUB Button1_Click()
DIM gros AS String
gros = TextBox1.Text
' Zu ändernde Zeichenfolge.
Textbox1.Text = UCase$(gros)
' In Großbuchstaben ändern.
END
```

Vielleicht geht es auch noch einfacher:

```
PUBLIC SUB Button1_Click()
Textbox1.Text = UCase$(Textbox1.Text)
END
```

Und siehe da, es klappt.

Als einfachste Lösung ist es das CHANGE Event zu nutzen:

```
PUBLIC SUB textbox_Change()
textbox.text = UCase$(textbox.text)
END
```

Das Change Event wird bei jeder veränderung der textbox aufgerufen und schreibt den Text gleich in Großbuchstaben. Damit sind zusätzliche Buttons oder ähnliches nicht nötig, da der Text sofort umgewandelt wird.

## 2.13 Direktausgabe Fenster verkleinern

Wie kann man das Direktausgabefenster und das Formfenster bei automatisch ablaufenden Programmen aus dem Code heraus, so platzieren, dass sie sich nicht überlappen.

Ein einfacher Trick ist das Formularfenster im Entwurfsmodus von links oben nach rechts zu verschieben. Oder Sie geben

```
PUBLIC SUB Form_Open()  
    ME.X = 500  
    ME.Y = 1  
END
```

In Ihren Code ein. Wie man das Direktfenster kleiner bekommt, weiß ich auch nicht.

## 2.14 Neues Verzeichnis anlegen, Verzeichnis kopieren

Wie kann man unter Gambas

- ein Verzeichnis erstellen
- ein Verzeichnis kopieren

Copy .. to ... klappt ja leider nur bei Dateien. Mit `Dir(...,"*.*")` bekommt man auch nur die Dateien.

Da die Gambas GUI auch mit Gambas erstellt wurde und für jedes Gambasprojekt ein neues Verzeichnis erstellt wird, ist es prinzipiell möglich, dies zu machen. Nur wie?

### 2.14.1 Neues Verzeichnis anlegen

Mit dem Befehl **MakeDir** kann man ein neues Verzeichnis anlegen. Vorsicht: Es klappt nur,

- wenn Sie einen korrekten Pfad eingegeben haben,
- wenn Sie die Schreibberechtigung für diesen Pfad haben

- wenn Sie im offenen Gambasprojet Fenster ein Verzeichnis schreiben wollen. (siehe Application.Name)
- keine Datei gleichen Namens in diesem Pfad existiert.

Beispiel:

Sie brauchen einen Befehlsbutton, um das Programm in Gang zu bringen:

```
PUBLIC SUB Button1_Click()  
  MKDIR "/home/ihrname/test"  
END
```

Ersetzen Sie den Pfad *ihrname* mit Ihrem eigenen Verzeichnisnamen.

Schauen Sie mit dem Konqueror nach, ob das Verzeichnis wirklich angelegt wurde.

Am besten fügt man eine Fehlerroutine ein, für den Fall das der Pfad schon existiert oder kein Schreibzugriff besteht.

```
PUBLIC SUB Button1_Click()  
  TRY MKDIR "/home/ihrname/test"  
  IF ERROR THEN  
    message.box("Mist, hat nicht geklappt!")  
  ENDF  
END
```

## 2.14.2 Verzeichnis kopieren

Der **COPY** Befehl kopiert eine Datei von einem Verzeichnis in ein anderes.

**COPY** Quellverzeichnis/Datei TO Zielverzeichnis/Datei

Obiger Befehl kopiert eine Datei aus dem Quellverzeichnis in das Zielverzeichnis. Dabei braucht die Datei nicht unbedingt denselben Namen zu haben, wie die Zieldatei. Unterverzeichnisse werden dabei allerdings nicht kopiert.

(Note that you cannot copy directories recursively with this function.)

Beispiel:

```
' Sichert die versteckte Gambaskonfigurationsdatei  
' Das Verzeichnis muß vorhanden sein  
' und man muß den Zugriff darauf haben  
COPY "/home/ihrname/.gambas/gambas.conf" TO "/mnt/save/gambas.conf.save"
```

Ersetzen Sie in dem Beispiel *ihrname* durch den Namen ihres Homeverzeichnis.

Das Beispiel ergibt eine Fehlermeldung, wenn:

- Das Verzeichnis nicht vorhanden ist
- Kein Schreibzugriff auf das Verzeichnis existiert

- Die Zieldatei schon vorhanden ist.

Beispiel2:

Sie brauchen einen Befehlsbutton auf Ihrer Form, um das Beispiel zu starten.

```
PUBLIC SUB Button1_Click()
  COPY "/home/ihrname/.gambas/gambas.conf" TO "/home/ihrname/test/gambas.conf.save"
END
```

Wenn man mehrere Dateien auf einmal kopieren will, hilft einem der *Dir* Befehl. Siehe [Gambas: Dateien](#)

Wie **ganze Verzeichnisse mit Gambas2 kopiert** oder verschoben werden können wird nachstehend geschildert:

Wir benötigen eine Form mit zwei DirView Anzeigen für die Auswahl des Quell- und des Zielverzeichnisses. Des weiteren benötigen wir 4 Befehlsbutton.

Mit dem ersten Button kopieren wir alle Dateien des ausgewählten Verzeichnisses. Mit zweiten Button werden nur die neueren Dateien kopiert und mit dem dritten wird ein Verzeichnis an eine andere Stelle verschoben. Der vierte Button dient lediglich zum Beenden des Programmes.

Da Gambas mit dem copy-Befehl keine vorhandenen Dateien überschreibt, wird für die Kopie aller Dateien ein eventuell bereits vorhandenes Zielverzeichnis zunächst gelöscht. Anschließend werden die Dateien kopiert.

Aus Zeitgründen ist es bei großen Verzeichnissen sinnvoll, nur neuere Dateien zu kopieren. Für diesen Fall werden die Dateien im Zielverzeichnis nicht gelöscht. Erkennt Gambas, dass eine Datei schon vorhanden ist, werden die Daten der letzten Speicherung miteinander verglichen. Ist die zu kopierende Datei neuer als die vorhandene, wird die vorhandene Datei gelöscht und durch die neuere ersetzt.

Das Verschieben eines Verzeichnisses erfolgt wie das Kopieren aller Dateien. Es wird lediglich zusätzlich das Quellverzeichnis gelöscht.

Hier ist der komplette Code der Form:

```
PUBLIC SUB Form_Open()
  DirView1.Root = "/" 'angezeigtes Startverzeichnis
  DirView2.Root = "/"
END

PUBLIC SUB cmdEnd_Click()
  ME.close
END

PUBLIC SUB cmdCopyAll_Click()
  ' wenn vorhanden: Zielverzeichnis löschen
  BasisModule.Folder_delete(DirView2.Current & DirView1.Current)
  ' Quellverzeichnis in das Zielverzeichnis kopieren
```

```

    BasisModule.Folder_kopieren(DirView1.Current, DirView2.Current)
END

PUBLIC SUB cmdCopyNew_Click()
    ' Quellverzeichnis in das Zielverzeichnis kopieren
    ' nur neuere Dateien werden kopiert
    BasisModule.Folder_kopieren(DirView1.Current, DirView2.Current)
END

PUBLIC SUB cmdMove_Click()
    ' wenn vorhanden Zielverzeichnis löschen
    BasisModule.Folder_delete(DirView2.Current & DirView1.Current)
    ' Quellverzeichnis in das Zielverzeichnis kopieren
    BasisModule.Folder_kopieren(DirView1.Current, DirView2.Current)
    ' Quellverzeichnis löschen
    BasisModule.Folder_delete(DirView1.Current)
END

```

Des Weiteren wird ein Modul mit dem Namen `BasisModule` benötigt, indem sämtliche Routinen für die obigen Aufgaben hinterlegt sind. (Die Routinen können auch in der Form enthalten sein, ich denke mit dem Modul ist es übersichtlicher) Erzeugen Sie das Modul und kopieren Sie den nachfolgenden Code.

```

' Gambas module file
PUBLIC SUB Folder_kopieren(folderFrom AS String, folderTo AS String)
    DIM file AS String
    DIM fileTemp AS String
    DIM info AS stat
    DIM info1 AS stat
    DIM fileFrom AS String
    DIM fileTo AS String
    DIM folderTemp AS String[]
    DIM j AS Integer
    FOR EACH file IN RDir(folderFrom, "**")
        fileFrom = folderFrom & "/" & file
        fileTo = folderTo & fileFrom
        TRY COPY fileFrom TO fileTo
        IF ERROR.Code = 38 THEN ' Datei ist vorhanden
            ' Überprüfung ob Quell- oder Zieldatei neuer ist
            info = Stat(fileFrom)
            info1 = Stat(fileTo)
            IF info.LastModified > info1.LastModified THEN
                ' Quelldatei ist neuer
                ' Zieldatei wird gelöscht
                KILL fileTo
                ' Quelldatei kopieren
                COPY fileFrom TO fileTo
            END IF
        END IF
    NEXT
    IF ERROR.Code = 45 THEN ' Ziel-Verzeichnisse existieren nicht
        folderTemp = Split(fileTo, "/")
        FOR j = 0 TO folderTemp.Count - 2
            fileTemp = fileTemp & "/" & folderTemp[j]
            TRY MKDIR Right$(fileTemp, Len(fileTemp) - 1)
            TRY COPY fileFrom TO fileTo
        NEXT
    END IF
    fileTemp = ""
NEXT
END

PUBLIC SUB File_delete(folder AS String)

```



```

DIM file AS String
DIM file_ AS String
'alle Dateien löschen
FOR EACH file IN RDir(folder, "**") ' *.* nur Dateien mit Endung werden gelöscht
  file_ = folder & "/" & File      ' * auch Dateien ohne Endung werden gelöscht
  TRY KILL file_
NEXT
END

PUBLIC SUB Folder_delete(folder AS String)
DIM file AS String
DIM file_ AS String
DIM folderTemp AS String[]
DIM i AS Integer
' zunächst alle Dateien löschen
File_delete(folder)
' alle Unterverzeichnisse löschen
FOR EACH file IN RDir(folder)
  file_ = folder & "/" & File
  folderTemp = Split(file_, "/")
  TRY RMDIR file_
  IF error.code = 0 AND file_ = folder THEN
    RETURN
  END IF
  IF error.code = 0 THEN ' Verzeichnis wurde gelöscht
    i = 1
    DO
      'versuchen das oder die Elternverzeichnis zu löschen
      file_ = Left(file_, Len(file_) - Len(folderTemp[folderTemp.Count - i]) - 1)
      TRY RMDIR file_
      IF error.code = 0 AND file_ = folder THEN
        RETURN
      END IF
      i = i + 1
    LOOP UNTIL error.code = 42 OR error.code = 38 'Verzeichnis nicht leer
  END IF
NEXT
END

```

## 2.15 Wie kann man das Home Verzeichnis des aktuellen Benutzers abfragen?

Der Befehl *Application.home* funktioniert dazu leider nicht. Auch der Befehl *Application.Name* ergibt den Pfad des Programmes, welches gerade läuft, aber nicht das Homeverzeichnis.

Das Homeverzeichnis des aktuellen Benutzers erhält man mit dem Befehl **system.home**

### Beispiel

```

PUBLIC SUB Form_Open()
PRINT system.Home
END

```

### Beispiel2:

```
PUBLIC SUB Form_Open()
ME.Text = "Systeminfo"
textareal.text = "Charset = " & system.Charset & Chr(13) & Chr(10)
textareal.text = textareal.text & "Domain = " & system.Domain & Chr(13) & Chr(10)
textareal.text = textareal.text & "Homeverzeichnis = " & system.Home & Chr(13) & Chr(10)
textareal.text = textareal.text & "Host = " & system.Host & Chr(13) & Chr(10)
textareal.text = textareal.text & "Sprache = " & system.Language & Chr(13) & Chr(10)
textareal.text = textareal.text & "Pfad = " & system.path & Chr(13) & Chr(10)
textareal.text = textareal.text & "Benutzer = " & system.User
' PRINT system.Charset
' PRINT system.Domain
' PRINT system.Home
' PRINT system.Host
' PRINT system.Language
' PRINT system.Path
' PRINT system.User
END
PUBLIC SUB Button1_Click()
ME.Close
END
```

Man braucht eine Textarea und einen Befehlsbutton, um das Programm zu starten.

## 2.16 Wie nutzt man den DateDiff Befehl?

... I don't know how to call the DateDiff() function. Could somebody give me a short example?

Of which class DateDiff is a method?

Der DateDiff Befehl stammt aus Visual Basic. Er errechnet die Differenz zwischen 2 Datumseingaben. Man braucht dazu die Gambas gb.vb Komponente Siehe [Gambas: Komponenten](#)

## 2.17 Anführungszeichen um einen Text herum

Ich möchte gern eine Textausgabe erzeugen, die mit Anführungszeichen eingeschlossen wird. Beispiel:

Aus `/home/user/file_name.asm` soll `"home/user/file_name.asm"` werden. Wie geht das?

Lösung:

```
button1.text = chr(34) & "home/user/file_name.asm" & chr(34)
```

---

## 2.18 Gambas Editor, nicht dokumentierte Funktionen

Hallo, ich arbeite an einem kleinen Programm, das den GambasEditor nutzt. Leider ist dieser in der Gambas Hilfe kaum dokumentiert. Wer kann mir folgende Eigenschaften und Methoden erklären:

- - Frozen
- - Highlight
- - Lineheight
- - Positions
- - Showchange
- - Showcurrent
- - Showproc
- - Symbols
- - Userelief
- - Analyze
- - Ensurevisible

## 2.19 Tabindex in gambas

Wie funktioniert die Tabtaste in einer Form? Kann man die Reihenfolge der angesprungenen Steuerelemente verändern? Gibt es so etwas wie den Tabindex in VB?

Am besten arbeitet man mit dem Befehl:**Setfocus**

Beispiel: Man hat mehrere Textboxen und man möchte eine mit der Tabtaste überspringen.

```
PUBLIC SUB TextBox1_KeyPress()  
    TextBox2.SetFocus  
END
```

Laut B.Minisini heißt der "TabIndex" in Gambas ZOrder. Diese ZOrder verändert man über die IDE.

Beispiel: Beim Erstellen der Form die TextBox markieren. Dann rechte Maus Taste im Menü > "Anordnung" > "In den Vordergrund" klicken nächste TextBox auswählen. in der selben Reihenfolge wird dann mit Tabtaste gesprungen.

Die ZOrder ist keine Object Eigenschaft (textBox im obigen Beispiel).

Wenn man die ZOrder mittels Code ändern will, muß man die Raise() oder Lower() Methode der Control Klasse benutzen.

Eine andere Möglichkeit, den Durchlauf des Fokus bei der Betätigung der Tabulatortaste zu beeinflussen: Man verläßt Gambas und öffnet das zu verändernde Formular mit einem Editor (Dateiname: *Formularname*.form im entsprechenden Projektverzeichnis). Hier ist der XML-Quelltext zu sehen. Jetzt ändert man die Reihenfolge der Objekte entsprechend der gewünschten Tabulator-Durchlaufreihenfolge, indem ein kompletter Block eines Objektes mit seinen Eigenschaften, verschoben wird. Ein Block ist an den geschweiften Klammern zu erkennen. Hier muß darauf geachtet werden, den Block nicht außerhalb eines übergeordneten Blockes (beispielsweise durch einen Rahmen entstanden) zu verschieben. Gambas arbeitet den Quelltext in Bezug auf die ZOrder von oben nach unten ab. Diese Methode hat gegenüber der eher unübersichtlichen Methode über die Gambas-IDE den Vorteil, bei Formularänderungen, wo beispielsweise nur ein Feld hinzugefügt wird, nicht noch einmal für alle Objekte die Ordnung festlegen zu müssen

Beispiel (ZOrder/TabIndex: 1.Nachname, 2.Vorname):

```
{ Nachname TextBox
  Move(128,32,448,24)
  Text = ""
}
{ Vorname TextBox
  Move(128,80,448,24)
  Text = ""
}
```

Umsortiert (ZOrder/TabIndex: 1.Vorname, 2.Nachname):

```
{ Vorname TextBox
  Move(128,80,448,24)
  Text = ""
}
{ Nachname TextBox
  Move(128,32,448,24)
  Text = ""
}
```

## 2.20 Zip Files

Wie kann man mit Gambas auf ein Archiv (zip, tar, Jar etc.) bzw. auf die darin enthaltenen Dateien zugreifen kann?

Antwort siehe [Gambas: Kompression](#)

## 2.21 Processabfrage mit Gambas

Wenn man den output eines Befehles der Process\_Read() enthält, auslesen will, dann bekommt man nur einen Prompt ( Eingabe ) zurück.

Beispiel:

```
SHELL command WAIT FOR READ WRITE AS myProcess
PUBLIC SUB Process_Read()
'      reads the line into a variable
'
      if the variable is "user prompt:" then
          WRITE getUserInput()
      end if
END
```

Dieses Beispiel funktioniert nicht, wie kann man es funktionstüchtig bekommen?

## 2.22 Wie funktionieren Hotkeys?

Wenn man Hotkeys in einem Projekt nutzen will, wird man vielleicht folgenden Code benutzen:

```
Public sub Form_KeyPress ()
```

```
    If key.code = ... then
        Something happens
```

Leider funktioniert das nicht richtig. Ist die Form leer, dann klappt es. Sind andere Steuerelemente vorhanden, funktioniert es nicht mehr. Wieso???

Bis jetzt gibt es kein Interface für globale Unterbrechungen mit dem Keyevent ( Tastenereignis) in QT. QT liegt Gambas zugrunde.

Dennoch gibt es einen Trick, um das Problem zu lösen.

Wenn man ein Menu in der Form angelegt hat, dann funktionieren die Tastenkürzel des Menüs auch global. Man kann also ein verstecktes Menu zur Form hinzufügen, in welches man die gewünschten Hotkeys einfügt.

Siehe [Gambas: Menü](#)

## 2.23 Image in Picture umwandeln und umgekehrt

In Gambas gibt es 2 Möglichkeiten Bilder zu bearbeiten: Image und Picture Was ist der Unterschied zwischen beiden? Wie kann man ein Image in ein Picture umwandeln und umgekehrt? Das Problem tritt zum Beispiel auf wenn man den Inhalt des Zwischenspeichers (in diesem fall ein Bild) in Gambas darstellen will? Siehe [Gambas: Bilder](#)

## 2.24 Wie kann man den Inhalt einer DrawingArea wieder einem Image zuordnen?

## 2.25 Eurozeichen in Gambas

Wenn man ein Euro Zeichen in eine TextArea einliest und dann folgendes versucht

```
t$ = Conv$(TextArea.Text, Desktop.Charset, "ISO-8859-1")
```

bekommt man die Fehlermeldung **”Bad string conversion”**.

Lösung: Das Eurozeichen existiert im Zeichensatz ISO-8859-1 nicht, erst in ISO-8859-15 ist es zu finden

## 2.26 Kann man in Gambas wie in VB mit den Zeichen \$ etc deklarieren?

## 2.27 IF zeile compiliert nicht in einer Zeile . Wieso?

Folgender Befehl `IF x = 3 THEN TRY MKDIR "temp_folder"` compiliert nicht in einer zeile . Wieso passiert das?

## 2.28 Replacebefehl

Warum funktioniert folgender Code nicht?

```
Dim a as string
Dim b as string
a = "Test"t"
b= Replace$(a, "", "\\")
```

b sollte "Test\t" werden

Das Zeichen \ ist ein spezielles Zeichen. Versuchen Sie

```
b= Replace$(a, "", "\\")
```

Siehe [Gambas:\\_Textbearbeitung#Der\\_Replace\\_Befehl\\_-\\_Zeichenersetzen](#)

## 2.29 Programmierbares Hilfesystem?

Gibt es unter Gambas ein programmierbares Hilfesystem wie zb .chm in vb?

Nein, nicht direkt. Dafür bietet GNU/Linux einige Möglichkeiten

- das KDE Hilfe System,
- das GNOME Hilfe System,
- man pages ( der Konqueror stellt sie sehr schön dar!)
  - Geben Sie einmal **man: /** an Ihrer Konqueroradressleiste ein.

– Siehe auch: [http://de.wikibooks.org/wiki/Linux-Kompodium:\\_-Linux\\_auf\\_dem\\_Desktop#Web-Browser](http://de.wikibooks.org/wiki/Linux-Kompodium:_-Linux_auf_dem_Desktop#Web-Browser)

html Seiten, die direkt einen Webbrowser aufrufen

## 2.30 Dialog Pfad merken

Wie kann man es erreichen, daß sich Gambas den einmal geöffneten Dialog Pfad merkt?

Siehe zb [Gambas:\\_Bilder#Laden\\_Sie\\_ein\\_Bild\\_mit\\_Dialog\\_in\\_eine\\_Picturebox](#)

## 2.31 Datenbank: Variablen mit Feldname statt Feldname?

Wenn man eine Datenbank öffnet, kann man mit einem Result-Handle auf die einzelnen Felder zugreifen, insbesondere mit der `.edit`-methode. Leider scheint der Result-Handle nur Namen zu unterstützen, aber keine Variablen.

Beispiel (nicht lauffähig, nur zur Demonstration):

```
DIM name AS String
DIM sFeldVorname AS String="vorname"
DIM sFeldNachname AS String="nachname"
.
.
sql_string = "SELECT * FROM kunde WHERE vorname LIKE " & name
.
.
rs = cn.EXEC(sql_string)
rs!sFeldVorname="Hugo"
rs!sFeldnachname="Müller"
```

—  
Die Felder 'vorname' und 'nachname' existieren im Table.

Das SQL-Statement ist nur zur Demonstration, real funktioniert es auch mit der `.edit`-Methode nicht. `rs!vorname` und `rs!nachname` funktionieren, wenn aber



deren Werte in den Variablen 'sFeldVorname' bzw. 'sFeldNachname' übergeben werden sollen, nicht.

Wie verwendet man also Variablen als Feldname ?

Schon mal als Try&Error abgetan: rs!"Feldname",rs![Feldname],rs!(Feldname),rs!("Feldname")

## 2.32 Gambas

- [Die Gambas Seite](#) (englisch)
- [Gambasforum](#)
- [Gambasforum](#)
- [Gambas-User Forum](#) (englisch)
- [Alle Befehle](#)
- [gambasforge.net Programme und Code im Austausch](#) (englisch) [nicht erreichbar]
- [Guter Übersichtsartikel in LinuxUser](#)
- [Älterer Übersichtsartikel in LinuxUser](#)
- [Einiges zum Thema Gambas in Deutsch, Miniprogramme](#)
- [Eine Einführung mit einem simplen Beispiel](#)
- [Slashdot hat Gambas zur Kenntnis genommen mit großer Resonanz](#)
- [Tutorial](#) (englisch und spanisch) [nicht erreichbar]
- [Gambas Vor- und Nachteile im Vergleich zu VB](#) (englisch, pdf, 126kB)
- [Ein paar schöne Programme](#) (spanisch)
- [Blog eines Aktivisten](#) (englisch)

## 2.33 Lernprogramme Basic

- [Schönes Lernprogramm für vb4](#) (auch für höhere Versionen geeignet)
- [Gut gemachter VB Kurs](#)

- [Umfangreiche VB-Informationen \(FAQ, Beispiele, Tipps\)](#)
- [Schöne VB Einführung](#)
- [VB Tutorial in Englisch](#)

Wenn Sie eine Programmidee oder einen Programmwunsch haben, denn Sie selbst nicht in Gambas realisieren können, weil Sie keine Zeit oder nicht genügend Gambas Know How haben, dann schreiben Sie es hier her.

Vielleicht findet sich ja jemand , der das Ganze realisiert.

#### **Abbildung: Der kleine g und der große G**

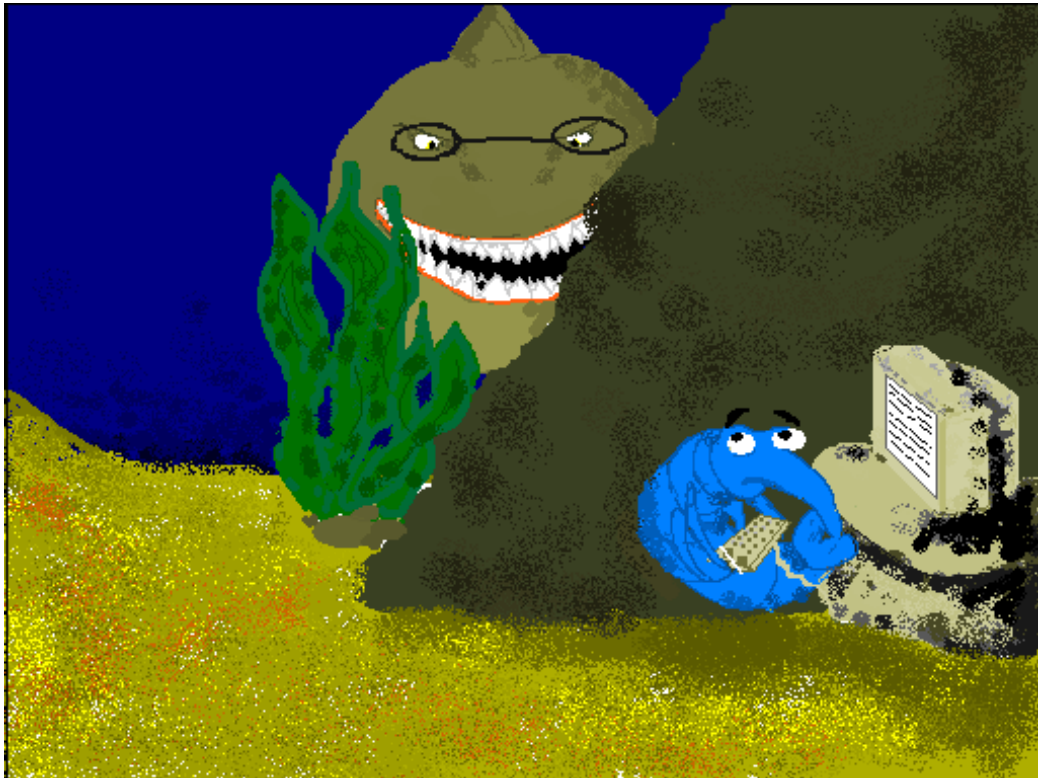


Abbildung 13: Bild:Der kleine g 1.PNG

Irgendwie fühlte sich der kleine g bei seiner Arbeit ständig beobachtet.

## 2.34 Ich würde gern einen Index erstellen.

Wie kann ich dies realisieren?

Was soll der Index indizieren ? Eine Datenbank ? Etwas genauere Informationen sind notwendig um hier zu helfen.

-Der Index soll ebenso wie die Gambas Hilfe ein Tutorial in einem Hilfefenster anzeigen.

Unter [http://de.wikibooks.org/wiki/Gambas:\\_Html](http://de.wikibooks.org/wiki/Gambas:_Html) steht ein Gambas Html Browser. Wenn man die Hilfeseiten als HTML Seiten erstellt, kann man ihn schön als Hilfetutorial nutzen.

## 2.35 Zeilen oder Zeichenkette aus Datei ausschneiden

Ich lese eine ganze Datei mit READ ein und suche später nach einen bestimmten String. Bei dessen auftreten lese ich die ganze Zeile ein. Nun möchte ich diese Zeile löschen und die nachfolgende an die vorherige Zeile anfügen, sodaß meine Datei nicht unnötig groß wird. Läßt sich dies mit den derzeitigen Befehlscode bewerkstelligen? Ich finde auch keinen Befehl mit dem ich eine bestimmte Zeile ansprechen kann, um an diese noch einen Zeichenstring anzuhängen.

**Rho:** Ich fange einmal an das Problem zu zerlegen. Als erstes habe ich den READ Befehl genauer beschrieben.

Siehe [http://de.wikibooks.org/wiki/Gambas:\\_Datei#Daten\\_einlesen\\_mit\\_READ](http://de.wikibooks.org/wiki/Gambas:_Datei#Daten_einlesen_mit_READ)

Zur weiteren Lösung des Problems müßte es aber etwas konkreter beschrieben werden. Ein Beispiel wäre hilfreich.

Benötigt werden: CommandButton, TextArea und ID.txt-File.

Als Bsp folgendes Programm:

```
i AS Integer
ii AS Integer
hfile AS File
PUBLIC SUB Form_Open()
OPEN "ID.txt" FOR READ AS #hFile
READ #hFile,tarl.Text,Lof(hFile)
END
PUBLIC SUB cmd1_Click()
i = Instr(tarl.Text, "A")
IF i > 0 THEN
```

```
ii = Instr(tar1.Text, "\n",i)
IF ii > 0 THEN
    tar1.Text = Mid(tar1.Text,i,ii-i)
ENDIF
ENDIF
END
```

In "ID.txt" stehen mehrere Zeichenstrings mit je nachfolgendem CR. Nun möchte ich einen davon ausschneiden. Jedoch den geänderten Text nicht in einer TextArea ablegen, sondern wieder in ID.txt abspeichern. Fehlermeldung bei öffnen der File mit CREATE oder WRITE kommt ein "Access Forbitten". Zugriffsrechte der File wurden aber auf "all" gesetzt. Gibt es Befehle die dies vereinfachen, oder besser gesagt an benötigte Zeile springen diese ausschneiden und nachfolgende Zeile hochrücken?

Weiteres Beispiel wäre:

**ListBox, CommandButton**

```
txt as String
PUBLIC SUB cmd1_Click()
    txt = libx1[0].Text
    libx1.Remove(0)
END
PUBLIC SUB Form_Open()
    libx1.Add("aaaaa\n")
    libx1.Add("bbbbbb\n")
    libx1.Add("aaaaa\n")
    libx1.Add("bbbbbb\n")
    libx1.Add("aaaaa\n")
END
```

Dies scheint mir die Beste Möglichkeit zu sein, jedoch erhebt sich die Frage welche Methode die geringste Bearbeitungszeit benötigt oder das Programm verlangsamt sollte die ListBox auch noch angezeigt werden. Die Liste könnte dabei sogar bis über 100 Einträge beinhalten.

## 2.36 Das Gambas Wikibook als Html Buch auf dem eigenen Rechner

Man kann sich das Gambas Wikibook recht einfach als verknüpfte HTML Seiten auf dem eigenen Rechner einrichten.

### 2.36.1 Verzeichnis */wiki/* einrichten

Damit die Verknüpfung korrekt funktioniert müssen Sie auf ihrem Rechner ein Verzeichnis */wiki* einrichten. Öffnen Sie dazu unter KDE eine Konsole, und erstellen Sie dort ein Verzeichnis:

```
mkdir /home/BENUTZER/wiki
```

Dabei BENUTZER jeweils durch Ihren Benutzernamen ersetzen. Anschließend in das neue Verzeichnis wechseln:

```
cd /home/BENUTZER/wiki
```

### 2.36.2 Alle Dateien aus dem Wikibook Gambas kopieren

Dann kopieren Sie aus dem Wikibook die folgende Datei in dieses Verzeichnis:

```
wget -nv -O index.html
'http://de.wikibooks.org/w/index.php?title=Spezial%3APrefixindex&namespace=0&from=Gambas'
```

Aus dieser Datei die Adressen der restlichen Seiten extrahieren und downloaden:

```
wget -nv -c $(cat index.html | tr '"' '\n' | egrep '^/wiki/Gambas' | sort -u | sed
's#^#http://de.wikibooks.org#')
```

Zum Schluss die Links anpassen:

```
for i in $(ls); do sed 's#href="/wiki/Gambas#href="./Gambas#g' $i > $i.temp; mv $i.temp $i; done
```

Dann können Sie die Übersicht als Gambas Hilfebrowser auf ihrem Rechner einrichten. Um immer die neuesten Seiten aus dem GambasWikibook herunterzuladen, wiederholen Sie die letzten drei Schritte.

## 2.37 Gambas-Programm

Das zuvor beschriebene Verfahren funktioniert schon recht gut, ist aber unbequem. Und schließlich ist dies hier ein Lehrbuch zur Gambas-Programmierung, was also könnte näherliegen, als ein entsprechendes Programm zu schreiben.

Erstellen Sie mit dem Wizard ein neues grafisches Projekt, nennen Sie es **Wikibook\_Download**, wählen Sie einen Speicherort und schließen sie den Wizard ab.

Erstellen Sie eine neue Form **Form1**. Platzieren Sie die folgenden Steuerelemente auf der Form:

- 4 Button
- 2 TextLabel

- 2 TextBox
- 1 TextArea
- 1 ProgressBar

Zuletzt noch das Codefenster öffnen und den folgenden Quellcode einfügen: [Wikibook Download Form1.class](#)

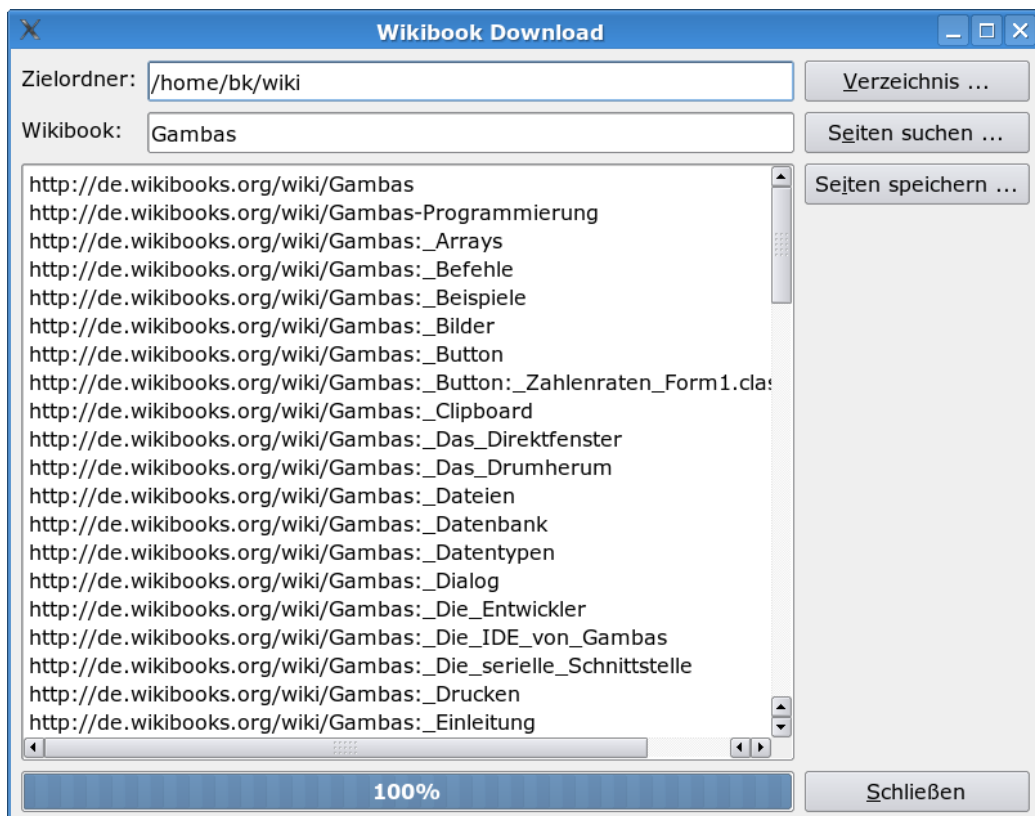


Abbildung 14: Wikibook Download

Dann können Sie das Programm starten.

## 2.38 Aufruf über den Konqueror

Das Ganze wird dann über den Konqueror aufgerufen. Auch andere Wikibücher können Sie so lokal auf Ihrem Rechner verfügbar machen.

Richten Sie dazu auf Ihrem Desktop die Seite `/wiki/Gambas` mit der rechten Maustaste als *Neu erstellen, Datei und Verknüpfung zu Adresse (URL)* ein. Geben Sie unter der Adresse `/home/BENUTZER/wiki/Gambas` ein.

Vielleicht suchen Sie sich dann noch ein schönes Symbol bei den Eigenschaften heraus.

Unter Windows funktioniert der Aufruf mit dem Explorer nicht korrekt.





# Kapitel 3

## Die Arbeitsumgebung (IDE)

IDE ist eine Abkürzung aus dem Englischen und steht für: *Integrated Development Environment*. Auf deutsch heißt dies *integrierte Entwicklungsumgebung* oder etwas kürzer **die Arbeitsumgebung** von Gambas.

### 3.1 Fenster in Gambas

#### 3.1.1 Projektfenster

Das Projektfenster ist die wichtigste Übersicht in Gambas. Vom Projektfenster aus erreicht man alle anderen Fenster und aus dem Projektfenster heraus beendet man meist auch seine Gambassitzung. Deswegen sollte man sich mit den [Menüpunkten](#) des Projektfensters vertraut machen. Am Anfang braucht man nur einige wichtige davon.

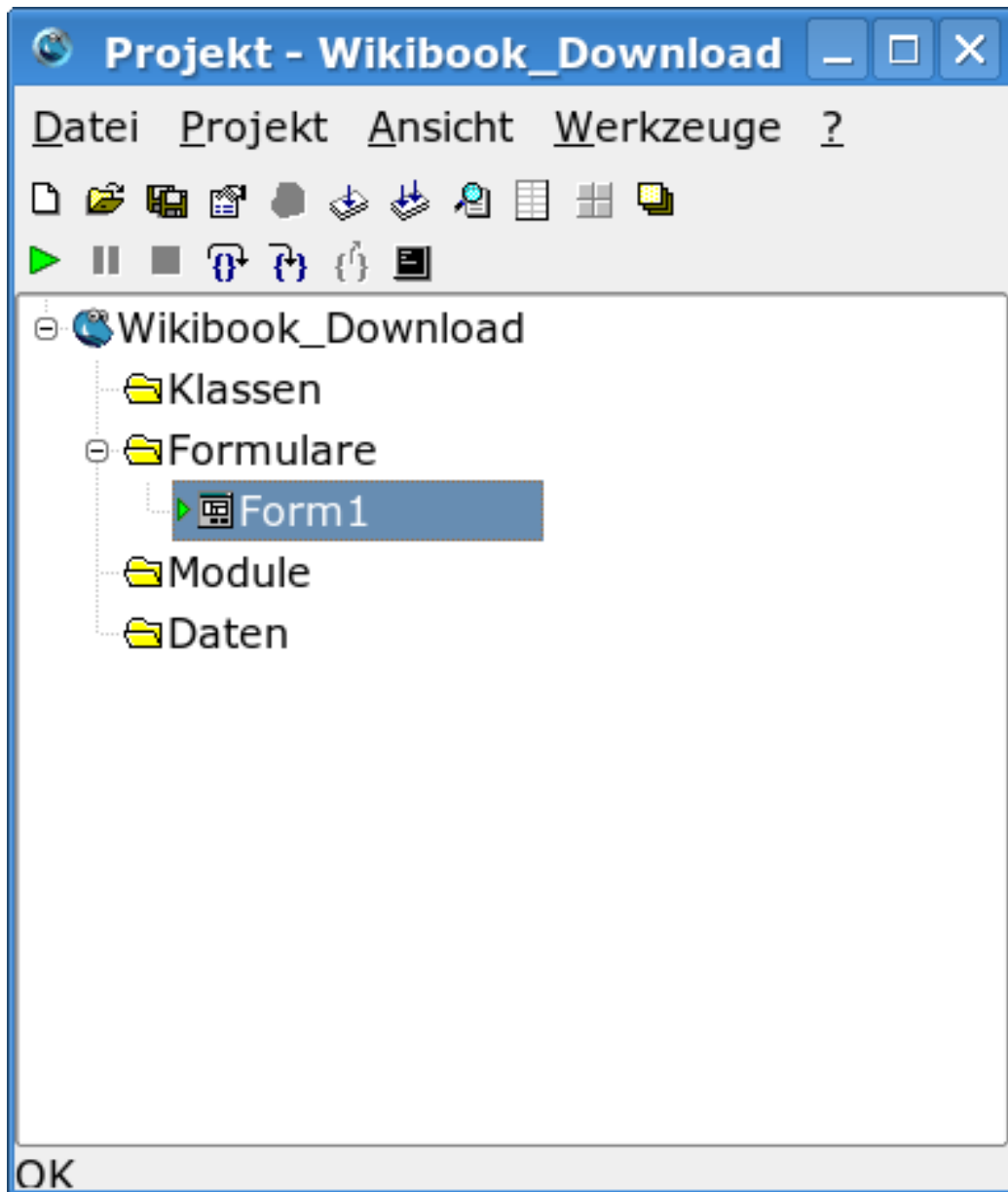


Abbildung 15: Das Projektfenster

### 3.1.2 Grafisches Formfenster

Das grafische Formfenster stellt die aktuelle Form und ihre Steuerelemente im Entwurfsmodus dar. Die Steuerelemente können mit der Maus verschoben, ver-

größert und verkleinert, markiert, kopiert und gelöscht werden. Durch einen Doppelklick auf ein Steuerelement gelangt man ins zugehörige Codefenster.

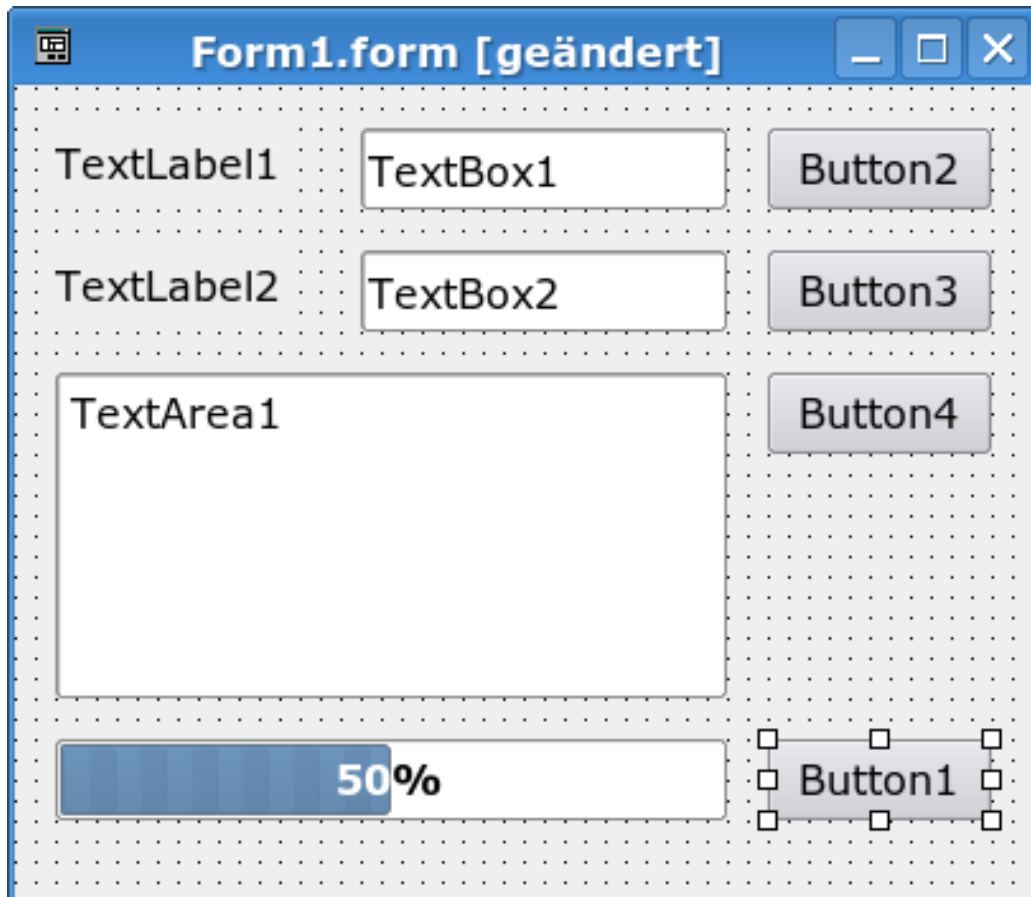
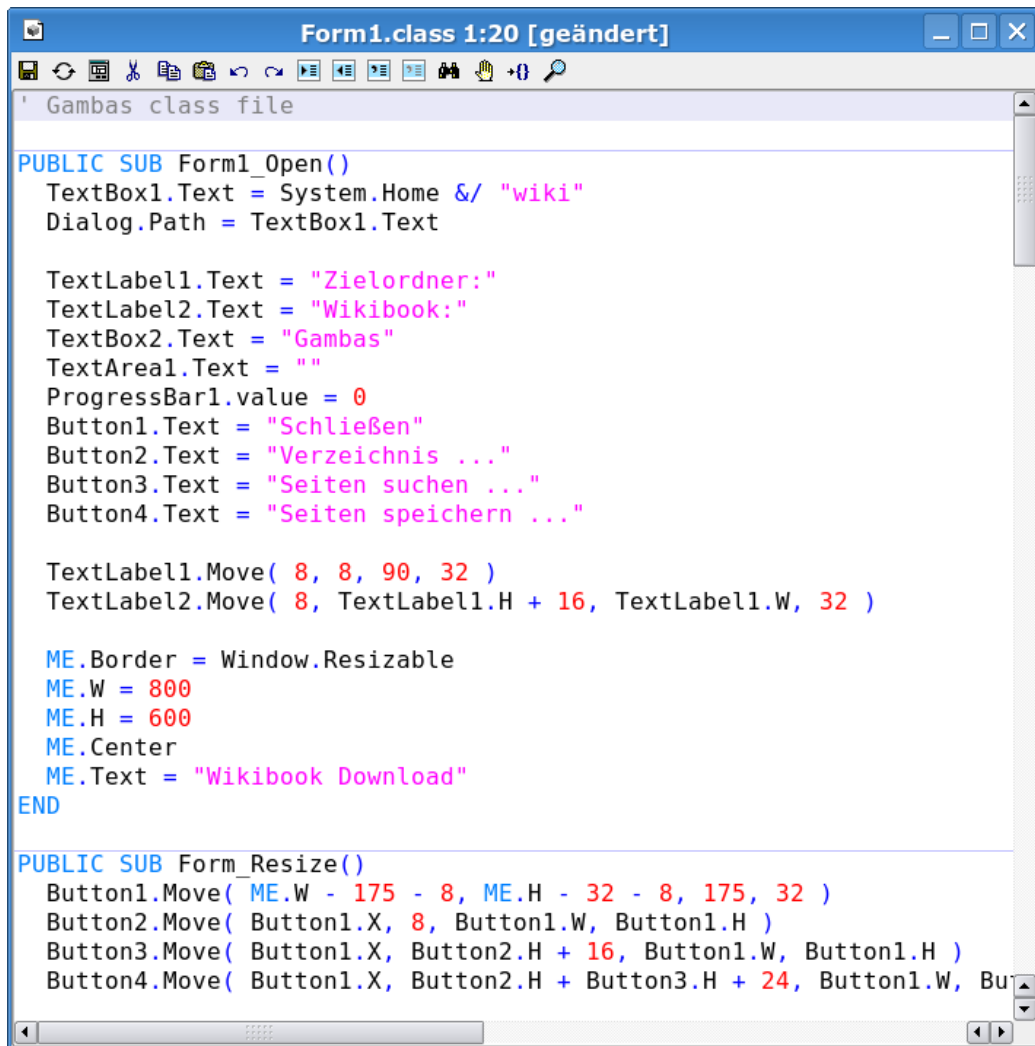


Abbildung 16: Das Formfenster

### 3.1.3 Codefenster

Im Codefenster wird tatsächlich programmiert. Das Codefenster ist ein praktischer Editor mit besonderer Farbhervorhebung der Gambasbefehle.



```
Form1.class 1:20 [geändert]
Gambas class file

PUBLIC SUB Form1_Open()
  TextBox1.Text = System.Home &/ "wiki"
  Dialog.Path = TextBox1.Text

  TextLabel1.Text = "Zielordner:"
  TextLabel2.Text = "Wikibook:"
  TextBox2.Text = "Gambas"
  TextAreal.Text = ""
  ProgressBar1.value = 0
  Button1.Text = "Schließen"
  Button2.Text = "Verzeichnis ..."
  Button3.Text = "Seiten suchen ..."
  Button4.Text = "Seiten speichern ..."

  TextLabel1.Move( 8, 8, 90, 32 )
  TextLabel2.Move( 8, TextLabel1.H + 16, TextLabel1.W, 32 )

  ME.Border = Window.Resizable
  ME.W = 800
  ME.H = 600
  ME.Center
  ME.Text = "Wikibook Download"
END

PUBLIC SUB Form_Resize()
  Button1.Move( ME.W - 175 - 8, ME.H - 32 - 8, 175, 32 )
  Button2.Move( Button1.X, 8, Button1.W, Button1.H )
  Button3.Move( Button1.X, Button2.H + 16, Button1.W, Button1.H )
  Button4.Move( Button1.X, Button2.H + Button3.H + 24, Button1.W, Bu
```

Abbildung 17: Das Codefenster

### 3.1.4 Werkzeugsammlung

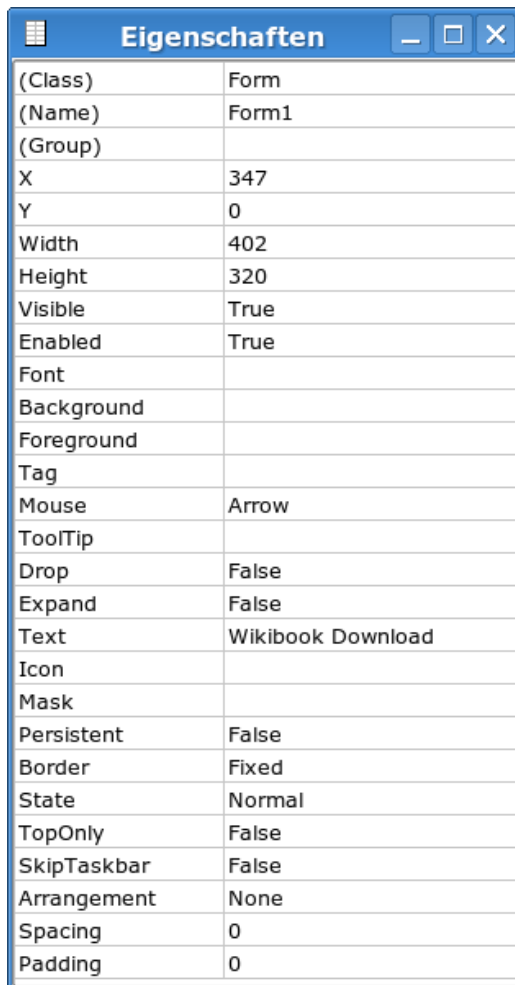
Die Werkzeugsammlung stellt die grafischen Steuerelemente für ein neues Projekt bereit. Durch einen Doppelklick auf ein Steuerelement wird es ins grafische Formfenster übernommen. Die Werkzeugsammlung erreicht man am schnellsten mit der Funktionstaste **F6**.



Abbildung 18: Die Werkzeugsammlung

### 3.1.5 Eigenschaftsfenster

Jede Form und jedes Steuerelement hat verschiedene Eigenschaften. Im Eigenschaftsfenster sind sie aufgelistet und können verändert werden. Alternativ kann man Eigenschaften aber auch im Code verändern oder festlegen. Das Eigenschaftsfenster erreicht man am schnellsten mit der Funktionstaste **F4**.



Property	Value
(Class)	Form
(Name)	Form1
(Group)	
X	347
Y	0
Width	402
Height	320
Visible	True
Enabled	True
Font	
Background	
Foreground	
Tag	
Mouse	Arrow
ToolTip	
Drop	False
Expand	False
Text	Wikibook Download
Icon	
Mask	
Persistent	False
Border	Fixed
State	Normal
TopOnly	False
SkipTaskbar	False
Arrangement	None
Spacing	0
Padding	0

Abbildung 19: Das Eigenschaftsfenster

### 3.1.6 Direktfenster (Terminalfenster)

Das Direktfenster ist eine Neuerung von Gimpas gegenüber VB. Es entspricht dem Terminalfenster von Linux und dient vor allem als Direktausgabe für PRINT Befehle im Code. Insbesondere für die Fehlersuche ist es sehr hilfreich. Manche Programme laufen auch komplett im Terminalfenster ab. Das Direktfenster erreicht man am schnellsten mit der Funktionstaste **F11**.

### 3.1.7 Ausführungsfenster

### 3.1.8 Hilfefenster

Mit F1 kommt man ins Hilfefenster. Hier findet man alle Befehle in alphabetischer und funktionaler Auflistung. Oft findet sich auch ein kleines Programmierbeispiel, so dass man das Hilfefenster ausgiebig nutzen sollte.

### 3.1.9 und einige mehr

Zusätzliche Screenshots mit allen Fenster findet sich auf [gambas.sourceforge.net](http://gambas.sourceforge.net)

## 3.2 Wechseln zwischen Fenstern mit ALT + TAB

Hat man auf einem Bildschirm mehrere Fenster gleichzeitig offen, dann kann man mit der Tastenkombination **ALT + TAB** zwischen den verschiedenen Fenstern hin- und herschalten. Es geht natürlich auch mit der Maus und einem Klick auf die angezeigten Fenster in der Kontrolleiste.

Da Gambas mehrere Fenster nutzt, platziert man diese am besten nebeneinander, so dass sie immer zu sehen sind oder wechselt zwischen ihnen mit **ALT + TAB** hin und her.

## 3.3 Werkzeugsammlung

Die Werkzeugkiste (= Toolbox) enthält alle grafischen Steuerungselemente (= Controls) die man auf einer Gambas Form nutzen kann. Das reicht von Bezeichnungsfeldern über Textfelder zu LCD Nummern und Zeitgeber.

Die Werkzeugkiste wird mit der Funktionstaste **F6** aufgerufen. Man kann sie auch über das Dateimenü aufrufen. Hat man eine Form offen und doppelklickt mit der Maus auf ein Steuerungselement, dann springt es automatisch auf die Form und kann dort weiter bearbeitet werden.

Die **Eigenschaften** eines Steuerungselementes werden mit **F6** aufgerufen.

Am Anfang genügen ein paar Werkzeuge: z.B. der Commandbutton, die Textbox, das Bildfeld, die Drawingarea und der Timer. Den Rest kann man sich nach und nach erarbeiten.

Siehe auch [Gambas: Steuerelemente](#).

### 3.3.1 Inhalt der Werkzeugsammlung



Abbildung 20: Die Werkzeugsammlung

(Reihenfolge wie im Bild)

Steuerelement	Beschreibung
Auswahlfeil	(Kein Steuerelement!) Dient zur Verschiebung von bereits auf der Form platzierten Steuerelementen



Label	Bezeichnungsfeld
TextLabel	Textbezeichnungsfeld
PictureBox	Bildbox
ProgressBar	Fortschrittsbalken
Button	Befehlsknopf
CheckBox	Auswahlknopf
RadioButton	Auswahlknopf
ToggleButton	an/aus Schalter
ToolButton	Befehlsknopf
TextBox	Textfeld
ComboBox	Kombinationslistenfeld
TextArea	großes Textfeld
ListBox	Listenfeld
ListView	Listenansicht
TreeView	Baumansicht
IconView	Iconansicht
GridView	Gitteransicht
ColumnView	Spaltenansicht
HBox	Ordnet die enthaltenen Steuerelemente horizontal in einer Reihe an
VBox	Ordnet die enthaltenen Steuerelemente vertikal in einer Spalte an
HPanel	Ordnet die enthaltenen Steuerelemente horizontal an, wenn nötig in mehreren Reihen
VPanel	Ordnet die enthaltenen Steuerelemente vertikal an, wenn nötig in mehreren Spalten
Frame	Rahmen, z.B. um Radiobuttons zusammenzufassen
Panel	Zum Zusammenfassen von zusammengehörigen Steuerelementen
TabStrip	Karteireiter
ScrollView	Ansicht mit Schieberegler waagrecht und senkrecht
DrawingArea	Zeichenfeld
Timer	Zeitgeber

### 3.3.2 Nachladbare Elemente

Einige andere Elemente sind nutzbar, müssen aber in extra Komponenten nachgeladen werden, siehe auch [Gambas: Komponenten](#).

- GambasEditor
- LCDNumber
- Dial
- SpinBox
- [ScrollBar \(Schieberegler\)](#)
- Slider
- TableView
- Splitter
- Workspace

## 3.4 Das Gambas Menü

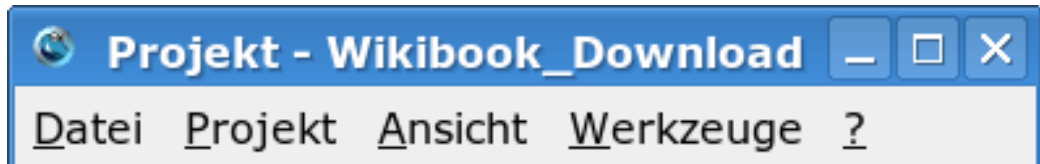


Abbildung 21: Das Menü

### 3.4.1 Menü „Datei“

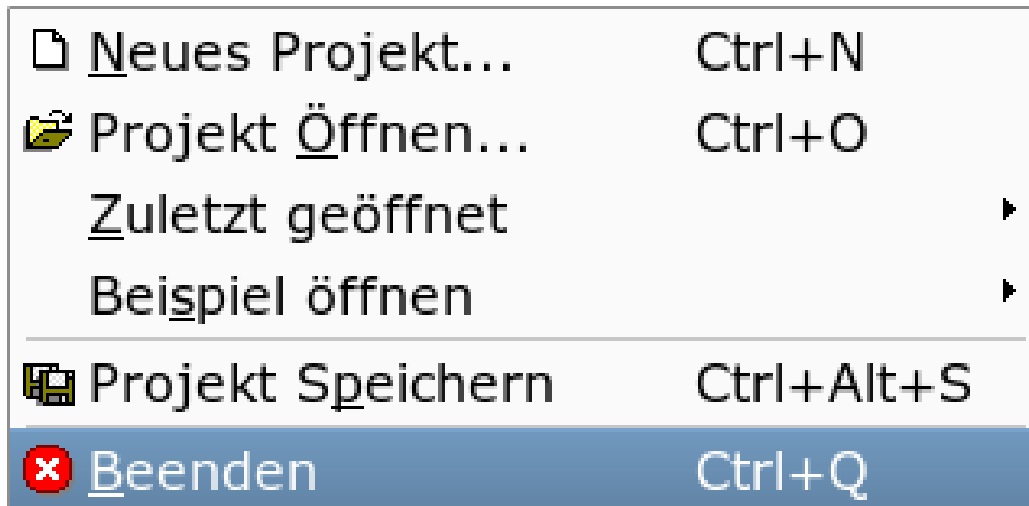


Abbildung 22: Das Menü „Datei“

### 3.4.2 Menü „Projekt“



Abbildung 23: Das Menü „Projekt“

### 3.4.3 Menü „Ansicht“

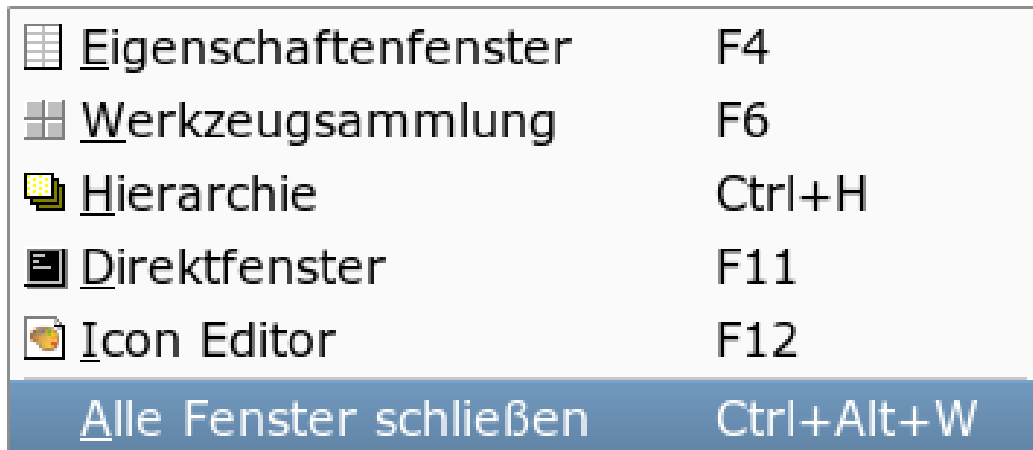


Abbildung 24: Das Menü „Ansicht“

### 3.4.4 Menü „Werkzeuge“

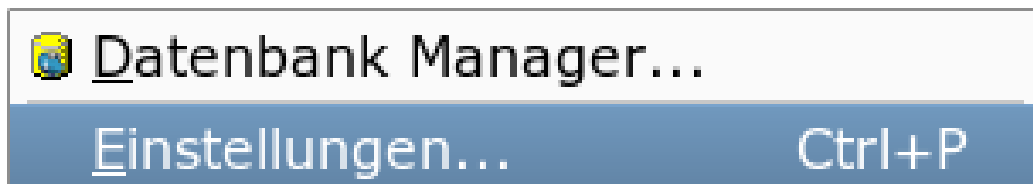


Abbildung 25: Das Menü „Werkzeuge“

### 3.4.5 Menü „?“



Abbildung 26: Das Menü „?“

## 3.5 Die Funktionstasten in Gambas

Einige Funktionstasten sind in Gambas sehr hilfreich:

<b>F1</b>	Damit ruft man wie üblich das Hilfesystem auf. Zu vielen Befehlen gibt es auch schon ein paar Beispielprogramme.
<b>F4</b>	Damit rufen Sie das Eigenschaftsfenster eines Objektes auf.
<b>F5</b>	Damit starten Sie ihr Programm
<b>F6</b>	Diese Taste zeigt Ihnen die Werkzeugkiste
<b>F7</b>	Compiliert ihr Programm
<b>F8</b>	Startet im Einzelschrittmodus
<b>F11</b>	Zeigt das Direktfenster

## 3.6 Die wichtigsten Steuerelemente (Werkzeugsammlung / Toolbox)

Die wichtigsten Steuerelemente sind die

- Form oder auch Formular genannt
- Der Befehlsbutton, (Commandbutton, Befehlsknopf, OK-Knopf in der Toolbox)
- Die Textbox (abc-Feld in der Toolbox)
- Die Textarea (gambas almost m..Feld in der Toolbox)
- Die DrawingArea (Feld mit Grafiksymbolen in der Toolbox)
- Der Zeitgeber (Timer, Feld mit Uhr in der Toolbox)



Abbildung 27: Timer Icon in Gambas

Fangen Sie mit diesen zu arbeiten an. Danach können Sie recht einfach auch die anderen nutzen.

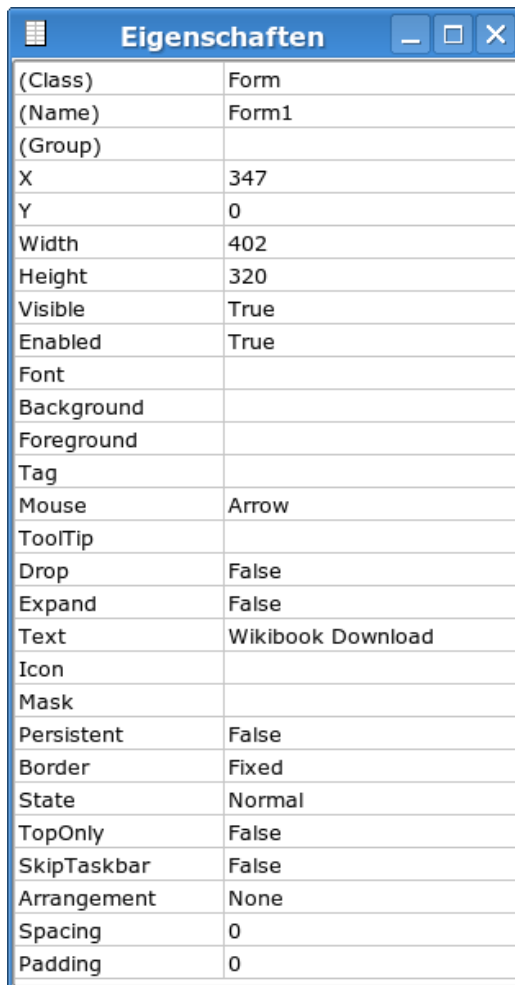


Abbildung 28: Die Werkzeugsammlung (Toolbox)

### 3.6.1 Eigenschaften

Jedes Steuerelement aus der Werkzeugkiste hat verschiedene Eigenschaften (Größe, Farbe, Umrahmung, Name, Beschriftung etc). Eigenschaften (= Properties) können **zur Entwurfszeit**, d.h. vor dem Programmstart festgelegt werden.

- Auf dem Formularentwurf wird das gewünschte Steuerelement markiert.
- Mit F6 wird das Eigenschaftenfenster aufgerufen.



Property	Value
(Class)	Form
(Name)	Form1
(Group)	
X	347
Y	0
Width	402
Height	320
Visible	True
Enabled	True
Font	
Background	
Foreground	
Tag	
Mouse	Arrow
ToolTip	
Drop	False
Expand	False
Text	Wikibook Download
Icon	
Mask	
Persistent	False
Border	Fixed
State	Normal
TopOnly	False
SkipTaskbar	False
Arrangement	None
Spacing	0
Padding	0

Abbildung 29: Das Eigenschaftenfenster

- Dort sind dann alle verfügbaren Eigenschaften aufgelistet.
- Man wählt die gewünschte Eigenschaft.

Probieren Sie dies z. B. mit der Umrahmung der Form, dem Name der Form oder der Größe der Form aus.

Die wichtigsten Eigenschaften von Steuerelementen werden im Folgenden beschrieben.



## Name-Eigenschaft

Diese Eigenschaft gibt es bei allen Objekten. Sie bestimmt den Namen, unter dem das Objekt im Gambas-Code angesprochen wird. Diese Eigenschaft kann nur im Entwurfsmodus, nicht im Programm selber geändert werden. Die Voreinstellung für diese Eigenschaft ist zum Beispiel beim ersten Formular **Form1**. Wenn man sich irgendwo im Code auf dieses Formular beziehen will, z.B. wenn man eine seiner Eigenschaften ändern will, muss man diesen Namen benutzen.

## Text-Eigenschaft

Diese Eigenschaft gibt es bei Formularen, Befehlsschaltflächen, Bezeichnungsfeldern und vielen anderen Steuerelementen. Bei Formularen gibt sie den Text an, der in der Titelleiste des Formulars angezeigt wird. Diese Eigenschaft kann sowohl im Entwurfsmodus als auch im Programm selber (zur Laufzeit) geändert werden. Beispiel:

```
PUBLIC SUB Form_Open()  
    Form.Text = "Neuer Titel"  
END
```

Unter Visual Basic hieß diese Eigenschaft meistens *Caption*.

## Visible Sichtbar oder versteckt

Diese Eigenschaft gibt es bei Formularen und allen Steuerelementen außer dem Timer = Zeitgeber, denn der ist immer zur Laufzeit unsichtbar. Mit dieser Eigenschaft wird festgelegt, ob ein Objekt sichtbar oder ausgeblendet ist. Sie kann sowohl im Entwurfsmodus als auch zur Laufzeit geändert werden. Diese Eigenschaft kann nur die Werte TRUE (engl. true: wahr, das Objekt ist also sichtbar) oder FALSE (engl. false: falsch, das Objekt ist also nicht sichtbar) annehmen.

Beispiel: Sie haben eine Form mit 2 Befehlsschaltflächen. Wenn Sie auf den einen Button drücken, soll der andere verschwinden.

```
PUBLIC SUB Button1_Click()  
    Button2.Visible = FALSE  
END
```

Die Befehlsschaltfläche mit dem Namen Button2 wird vom Bildschirm gelöscht, also unsichtbar gemacht.

### **Enabled Aktiviert oder deaktiviert**

Diese Eigenschaft gibt es für Formulare und alle Steuerelemente. Sie bestimmt, ob die Form oder das Steuerelement auf vom Benutzer erzeugte Ereignisse reagieren kann. Mit dieser Eigenschaft können Formulare und Steuerelemente also zur Laufzeit entweder verfügbar gemacht oder gesperrt werden. Wenn Steuerelemente gesperrt sind, werden sie grau statt schwarz dargestellt und sind nicht mehr anwählbar. Wie die Visible-Eigenschaft kann auch Enable nur die Werte TRUE (das Objekt kann auf Ereignisse reagieren) oder FALSE (das Objekt kann nicht auf Ereignisse reagieren) annehmen.

Man kann also auf diese Weise, ähnlich wie mit Visible, Steuerelemente sperren, die gerade nicht verwendet werden sollen. Man kann damit auch zum Beispiel ein Textfeld sperren, das nur zur Textanzeige gebraucht wird.

Wenn man bei einem Zeitgeber Enabled auf FALSE setzt, dann wird er angehalten und löst keine Ereignisse mehr aus, bis man ihn wieder weiterlaufen lässt.

Beispiel: Der Befehlsknopf Button2 wird stumm geschaltet.

```
PUBLIC SUB Button1_Click()  
    Button2.Enabled = FALSE  
END
```

### **BackColor Hintergrundfarbe**

BackColor legt die Hintergrundfarbe für ein Objekt fest und kann sowohl im Entwurfsmodus als auch zur Laufzeit geändert werden. Beispiel:

```
PUBLIC SUB Form_Open()  
    Form1.BackColor = &00FF00  
END
```

ergibt einen gelben Hintergrund im Formular Form1.

Wenn man die BackColor-Eigenschaft im Eigenschaftenfenster ändert, kann man eine von vielen vorgegebenen Farben aussuchen. Wenn die nicht ausreichen, kann man im Farbpalettenfenster eigene Farben „mischen“, und die werden dann auch angezeigt. Siehe auch den Abschnitt über Grafikbefehle.

Man kann die Backcolor auch über einen Standarddialog in der Laufzeit wählen.

### **ForeColor Vordergrundfarbe**

In Formularen und manchen Steuerelementen kann man mit dieser Eigenschaft die Farbe einstellen, mit der Buchstaben und Zeichen dargestellt werden. Wenn man

diese Eigenschaft zur Laufzeit ändert, erscheinen nur diejenigen Buchstaben, die danach geschrieben werden, in der neuen Farbe.

### **X, Y entspricht den Koordinaten der Position**

Diese Eigenschaften gibt es bei Formularen und bei fast allen Steuerelementen. Sie bestimmen die Koordinaten der linken oberen Ecke des Objekts, und zwar bei Formularen relativ zum Bildschirm, und bei den anderen Steuerelementen relativ zur linken oberen Ecke des Formulars, in dem sie stehen.

Die beiden Eigenschaften können einerseits im Entwurfsmodus geändert werden: entweder dadurch, dass man das Objekt einfach mit der Maus woanders hin zieht, oder indem man sie direkt im Eigenschaftenfenster verändert. Andererseits kann man sie zur Laufzeit ändern und damit das Objekt verschieben (- außer natürlich beim Zeitgeber).

### **Width, Height entspricht Breite und Höhe**

Diese Eigenschaften bestimmen die Breite und die Höhe eines Objekts. Auch sie können sowohl im Entwurfsmodus als auch zur Laufzeit geändert werden. Die Größe des Objekts ändert sich dann sofort.

Bei einem Formular gehören zur Höhe und Breite auch die Fensterränder oben, unten und an den Seiten dazu. Bei Steuerelementen wird raffinierterweise vom Mittelpunkt des Steuerelementrahmens aus gemessen, damit Steuerelemente mit verschiedenen Rahmenbreiten hübsch ausgerichtet werden können.

## **3.6.2 Änderung zur Laufzeit**

Eigenschaften können auch **zur Laufzeit verändert** werden. Die Eigenschaften werden dann erst nach dem Programmstart im Programmablauf geändert.

Die allgemeine Befehlsform lautet dazu:

```
Name_Des_Steuerelementes.Eigenschaftsname = Eigenschaftswert
```

Beispiel:

Sie haben eine `DrawingArea` auf Ihrer Form platziert. Jetzt wollen Sie während der Laufzeit die Farbe ändern. Platzen Sie noch einen Befehlsbutton dazu, um die Eigenschaft während der Laufzeit zu ändern:

```
PUBLIC SUB Button1_Click()  
    DrawingArea1.BackColor = &H00FF00&  
END
```

Die DrawingArea wird Grün.

Als Übung können Sie zusätzlich die Größe der DrawingArea ändern. Wie geht das?

```
DrawingArea1.Width = 200  
DrawingArea1.Height = 200
```

Das Programm schaut dann so aus:

```
PUBLIC SUB Button1_Click()  
    DrawingArea1.BackColor = &H00FF00&  
    DrawingArea1.Width = 200  
    DrawingArea1.Height = 200  
END
```

Die englischen Steuerelement\_Namen werden automatisch von Gambas zugeteilt und mit einer laufenden Nummer unterscheidbar gemacht (Button1, Button2 etc). Diese automatisch vergebenen Namen können mit der Name-Eigenschaft im Eigenschaftsfenster verändert werden.

### 3.6.3 Methoden

Methoden werden ähnlich wie Eigenschaften verwendet. Sie werden vom Programm aufgerufen. Methoden sind kleine Programme, die Aktivitäten des betreffenden Steuerelements abarbeiten.

Nehmen Sie die Zwischenablage als Beispiel. Sie hat 3 verfügbare Methoden: Copy, Paste und Clear

Wenn im Codefenster programmiert wird, erscheint hinter dem Namen einer Komponente nach Schreiben des Punktes eine Liste der verfügbaren Eigenschaften und Methoden.

Eine Auswahl per Doppelklick erleichtert einem hier die Arbeit.

### 3.6.4 Ereignisse

Ereignisse sind z. B. Mausklicks, Mausbewegungen oder ein Tastendruck.

Das Gambasprogramm kann auf solche Ereignisse reagieren, wenn man es wünscht. Man muss nur eine Methode festlegen, die dem Programm sagt, was es tun soll.

Beispiele von Ereignissen:

Der Benutzer klickt auf einen Button oder ein Menü	Click-Ereignis
Der Benutzer ändert einen Text in einem Textfeld	Change-Ereignis
Ein Programm wird beendet, weil irgendwo ein <i>Ende-Befehl</i> erreicht wird (Benutzer oder Programmablauf), das Fenster wird aus dem Speicher entladen	Unload-Ereignis
Ein Timer erreicht einen vorher bestimmten Wert	Timer-Ereignis
Ein Fenster wird in der Größe verändert	Resize-Ereignis

Wenn man mit der **rechten Maustaste** eine Komponente im Form Editor auswählt und dann auf *Ereignis* (= Event) klickt, werden alle Ereignisse angezeigt, die für diese Komponente verfügbar sind. Wählen Sie die, die Sie brauchen.

Ereignisse, auf die ein Steuerelement reagieren soll, werden zur Entwurfszeit festgelegt.

Liste der Ereignisse:

- Click = Click mit der Maus (Ereignis wird nicht beim Drücken der Maustaste, sondern erst beim Wiederloslassen ausgelöst!)
- DbClick = Doppelclick
- Drag = Ziehen mit der Maus
- DragMove
- Draw
- Drop = Fallen lassen, nach dem Ziehen
- Enter = Mit der Maus in eine Fläche hinein fahren
- GotFocus = Die Markierung (bzw den Focus) bekommen
- KeyPress = Taste gedrückt
- KeyRelease
- Leave = Verlassen einer Fläche

- LostFocus
- Menu
- MouseDown = Drücken einer beliebigen Maustaste
- MouseMove
- MouseUP
- MouseWheel
- Open

Das Click-Ereignis funktioniert auch im Entwurfsmodus. Mit ihm kann man das Steuerelement markieren. Durch einen Doppelklick öffnet sich das zugehörige Codefenster. Automatisch wird der Rahmen für ein Unterprogramm mit SUB und END erzeugt und dabei das häufigste Ereignis eines Steuerelementes bereits zugewiesen.

Element	Standardereignis
Form	Open
Button	Click
TextBox	KeyPress
ScrollView	MouseDown
DrawingArea	Draw
Timer	Timer
Label	MouseDown
PictureBox	MouseDown
ProgressBar	MouseDown
CheckBox	Click

In diesen bereitgestellten Rahmen gibt man wie üblich seinen Programmcode ein.

### **3.7 Befehlsknopf (Commandbutton, Button)**

Siehe [Gambas: Button](#)

### **3.8 Textarea**

Siehe [Gambas: Textarea](#)

## 3.9 Progressbar = Fortschrittsbalken

Wie kann man den Fortschrittsbalken programmieren? Dies sollen die folgenden Beispiele zeigen:

### 3.9.1 Allein

Sie brauchen eine ProgressBar und drei Befehlsbuttons um das Programm in Gang zu bekommen. Holen Sie sich diese Elemente mit F6 und einem Doppelklick aus der Werkzeugkiste auf Ihre Form.

Der Quellcode:

```
STATIC PUBLIC SUB Main()  
    hForm AS Fmain  
    hForm = NEW Fmain  
    hForm.show  
END  
  
PUBLIC SUB _new()  
    ProgressBar1.value = 0.01  
    'Der Wert für value muss zwischen 0 und 1 liegen  
END  
  
PUBLIC SUB Button1_Click()  
    ProgressBar1.value = ProgressBar1.value - 0.01  
END  
  
PUBLIC SUB Button2_Click()  
    ProgressBar1.Reset  
END  
  
PUBLIC SUB Button3_Click()  
    ProgressBar1.value = ProgressBar1.value + 0.01  
END
```

### 3.9.2 Zusammen mit dem Timer

Das folgende Beispiel nutzt den Progressbar zusammen mit dem Timer. Je nach Delay des Timers kann man die Geschwindigkeit variieren und damit z. B. einen Ladevorgang anzeigen.

Sie brauchen einen Progressbar und einen Timer, um das Programm in Gang zu bringen:

Der Code schaut so aus:

```
PUBLIC SUB Form_Open()  
    'Timer einstellen  
    timer1.Delay = 100  
    timer1.enabled = TRUE  
    'Progressbar einstellen  
    ProgressBar1.Value = 0.01  
END  
  
PUBLIC SUB Timer1_Timer()  
    IF ProgressBar1.value < 0.99 THEN  
        ProgressBar1.Value = ProgressBar1.Value + 0.01  
    END IF  
END
```

### 3.10 Der Timer (Zeitgeber, Zeituhr)

Siehe [Gambas: Timer](#)

### 3.11 Die Spaltenansicht (ColumnView)

In der Werkzeugbox findet sich ein Steuerelement mit dem Namen ColumnView. Wie man es nutzen kann zeigt folgendes Programm. Man braucht eine Form mit einem ColumnView Steuerelement auf der Form, um es in Gang zu bringen.

```
PUBLIC SUB Form_Open()  
    Create()  
END  
  
PUBLIC SUB Create()  
    it AS Integer  
    it2 AS Integer  
    nLines AS Integer  
    ColumnView1.Clear  
    ColumnView1.Width = 776  
    ColumnView1.Height = 464  
  
    '4 Spalten (Columns) erzeugen  
    ColumnView1.Columns.Count = 4  
  
    'Die Breite (width) der Spalten einstellen  
    FOR it = 0 TO 3  
        ColumnView1.Columns[it].Width = ColumnView1.Width / ColumnView1.Columns.Count  
        ColumnView1.Columns[it].Text=" Title " & it  
    NEXT  
  
    '5 Zeilen (rows) erzeugen  
    nLines = 5  
    FOR it = 0 TO 4  
        ColumnView1.Add(it, it)
```



```
FOR it2 = 0 TO ColumnView1.Columns.Count - 1
    ColumnView1[it][it2] = "Col: " & it2 & " Row: " & it
NEXT
NEXT
END
```

## 3.12 Der Schieberegler

Siehe [Gambas: Scrollbar](#)

## 3.13 Die Form

Siehe [Gambas: Form](#)

## 3.14 Das Gitter (Gridview)

Siehe [Gambas: Gitter](#)

## 3.15 Tableview (Tabellenansicht)

Siehe [Gambas: Tabelle](#)

## 3.16 Listview (Listenansicht)

Siehe [Gambas: Liste](#)

## 3.17 Der Command Button (Befehlsknopf)

Der Commandbutton ist einer der wichtigsten Steuerelemente überhaupt. Die meisten kleinen Beispielprogramme in diesem Gambas Wikibook werden über einen Commandbutton gestartet.

So schaut der Befehlsknopf auf der Werkzeugleiste aus. Klicken Sie in Ihrer Werkzeugleiste darauf, wenn Sie einen auf Ihrer Form brauchen. Die Werkzeugleiste bekommen Sie mit **F6**.



Abbildung 30: Bild:Button.png

Halten Sie dabei folgende Reihenfolge ein. Erst die Form anzeigen, dann die Werkzeugkiste aktivieren, sonst bekommt man keinen Commandbutton auf die Form.

So schaut dann der Befehlsknopf auf Ihrer Form in der Entwurfsansicht aus.



Abbildung 31: Bild:Button1.png

Wenn Sie den Befehlsknopf in der Entwurfsansicht anklicken, dann wird er mit 4 kleinen weißen Quadraten an den Ecken markiert. Sie können ihn größer oder kleiner machen und natürlich irgendwohin auf ihrer Form verschieben.

Mit **F4** können Sie die Eigenschaften des Buttons festlegen. Lassen Sie sich nicht von der großen Zahl beeindrucken. Probieren Sie z.B. einmal die Eigenschaft *Text* aus und geben Sie ihrem Button einen neuen Namen.

Wenn Sie Code an den Button anhängen wollen, dann öffnen Sie das Codefenster mit einem Doppelklick auf den Commandbutton.

Die Handhabung des Commandbuttons ist im Prinzip also sehr einfach. Man holt ihn aus der Werkzeugkiste auf die Form, benennt ihn um und schreibt dann den entsprechenden Code dahinter:

```
PUBLIC SUB Button1_Click()  
    Print 1  
END
```

### 3.17.1 Aufruf des Befehlsknopfes mit der Tastatur

Ist der Knopf bei laufendem Programm im Fokus, reagiert er auf das 'KeyPress'-Ereignis.

Der Code zur Behandlung dieses Ereignisses ist dann z.B.:

```
PUBLIC SUB Button1_KeyPress()  
  IF key.Code = key.Space OR key.Code = key.Return OR key.code=key.Enter THEN  
    Print 1  
  ENDF  
END
```

Ein paar Besonderheiten des Commandbuttons sollen folgende Programme zeigen.

### 3.17.2 Den Commandbutton verschwinden lassen

Mit dem folgenden Miniprogramm können Sie einen Commandbutton unsichtbar machen:

```
PUBLIC SUB Button1_Click()  
  Button1.visible = false  
END
```

Dasselbe macht übrigens:

```
PUBLIC SUB Button1_Click()  
  Button1.hide  
END
```

Wenn er wieder auftauchen soll, hilft folgender Code mit einem Button2:

```
PUBLIC SUB Button2_Click()  
  Button1.show  
END
```

Wie könnte man den Button1 wieder anzeigen, ohne das man einen Button1 nutzt.

Beispielsweise mit einem Doppelklick auf die Form:

```
PUBLIC SUB Form_DblClick()  
  Button1.show  
END
```

### 3.17.3 Den Commandbutton endgültig verschwinden lassen

Mit dem folgenden Miniprogramm können Sie einen Commandbutton völlig verschwinden lassen.

```
PUBLIC SUB Button1_Click()  
  Button1.delete  
END
```

### 3.17.4 Den Commandbutton verschieben

Wollen Sie den Commandbutton während der Laufzeit verschieben, dann können Sie folgendes Miniprogramm nutzen. Sie brauchen einen Commandbutton auf der Form um es in Gang zu bringen.

```
PUBLIC SUB Form_Open()
    Button1.Move(160,64,64,32)
END
PUBLIC SUB Button1_Click()
    Button1.Move(1,64,64,32)
END
```

Wie kann man den Button mit der Maus verschieben?

Wollen Sie einmal erleben wie ein Button Ihrer Maus folgt wie ein Hündchen?

Dann probieren Sie folgendes Programm. Sie brauchen eine Form und einen Button, um es in Gang zu bringen.

```
PUBLIC SUB Form_MouseMove()
    Button1.X = Mouse.X
    Button1.Y = Mouse.Y
END
```

Wenn Sie das Programm gestartet haben, dann klicken Sie in die Form und fahren mit der Maus auf der Form herum.

### 3.17.5 Einen Commandbutton zur Laufzeit neu erstellen

Meistens wird man alle notwendigen Befehlsknöpfe bereits während der Entwurfszeit auf der Form platzieren, sie mit Code versehen und dann erst das Programm starten. Man kann aber auch während der Laufzeit einen oder mehrere Commandbuttons noch während des Programmablaufes (Laufzeit) neu erstellen. Wie das geht ist hier beschrieben.

Sie brauchen nur eine leere Form um es in Gang zu bringen.

```
b AS Button
PUBLIC SUB _New()
    b = NEW Button(ME) AS "Ende"
    b.show
    b.Caption = "Ende"
    b.x = 100
    b.y = 100
END

PUBLIC PROCEDURE Ende_Click()
    ME.Close
END
```

Eine lustige Variante:

```
b AS Button
PUBLIC SUB _New()
  b = NEW Button(ME) AS "Ende"
  b.show
  b.Caption = "Ende"
  b.x = 100
  b.y = 100
END

PUBLIC PROCEDURE Ende_Click()
  b.visible = FALSE
END

PUBLIC SUB Form_Leave()
  b.visible = TRUE
END
```

### 3.17.6 Commandbuttons zur Laufzeit in einer Schleife neu erstellen

Will man viele Buttons auf einer Form schnell erstellen, ist dazu eine Schleife am besten geeignet. Das Ganze hat aber ein paar Haken. Mit folgendem Beispielprogramm funktioniert es dann doch ganz gut. Man braucht nur eine leere Form, um es in Gang zu bringen.

```
btn AS Button
btnname AS String

PUBLIC SUB _new()
  x AS Integer
  FOR x = 1 TO 4
    btnname = "btn" & Str(x)
    btn = NEW Button(ME) AS btnname
    btn.X = 100 * x
    btn.Y = 30
    btn.Width = 50
    btn.Height = 30
    btn.Caption = Str(x)
    btn.enabled = TRUE
  NEXT
END
```

Will man die Buttons in einer Schleife alle wieder weglöschen, dann geht das schon bedeutend schwieriger.

### 3.17.7 Der wandernde Befehlsknopf

Völlig nutzlos ist das folgende Programm, aber es ist Bewegung im Spiel. Sie brauchen eine Form und einen Button um es in Gang zu bringen.

```

PUBLIC SUB Button1_Click()
  x AS Integer
  FOR x = 1 TO 20
    Button1.X = x * 10
    Button1.Y = x * 10
    WAIT 0.5
    Button1.Text = Str(x)
  NEXT
END

```

### 3.18 Zahlenraten bis 27

Bei dem folgenden kleinen Spiel denkt sich der Computer eine Zahl zwischen 1 und 27. Der Benutzer soll diese Zahl mit möglichst wenig Versuchen erraten. Versuchen Sie mit maximal 5 Rateversuchen auszukommen!

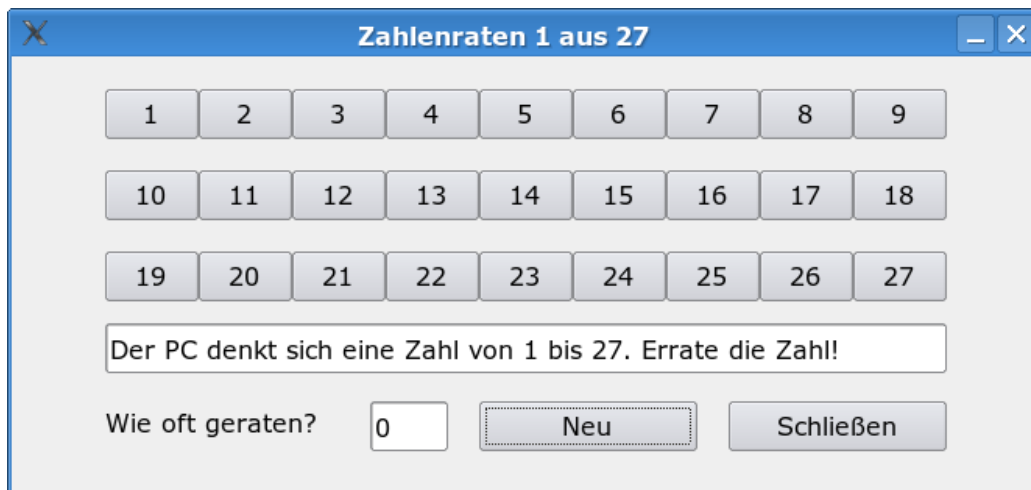


Abbildung 32: Bild:Gambaszahlenraten.png

Sie brauchen 2 Befehlsbuttons, 2 Textboxen und 1 Labelsteuerelement auf Ihrer Form, um das Programm in Gang zu bringen.

Quelltext: [Zahlenraten Form1.class](#)

Die 27 Zahlenbuttons werden zur Laufzeit erzeugt. Erstaunlich ist der Array aus Buttons. Damit wir nicht für alle 27 Buttons eine eigene **Click()** Funktion schreiben müssen, bekommen alle Buttons in der Zeile

```
b[n] = NEW Button(ME) AS "btn"
```

den gleichen Namen **btn**. Dadurch rufen alle 27 Buttons, wenn sie angeklickt werden, die gleiche Funktion **btn\_Click()** auf. Um nun innerhalb dieser Funktion zu ermitteln, welcher Button angeklickt wurde, benutzen wir **LAST**. **LAST** ist eine Referenz auf das letzte Objekt, das ein Ereignis ausgelöst hat. Durch **LAST** können wir also auf das entsprechende Objekt zugreifen. Um dieses jetzt eindeutig zu identifizieren, benutzen wir dessen **Tag**-Eigenschaft. Die **Tag**-Eigenschaft kann verwendet werden, um unsichtbar Informationen in einem Objekt zu speichern, in unserem Fall die dem Button zugeordnete Zahl. Innerhalb der Funktion **btn\_Click()** liefert **LAST.Tag** also den Zahlenwert des angeklickten Buttons.

Die Textarea ist ein häufig genutztes Steuerelement, welches im Gegensatz zur Textbox auch Platz für größere Texte bietet.

## 3.19 Die Eigenschaften, Methoden und Ereignisse der Textarea

Wenn man die Textarea näher kennenlernen will, kann man sich die

- die Eigenschaften (properties)
- die Methoden (methods)
- und die Ereignisse (events)

der Textarea am besten in der Hilfefunktion von Gambas ansehen.

Man drückt die F1 oder wählt im Projektfenster aus dem Menü den Menüpunkt ? (Fragezeichen), dort nimmt man den ersten Menüpunkt *Hilfe Browser F1*. (Siehe auch [Gambas: Hilfe](#)) Dann wählt man auf der Seite *Gambas Documentation* den Punkt **Components**. Aus den Components wählt man sich die Komponente **gb.qt** aus und sucht dort nach der Klasse **Textarea**. Hier findet man alle Eigenschaften, Methoden und Ereignisse der Textarea.

Im Internet findet man das Ganze auch sehr schnell unter: [gambasdoc.org](http://gambasdoc.org)

### 3.19.1 Eigenschaften (Properties) der Textarea

BackColor, Background, Border, Column, Cursor, Design, Drop, Enabled, Expand, Font, ForeColor, Foreground, H, Handle, Height, Id, Left, Length, Line, Mouse, Next, Parent, Pos, Previous, ReadOnly, ScreenX, ScreenY, Select, Selection, Tag, Text, ToolTip, Top, Visible, W, Width, Window, Wrap, X, Y







Deswegen schalten Sie seinen Eigenschaft Enabled im Eigenschaftsfenster (F4) auf TRUE.

Dann klicken Sie auf den Timer und geben Sie folgenden Code ein:

```
PUBLIC SUB Timer1_Timer()
    PRINT 1
END
```

Programmablauf:

Je nach dem wie groß man den Wert für das Delay einstellt, um so schneller oder langsamer erfolgt die Printausgabe. Diese wird immerwieder wiederholt.

Variation: Die Ausgabe wird mit einem Button gestoppt.

```
PUBLIC SUB Timer1_Timer()
    PRINT 1
END

PUBLIC SUB Button1_Click()
    Timer1.Enabled = FALSE
END
```

### 3.23 Delay zur Laufzeit ändern

Am Anfang steht der Timer immer auf einem Delay von 1000 Millisekunden und ist *Disabled* (Enabled = FALSE). Er muss also erst angeschaltet werden.

Diese Voreinstellungen kann man bei den Eigenschaften ändern oder auch programmieren.

Starten Sie ein neues Projekt Timer1. Holen Sie sich eine neue Form mit der Rechten Maustaste auf Form im Projekt. Nennen Sie die Form wie Sie wollen. Deklarieren Sie die Form als Startform.

Man kann die Zeitverzögerung im Timer in Millisekunden festlegen. Wenn der Timer mit der *Enabled*-Eigenschaft angeschaltet ist, dann startet die Methode Timer1\_Timer() periodisch mit der vorgegeben Zeitverzögerung.

Platzieren Sie auf der Form einen Timer und 2 Befehlsbuttons durch Klick auf die entsprechenden Felder der Werkzeugkiste.

Das Ganze schaut dann so aus: Verschieben Sie ihre Form so, dass Sie das Direktfenster sehen!

Der Programmcode dazu:

```
' Gambas class file

PUBLIC n AS Integer
```

```
PUBLIC SUB Form_Open()  
    Timer1.Enabled = TRUE  
END  
  
PUBLIC SUB Timer1_Timer()  
    PRINT Timer1.Delay  
    Timer1.Delay = 1000 + n  
END  
  
PUBLIC SUB Button1_Click()  
    n = n + 200  
END  
  
PUBLIC SUB Button2_Click()  
    n = n - 200  
END
```

**Programmablauf:** Der Delay-Wert kann jetzt schneller oder langsamer geschaltet werden

**Variation:**

Wenn Sie ihrem Formular noch eine TextArea hinzufügen, dann erfolgt die Anzeige in der TextArea, wenn Sie folgende Codevariation nutzen.

```
' Gambas class file  
  
PUBLIC n AS Integer  
  
PUBLIC SUB Form_Open()  
    Timer1.Enabled = TRUE  
END  
  
PUBLIC SUB Timer1_Timer()  
    TextArea1.Text = Timer1.Delay & "\n" & TextArea1.Text  
    Timer1.Delay = 1000 + n  
END  
  
PUBLIC SUB Button1_Click()  
    n = n + 200  
END  
  
PUBLIC SUB Button2_Click()  
    n = n - 200  
END
```

## 3.24 Das Delay mit Zufallszahlen verändern

Im folgenden kleinen Programm werden Zufallszahlen zwischen 1 und 25 erzeugt. Diese Zahlen werden im Direktfenster ausgegeben und gleichzeitig wird das Delay des Timers je nach Größe der Zahl verändert. Um das Programm in Gang zu bringen brauchen Sie eine Form und darauf ein Timer-Steuerelement. Dieses holen Sie sich mit F6 aus der Toolbox.

```

PUBLIC SUB Form_Open()
    Timer1.Enabled = TRUE
END

PUBLIC SUB Timer1_Timer()
    x AS Integer
    Randomize
    x = Int(Rnd(1,25))
    PRINT x
    Timer1.Delay = x * 50
END

```

Ziemlich nutzlos dieses Programm, aber man kann sich mit dem Timer vertraut machen und ein bisschen mit Zufallszahlen herumspielen.

## 3.25 Countdown

Dieses Programm stammt von Jonás Alvarez. Siehe: [linuxfocus.org](http://linuxfocus.org)

Es wurde ins Deutsche übertragen, verbessert und kommentiert:

Sie brauchen einen Timer, eine TextBox, einen ToggleButton, ein Labelfeld und 2 Commandbuttons, um es in Gang zu bekommen.

Der Code:

```

CONST fStandart AS Float=20.0
fSekunden AS Float
'fSekunden wird schon außerhalb aller Unterprogramme definiert,
'so kann es in allen UPs benutzt werden.
'fSekunden ist die aktuelle Zeit in Sekunden bis zum Start
'fStandart ist die Zeit des Countdownbeginns.
'Sie ist als Konstante festgelegt.
'fStandart kann aber über den Reset-Commandbutton Button1_Click()
'und die Textbox verändert werden.

PUBLIC SUB Form_Open()
    'bereits beim Programmstart wird einiges festgelegt und angezeigt
    fSekunden = fStandart
    Anzeige
    ToggleButton1.Value = FALSE
    AnAus
    Textbox1.Text = fSekunden
END

PUBLIC SUB ToggleButton1_Click()
    'das ist der wichtigste Knopf
    'er startet den Countdown oder unterbricht ihn
    Timer1.Enabled = Togglebutton1.Value
    'Ist der Togglebutton gedrückt kann auch der Timer laufen.
    'ToggleButton1.Value = True >> Timer1.Enabled = True
    'Ist er nicht gedrückt steht der Timer still
    'ToggleButton1.Value = False >> Timer1.Enabled = False
    AnAus

```

```
END

PUBLIC SUB Button1_Click()
    fSekunden = Val(Textbox1.Text)
    Anzeige
END

PRIVATE SUB AnAus()
    IF ToggleButton1.Value THEN
        ToggleButton1.Text = "&Countdown läuft"
    ELSE
        ToggleButton1.Text = "&Unterbrechung"
    ENDIF
    'wenn der Togglebutton angeklickt wurde erscheint als Text:
    'Countdown läuft.
    'ist er nicht angeklickt erscheint der Text Unterbrechung
END

PRIVATE SUB Anzeige()
    DIM fMinuten AS Integer
    fMinuten = Int(Int(fSekunden) / 60)
    'der Befehl INT schneidet die Nachkommazahlen weg
    'das passiert hier gleich 2 mal
    'erst werden die hundertstel Sekunden gekappt
    'dann wird nach Minuten umgerechnet und auch hier alles nach dem Komma gekappt.
    Label1.Caption = fMinuten & ":" & Format(fSekunden - fMinuten * 60, "00.0")
    'der Formatbefehl bietet viele Möglichkeiten.
    'Schauen Sie ihn sich einmal mit F1 in der Hilfe an
END

PUBLIC SUB Timer1_Timer()
    IF fSekunden < 0.1 THEN
        Label1.Caption = "0:00,0"
        ToggleButton1.Value = FALSE
        ToggleButton1_Click
        'Hier wird aus dem Programm heraus der Togglebutton abgeschaltet
        'da der Countdown kurz vor der Null ist
        'das Programm springt zum Unterprogramm ToggleButton1_Click
    ELSE
        fSekunden = fSekunden - 0.1
        Anzeige
    END IF
END

PUBLIC SUB Button2_Click()
    ME.Close
END
```

Drückt man auf den ToggleButton beginnt die Uhr rückwärts zu zählen. Drückt man noch einmal darauf, wird der Countdown unterbrochen. Mit Reset kann man über die Textbox den Beginn des Countdown verändern.

Das Programm hat 2 Prozeduren AnAus und Anzeige.

## 3.26 Reaktionszeit testen

Mit diesem kleinen Programm wird die Reaktionszeit gemessen.

Für dieses Programm wird die Timer-Funktion benutzt. Diese sollte man nicht mit dem Timersteuerelement verwechseln. Außerdem wird die Anzeige Bremsen erst nach einer anfänglichen Zufallszeit ausgegeben.

Sie brauchen 3 Befehlsbuttons und eine Textbox um das Programm in Gang zu bringen. Das Programm ist noch verbesserungsbedürftig, aber es funktioniert.

Programmcode:

```
fStart AS Float

PUBLIC SUB Button1_Click()
    fZufall AS Float
    i AS Integer

    Randomize()

    fZufall = Rnd(0.5, 3)
    'PRINT fZufall
    WAIT fZufall

    TextBox1.Text = "Bremsen!!!"

    fStart = Timer 'Anfangszeit festlegen.
    'erst ab jetzt wird die Reaktionszeit gemessen
END

PUBLIC SUB Button2_Click()
    fReaktionszeit AS Float

    fReaktionszeit = Timer - fStart
    fReaktionszeit = Int(fReaktionszeit * 1000)
    TextBox1.Text = fReaktionszeit
    'TextBox1.Text = "Sie haben " + Str$(fReaktionszeit) + " Millisekunden gebraucht, um zu bremsen!"

END

PUBLIC SUB Button3_Click()
    ME.Close
END
```

## 3.27 Der Wait-Befehl, eine Alternative zum Timer

In einer Schleife ist es gar nicht so einfach den Timer korrekt zu programmieren. Versuchen Sie es einfach einmal mit dem *WAIT* Befehl als Alternative.

Die Syntax des Befehls lautet:

```
WAIT [ Delay ]
```

Der Befehl ruft die Ereignisschleife auf. Wenn für Delay ein Wert angegeben wird, dann wartet der Computer bis die angegebene Zeitverzögerung verstrichen ist. Die Zahl Delay ist eine Fließkommazahl. Wenn Sie also 100 ms warten wollen, dann geben Sie folgendes ein:

```
WAIT 0.1
```

Während der Pause werden keine Maus- oder Tastatureingaben verarbeitet. Nur Zeichnungs-, Timer- und Dateibeschiebungsereignisse werden noch ausgeführt.

Ein kleines Miniprogramm zeigt die Nutzung des Befehls. Sie brauchen einen Befehlsbutton und eine Textbox um es in Gang zu bringen. Sie müssen 5 Sekunden warten bis etwas passiert.

```
PUBLIC SUB Button1_Click()  
    WAIT 5  
    'wartet 5 Sekunden  
  
    WAIT 0.1  
    'wartet 100 Millisekunden  
  
    TextBox1.Text = ""  
END
```

Oder was sagen Sie zu diesem Programm mit dem wandernden Befehlsknopf: Sie brauchen eine Form und einen Button um es in Gang zu bringen.

```
PUBLIC SUB Button1_Click()  
    DIM x AS Integer  
    FOR x = 1 TO 20  
        Button1.X = x * 10  
        Button1.Y = x * 10  
        WAIT 0.5  
        Button1.Text = Str(x)  
    NEXT  
END
```

## 3.28 ScrollBar (Schieberegler)

Mit dem Schieberegler kann man sehr schön kontinuierliche Zahlenwerte eingeben und ausgeben.

Den Schieberegler muss man sich aus der QT Werkzeugkiste holen, da er nicht in der normalen Gambas Werkzeugkiste zu finden ist.

Dazu lädt man die **gb.qt.ext-Komponente**. Dies kann man über den Menüpunkt Projekt -> Eigenschaften des Gambas-Projektfensters erreichen. In der Komponentenanzeige wählt man die Komponente gb.qt.ext zusätzlich aus.



In der Werkzeugbox erscheint dann eine Schaltfläche mit der Bezeichnung QT. Diese klickt man an und hat dann eine Reihe weiterer Steuerelemente darunter den ScrollBar zur Verfügung.

### 3.29 Beispielprogramm

Das Beispielprogramm ist sehr einfach. Man braucht den Schieberegler und eine TextBox auf der Form.

Betätigt man den Schieberegler wie gewohnt mit der Maus, dann wird ein numerischer Wert zwischen 0 und 99 in der Textbox angezeigt. Verändert sich die Scrollbar (Change-Ereignis), dann wird der Scrollbarwert (Value) an die Textbox übergeben.



Abbildung 33: Screenshot Scrollbar

Hinter dem Schieberegler steht folgender Code:

```
PUBLIC SUB ScrollBar1_Change()  
    TextBox1.Text = ScrollBar1.Value  
END
```

Siehe auch <http://www.madeasy.de/7/prgmini.htm#form>

### 3.30 Wie bekommt man eine neue Form?

In der Gambas-IDE geht man mit der rechten Maustaste auf *Formulare*. Dann wählt man *Neu* und *Formular*. Meist nutzt man die gewählte Form auch als Start-

form und muss dies dann mit dem kleinen grünen Pfeil angeben. Oder aber man hat bereits eine Startform oder Startklasse, dann wählt man einfach eine Form ohne den grünen Pfeil.

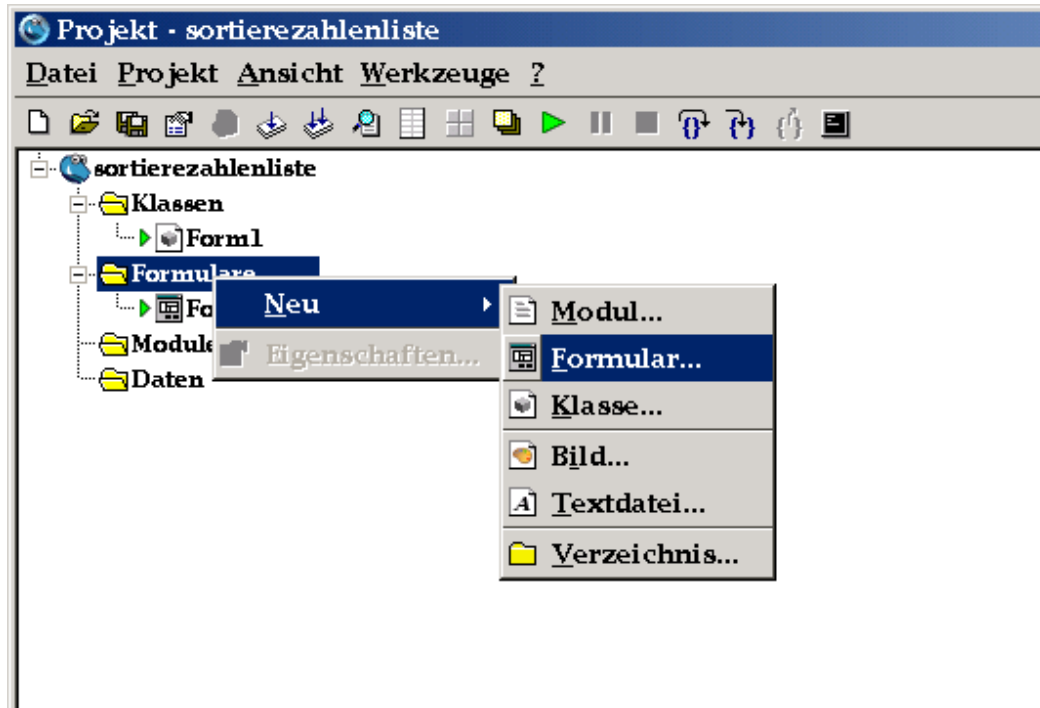


Abbildung 34: Bild:gambasformneu.png

### 3.31 Einfaches zum Thema Form

Im folgenden werden ein paar Dinge zu einer Form erklärt, die man immer wieder braucht.

Starten eines Programmes direkt beim Öffnen der Form

```
Form_Open()
```

Den Namen einer Form ändern mit  
 ME.text = „neuer Titel“

Schließt eine Form

```
ME.Close
```

Startet ein Programm wenn man mit der Maus darauf kommt

```
Form_Enter()
```

## 3.32 Fragen

Wie kann man 2 Formulare gleichzeitig darstellen?

```
STATIC PUBLIC SUB Main()  
    Form1.Show  
    Form2.Show  
END
```

oder eben noch mehr als 2 gleichzeitig

```
STATIC PUBLIC SUB Main()  
    Form1.Show  
    Form2.Show  
    Form3.Show  
    Form4.Show  
    ....  
END
```

Wie kann man 2 Formulare gleichzeitig darstellen, ohne dass sie sich überlappen?

Formular 2 liegt rechts neben Formular 1:

```
STATIC PUBLIC SUB Main()  
    Form1.Show  
    Form2.X = Form1.X + Form1.ClientWidth  
    Form2.Y = Form1.Y  
    Form2.Show  
END
```

Formular 2 liegt unterhalb Formular 1:

```
STATIC PUBLIC SUB Main()  
    Form1.Show  
    Form2.X = Form1.X  
    Form2.Y = Form1.Y + Form1.ClientHeight  
    Form2.Show  
END
```

Eine leichte Überlappung ist noch vorhanden; Ursache ist vermutlich, dass die Fensterrahmenbreite und -höhe nicht bei ClientWidth bzw. ClientHeight mitgezählt wird.

## 3.33 Startform

Die Startform erhält von Gambas automatisch die Main Class zugeordnet. Dies erleichtert anfangs die Programmierung, erschwert aber den Umgang mit mehreren Formen.

### 3.34 Modale Form

Wenn man eine modales Formular aufruft, werden automatisch alle anderen Fenster passiv gestellt. Man kann also nur noch auf dem modalen Formular weitermachen.

Die Programmausführung des aufrufenden Formulars wird an der Stelle des Aufrufes des modalen Formulars eingestellt, bis das modale Formular beendet wird (wird ein nichtmodales Formular aufgerufen, geht die Programmausführung des aufrufenden Formulars parallel weiter).

Der Aufruf eines modalen Formulars erfolgt mit:

```
Form2.ShowModal
```

Hier gibt es leider einen kleinen Schönheitsfehler: Das aufrufende Formular wird zwar passiv, es lässt sich nicht mehr bedienen, die Bedienelemente sind aber nicht 'ausgegraut' und suggerieren dem Benutzer die Bedienbarkeit (Gambas Version 1.0.15). Abhilfe erreicht man durch folgenden Code:

```
ME.Enabled = False  
Form2.ShowModal  
ME.Enabled = True
```

Die 3. Zeile ist nötig, damit nach Beendigung des modalen Dialogs das aufrufende Formular wieder bedienbar wird.

Dieses gilt auch für die vordefinierten modalen Dialoge

```
message  
message.Info  
message.Error  
message.Delete  
message.Warning  
message.Question
```

Dieser Abschnitt muss noch ausführlich bearbeitet werden.

Siehe auch: <http://www.binara.com/gambas-wiki/static/Gambas/ModalForm.html>

### 3.35 Eigenschaften eines Formulars

Arrangement BackColor Background Border Caption Children ClientH ClientHeight ClientW ClientWidth ClientX ClientY Cursor Design Drop Enabled Expand Font ForeColor Foreground H Handle Height Icon Id Left Mask Menus Modal Mouse Next Padding Parent Persistent Previous ScreenX ScreenY Skip-Taskbar Spacing State Tag Text Title ToolTip Top TopOnly Visible W Width Window X Y

W bzw. Width

liefert die Breite des Formulars, aber abzüglich der Fensterelemente (Rahmen links und rechts)

H bzw. Height

liefert die Höhe des Formulars, aber abzüglich der Fensterelemente (Rahmen oben incl. Überschrift und Rahmen unten)

### 3.36 Methoden einer Form

Activate Close DblClick Deactivate Drag DragMove Drop Enter GotFocus Hide KeyPress KeyRelease Leave LostFocus Menu MouseDown MouseMove MouseUp MouseWheel Move Open Resize Show

### 3.37 Ereignisse einer Form

Activate Close DblClick Deactivate Drag DragMove Drop Enter GotFocus Hide KeyPress KeyRelease Leave LostFocus Menu MouseDown MouseMove MouseUp MouseWheel Move Open Resize Show

### 3.38 Konstanten der Form

Fixed FullScreen Maximized Minimized None Normal Resizable

### 3.39 Neue Form

Sie wollen mit 2 Formen arbeiten und von einer Form zu anderen hin und herspringen:

Beispiel: Sie brauchen 2 Formen und auf jeder Form einen Commandbutton.

Der Code für Form1.class lautet:

```
PUBLIC SUB Button1_Click()  
    Form2.Show  
    Form1.Hide  
END
```

Der Code für Form2.class lautet:

```
PUBLIC SUB Button1_Click()  
    Form1.Show  
    Form2.Close  
END
```

Beachten Sie, dass die beiden Formen nicht gleichberechtigt sind. Die Form2 ist ein Objekt der Startklasse für Form1. Die Form1 ist die sogenannte Hauptklasse (*main class*)

Wenn Sie bei Form1 folgenden Code eingeben:

```
PUBLIC SUB Button1_Click()  
    Form2.Show  
    Form1.close  
END
```

funktioniert das Programm nicht, da es vorzeitig beendet wird. Auch Form2 wird dann nicht angezeigt, da sie von Form1 abhängig ist.

Diese Abhängigkeit kann sich auch auf die Eigenschaften der Form2 auswirken.

Siehe <http://www.madeasy.de/7/prgnewform.htm>

## 3.40 Datenübergabe von einem Formular zu einem anderen Formular

Siehe <http://www.madeasy.de/7/prgformtoformplus.htm> Reichlich kompliziert das Ganze. Vielleicht geht es ja auch noch einfacher.

Es geht einfacher: Siehe das nächste Kapitel mit den globalen Variablen.

## 3.41 Globale Variablen in Gambas

In VB ist es möglich, z.B. aus der Form2 heraus eine Textbox in Form1 direkt anzusprechen. Der Befehl dazu lautet: `Form1!Textbox1.Text = "zb"`. Dies funktioniert in Gambas nicht. In Gambas gibt es keine projektübergreifenden globalen Variablen. Man kann sich behelfen, wenn man eine Klasse erzeugt, die man Global nennt. Die globalen Variablen deklariert man als statische public-Variablen in dieser Klasse. Dann kann man auf sie mit `Global.variablename` im ganzen Projekt zugreifen.

*Minsini: It's still poor programming practice but at least they'll be identified as global variables whenever you use them ;)*

### 3.41.1 Programmbeispiel:

Sie haben 2 Formen und wollen Text zwischen den Formen austauschen. Die globale Variable zum Textaustausch soll einfach t heißen.

- Auf Form1 haben Sie 3 Commandbuttons und eine Textbox.
- Auf Form2 haben Sie 2 Commandbutosn und eine Textbox

Der Code für die Global class (die gleichzeitig Startklasse ist) lautet:

```

STATIC PUBLIC t AS String
STATIC PUBLIC SUB Main()
form1.Show
'Wenn Sie diese Zeile nicht einfügen, wird keine Form angezeigt.
END

```

Der Code für Form1 lautet:

```

PUBLIC SUB Button1_Click()
Textbox1.Text = "Dies ist ein Test"
Global.t = Textbox1.Text
END
PUBLIC SUB Button2_Click()
Global.t = Textbox1.Text
Form2.Show
END
PUBLIC SUB Button3_Click()
Textbox1.Text =Global.t
END

```

Der Code für Form2 lautet:

```

PUBLIC SUB Button1_Click()
Textbox1.Text = Global.t
END
PUBLIC SUB Button2_Click()
Global.t = Textbox1.Text
Form1.Show
'Form2.Close
END

```

Wenn Sie jetzt den Text der Textbox in Form1 ändern, wird er in Form2 übergeben und durch klicken von Button1 eingefügt. Dasselbe passiert in Form2. Auch hier können Sie den Text der Textbox ändern und der geänderte Text wird mit anklicken des Button3 auf Form1 in der Textbox gezeigt.

### 3.41.2 Vereinfachtes Programmbeispiel zum Datenaustausch zwischen Formen

Das ganze geht noch einfacher wie folgendes Beispiel zeigt:

- Sie brauchen eine Global.class, die Startklasse ist.
- In Form1 brauchen Sie eine Textarea und einen Commandbutton.
- In Form2 brauchen Sie eine Textarea und einen Commandbutton.

Code für die Global.class

```
STATIC PUBLIC t AS String
STATIC PUBLIC SUB Main()
form1.show
END
```

Für den Namen Global.class Sie können aber auch den vorgegebenen Namen Class1.class belassen, dann heißt die globale Variable nicht mehr Global.t sonder Class1.t.

Code für Form1

```
PUBLIC SUB Form_Open()
IF NOT Global.t THEN
'Falls Global.t noch leer ist, dann fülle die Textarea mit einem text
textareal.text = "Dies ist ein Anfang"
ELSE
textareal.Text = global.t
ENDIF
END
PUBLIC SUB Button1_Click()
Global.t = textareal.Text
Form1.Close
Form2.show
END
```

Code für Form2

```
PUBLIC SUB Form_Open()
Textareal.Text = global.t
Textareal.Text = Textareal.Text & " und etwas dazu." & Chr(10)
END
PUBLIC SUB Button1_Click()
Global.t = textareal.Text
form2.close
form1.Show
END
```

Jedesmal wenn man von einer Form zur anderen wechselt, wird der Text der Textarea der globalen Variablen t zugeordnet und in der nächsten Form noch etwas hinzugefügt.



### 3.42 Kopieren des Formlayouts

Sie wollen ein umfangreiches Formlayout exakt von einer Form auf eine andere kopieren. Wenn man dazu alle Elemente der Form anmarkiert und sie mit der rechten Maustaste mittels Copy und Paste auf die andere Form übertragen will, dann stehen die Elemente (z.B. die Buttons) oft nicht mehr an der richtigen Stelle.

Mit einem simplen Trick sind auch nach dem Kopieren alle Elemente richtig an Ort und Stelle.

Man platziert behelfsweise zusätzlich ein Steuerelement exakt in die linke obere Ecke (Position 0,0), hier z.B. der Button3. Dieser dient als Sperre dagegen, dass Gambas die kopierten Elemente beim Einfügen verschiebt. Vor allem bei komplexeren Layouts mit vielen, teilweise auch überlappenden Elementen ersparen man sich so jede Repositionierungsarbeit. Dieser kleine Trick hilft auch in einer ganzen Reihe von anderen Grafik-Programmen.

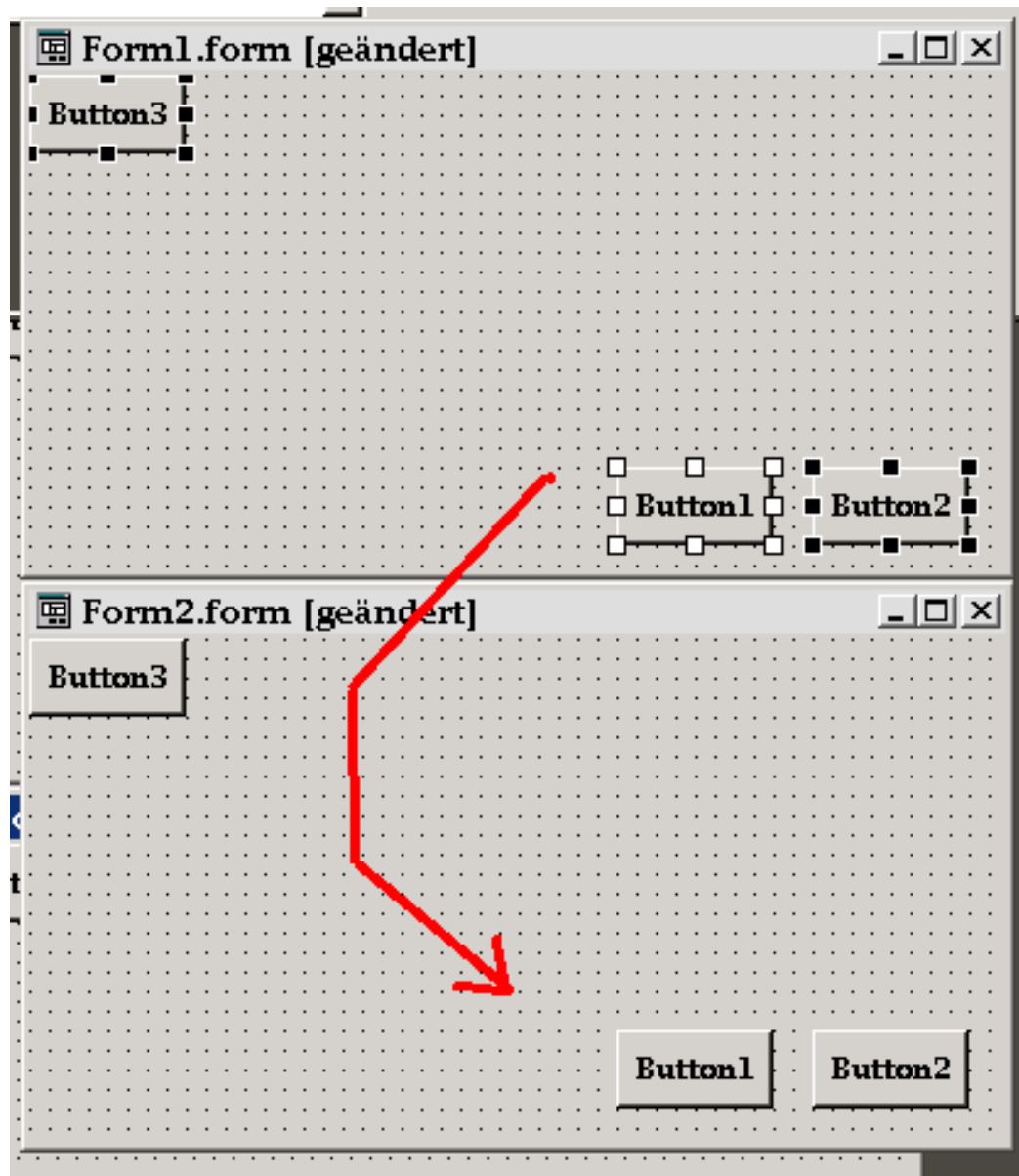


Abbildung 35: Bild:Prgformcopy.png

Siehe <http://www.madeasy.de/7/prgformcopy.htm>

## 3.43 Resize einer Textarea an eine Form

Sie haben eine Form mit einer Textarea. Die Größe dieser Textarea möchten Sie immer an Ihre Formgröße anpassen.

### 3.43.1 Das Programm

Um das Programm in Gang zu bringen, brauchen Sie eine Form und eine Textarea.

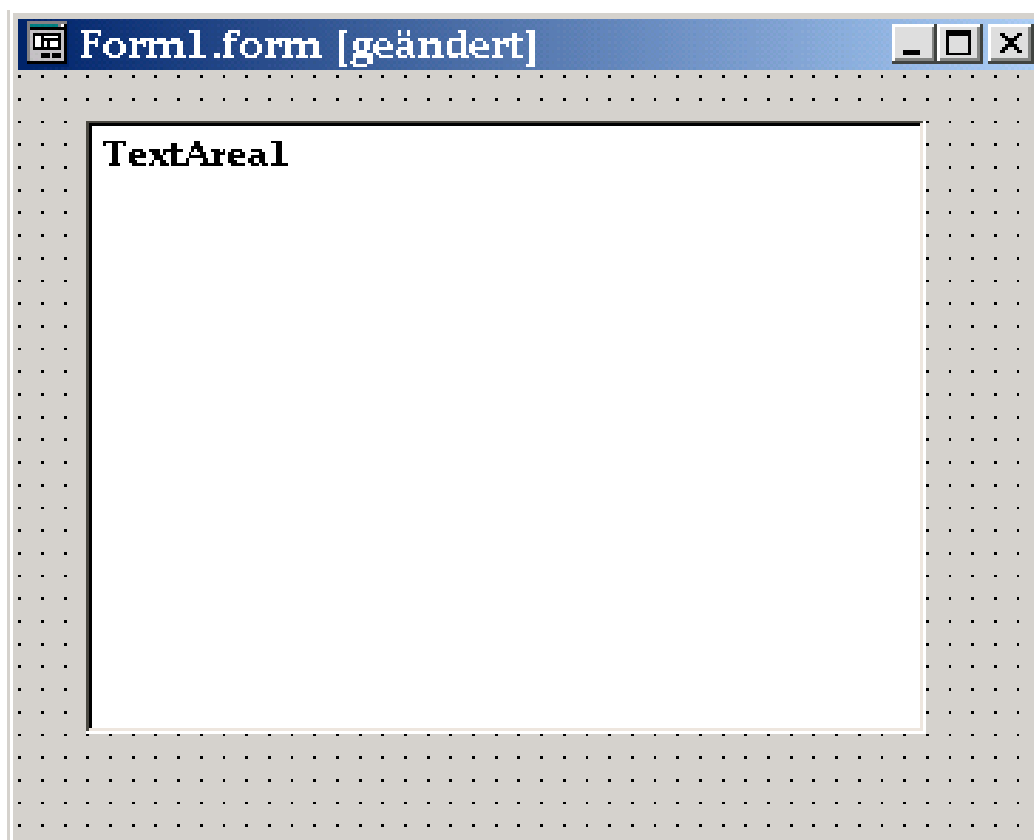


Abbildung 36: Bild:Prgresize1.png

Wenn Sie die Form verschieben zieht die Textarea immer mit.

Wenn Sie bei den Formeigenschaften die Bordereigenschaft auf `Resizable` stellen, wird die Textarea automatisch an eine Größenänderung der Form angepasst.

So schaut das Programm im Laufmodus aus:

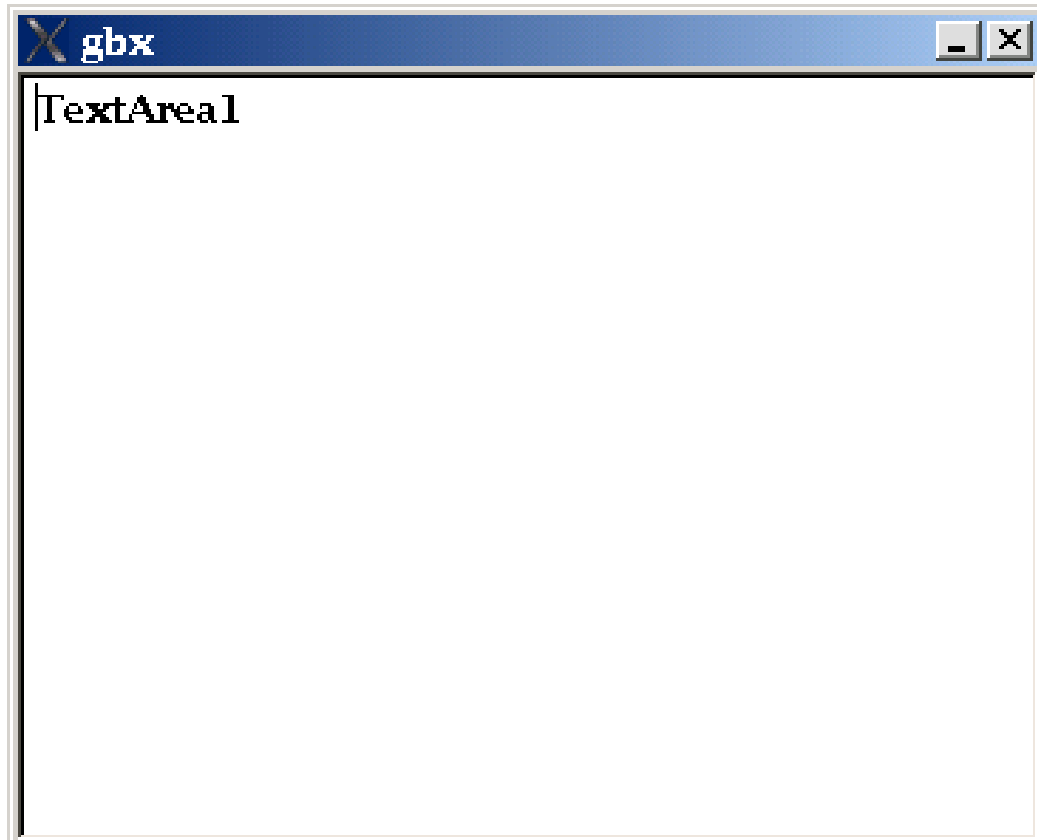


Abbildung 37: Bild:Prgresize2.png

### 3.43.2 Der Code:

```
PUBLIC SUB Form_Resize()  
TextAreal.Move(0, 0, ME.ClientWidth, ME.ClientHeight)  
END
```

Was hier für eine Textarea gezeigt wurde, geht natürlich genauso auch mit einer PictureBox oder einer Drawingarea.

Siehe <http://www.madeasy.de/7/prgresize.htm>

## 3.44 Ändern der Breite eines Textfeldes beim Ändern der Breite des Formulares

```
PUBLIC SUB Form_Resize()
    TextBox1.Move(TextBox1.X,TextBox1.Y,Form1.ClientW - ( TextBox1.X * 2 ) ,TextBox1.H)
END
```

Die Breite der Textbox wird bei Breitenänderung des Formulares so angepaßt, daß rechts derselbe Abstand zum Rand besteht wie links.

## 3.45 Allgemeines

Grid heißt auf deutsch Gitter. Noch besser kann man ein Grid auch als Tabelle bezeichnen. Tabellen eignen sich sehr gut, um Datenbanken übersichtlich darzustellen.

Verwandt mit GridView ist der [TableView](#), die Tabellenansicht, wie sie z.B. in dem mitgelieferten Beispielprogramm DataReportExample genutzt wird.

Bei einem Gitter kann man die Spaltenanzahl, die Reihenanzahl, die Breite und Höhe und vieles mehr definieren. Angesprochen werden die einzelnen Felder des Gitters mit einem Array, d.h. mit einem Wert für die Spalten- und die Reihenordinate.

Man kann die Gitterplätze mit Text, mit Zahlen, aber auch mit einem Bild füllen.

## 3.46 Beispielprogramm

Um das Programm in Gang zu bringen, braucht man eine neue Form. Auf diese platziert man ein GridView (ein Gitter) aus der Werkzeugkiste. Im Programmverzeichnis sollte die Bilddatei x.png vorhanden sein, sonst wird das Bild nicht dargestellt. Eine Fehlermeldung gibt es nicht.

```
STATIC PUBLIC SUB Main()
    hForm AS Fmain
    hForm = NEW Fmain
    hForm.show
END

PUBLIC SUB _new()
    GridView1.Columns.Count = 4
    GridView1.Rows.Count = 3
```

```
GridView1.Columns.Width = 52
GridView1.Rows[1].Height = 52
GridView1[0,0].Text = "0,0"
GridView1[0,0].Alignment = 4
GridView1[1,1].Text = "1,1"
GridView1[0,1].Text = "0,1"
GridView1[1,0].Picture = Picture["x.png"]
END
```

## 3.47 Theorie des GridView

Mit dem GridView wird ein Steuerelement bereitgestellt, welches Daten in einem Gitternetz anzeigt. Die Klasse kann man aus der Werkzeugkiste holen. Sie ist aber auch per Code erzeugbar.

```
g AS GridView
PUBLIC SUB _new()
  g = NEW GridView(ME) AS "GridView1"
  g.show
  g.Columns.Count = 4
  g.Rows.Count = 3
  g.Columns.Width = 52
  g.Rows[1].Height = 52
END
```

Sie brauchen eine leere Form um obiges Programm in Gang zu bringen. Es erzeugt ein neues GridView.

### 3.47.1 Eigenschaften des GridView

BackColor, Background, Border, ClientH, ClientHeight, ClientW, ClientWidth, Column, Columns, Current, Cursor, Design, Drop, Enabled, Expand, Font, ForeColor, Foreground, Grid, H, Handle, Height, Id, Left, Mouse, Parent, Row, Rows, ScreenX, ScreenY, ScrollBar, Tag, ToolTip, Top, Visible, W, Width, Window, X, Y

### 3.47.2 Methoden

Clear, Delete, Drag, Grab, Hide, Lower, Move, Raise, Refresh, Resize, SetFocus, Show

### 3.47.3 Ereignisse

Activate, Click, DbClick, Drag, DragMove, Drop, Enter, GotFocus, KeyPress, KeyRelease, Leave, LostFocus, Menu, MouseDown, MouseMove, MouseUp, MouseWheel, Scroll

## 3.48 GridView mit Werten füllen

Sie haben eine Liste von Werten erstellt, diese wollen Sie in das Gitter übernehmen. Wie das geht zeigt das folgende Beispielprogramm:

Sie brauchen dazu eine neue Form mit folgenden Elementen:

- 1 TextArea
- 1 GridView
- 2 Commandbuttons

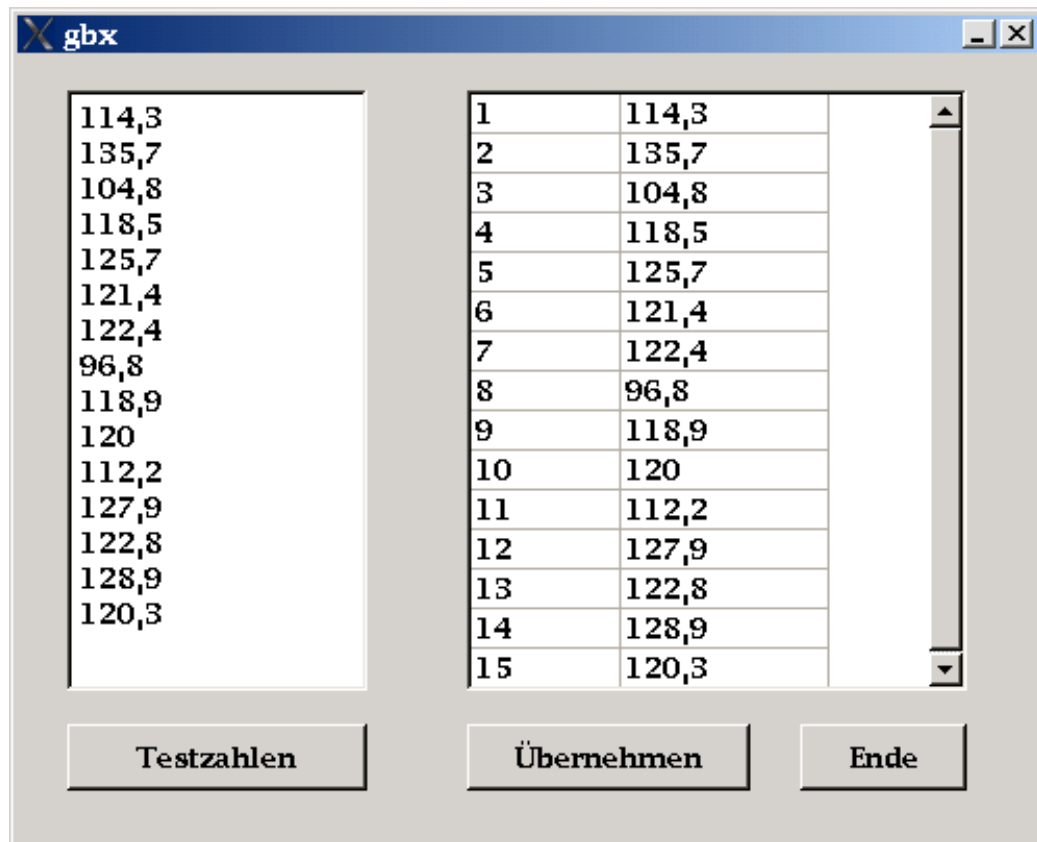


Abbildung 38: Gitter mit Werten füllen

Der Code dazu:

```

PUBLIC SUB Button1_Click()
    TextAreal.Text = "114,3\n135,7\n104,8\n118,5\n125,7\n"
    TextAreal.Text = TextAreal.Text & "121,4\n122,4\n96,8\n118,9\n120\n"
    TextAreal.Text = TextAreal.Text & "112,2\n127,9\n122,8\n"128,9\n120,3"
END

PUBLIC SUB _new()
    GridView1.Columns.Count = 2
    GridView1.Rows.Count = 15
    GridView1.Columns.Width = 72
END

PUBLIC SUB Button2_Click()
    DIM text AS String
    DIM Liste AS String[]
    DIM Einzelwert AS String
    DIM x AS Integer

    x = 0
    text = TextAreal.Text

```



```

Liste = Split(text,Chr(10))

FOR EACH Einzelwert IN Liste
  GridView1[x,1].Text = Einzelwert
  GridView1[x,0].Text = x
  x = x + 1
NEXT

PRINT liste.Length
END

```

Wenn man auf den Button1 drückt wird die TextArea mit Werten gefüllt. Drückt man auf den Button2, dann werden diese Werte in das Gitter übernommen.

Wie kann man das Programm noch verbessern?

1. Die Länge des Gitters ist mit einem Wert von 15 fest vorgegeben. Kann man die Länge so ändern, dass sie variabel wird und der Länge der Liste (Liste.Length) in der TextArea entspricht?

### 3.49 Hintergrundfarbe für eine beliebige Zelle setzen

Manchmal möchte man, vielleicht um den Inhalt bestimmter Zellen hervor zu heben, die Hintergrundfarbe einer oder mehrerer Zellen auf eine beliebige Farbe ändern. Eigentlich geht das nicht, denn dieses ist nicht in den Eigenschaften des GridView vorgesehen. Aber es gibt einen kleinen „Workaround“, den ich im folgenden beschreibe.

1. man braucht natürlich ein GridView. Nennen wir es z.B. **GridViewTest**.
2. nun definiert sich eine Variable (z.B. **bild**) als Picture.
3. dann erstellt sich eine TextBox (z.B. **TextBoxTemp**) in der Größe einer Zelle des GridView.
4. man setzt die Eigenschaft **TextBoxTemp.Visible = False** (TextBoxTemp ist unsichtbar).
5. im nächsten Schritt legt die Hintergrundfarbe der TextBox fest.
6. man kopiert den Inhalt der Zelle in die TextBox, dessen Hintergrundfarbe geändert werden soll.
7. dann setzt die Eigenschaft von **TextBoxTemp.Visible** auf True (sichtbar machen).

8. jetzt macht nun mit **bild = TextBoxTemp.Grab()** ein Bild,
9. setzt **TextBoxTemp.Visible** auf False und
10. kopiert das Bild in die GridView Zelle.

Sollte es nicht funktionieren, muss noch ein **WAIT** an die richtige Stelle gesetzt werden. Die wesentlichen Teile hier noch einmal als Programm-Code.

```
DIM bild AS Picture
bild = NEW Picture( <Breite der TextBox>, <Höhe der TextBox>, TRUE )
TextBoxTemp.Visible = TRUE
TextBoxTemp.Text = GridViewTest[ <Reihe>, <Spalte> ].Text
WAIT
bild = TextBoxTemp.Grab()
TextBoxTemp.Visible = FALSE
GridViewTest[ <Reihe>, <Spalte> ].Picture = bild
```

Wichtig ist, dass **WAIT** genau so, und vor der *Grab-Zeile* eingefügt wird. **WAIT bild = TextBoxTemp.Grab()** funktioniert NICHT.

Das ist vielleicht nicht ganz „sauber“ programmiert, aber eine andere Möglichkeit zur Lösung des Problems habe ich bislang noch nicht gefunden.

## 3.50 TableView-Komponente dazuladen

Die Tabelle muss man sich aus der QT-Werkzeugkiste holen, da sie nicht in der normalen Gambas-Werkzeugkiste zu finden ist.

Dazu lädt man die **gb.qt.ext**-Komponente. Dies kann man über den Menüpunkt Projekt -> Eigenschaften des Gambas-Projektfensters erreichen. In der Komponentenanzeige wählt man die Komponente **gb.qt.ext** zusätzlich aus.

Siehe auch: [Gambas: Komponenten](#).

In der Werkzeugbox erscheint dann eine Schaltfläche mit der Bezeichnung QT. Diese klickt man an und hat dann eine Reihe weiterer Steuerelemente darunter die Tabellenansicht (TableView) zur Verfügung.

## 3.51 TableView nutzen

Das folgende Programm nutzt das TableView-Steuerelement aus **gb.qt.ext**. In eine Tabelle mit 10 Spalten und 10 Zeilen werden die Zahlen des 1\*1 eingefügt. Sie

brauchen eine Form und das TableView-Steuerelement aus der Werkzeugkiste, um es in Gang zu bringen.

```
PRIVATE aro AS NEW Object[]
PUBLIC SUB _New()
    DIM ari AS Variant[]
    DIM i AS Integer
    DIM j AS Integer

    'es wird ein Array 10 X 10 erzeugt
    FOR i = 0 TO 9
        ari = NEW Variant[]
        ari.Resize(10)
        aro.Add(ari)
    NEXT

    'die Werte 1 bis 100 werden den Array-Elementen zugeordnet
    FOR i = 0 TO aro.Count - 1
        FOR j = 0 TO aro[i].Count - 1
            aro[i][j] = (i + 1) * (j + 1)
        NEXT
    NEXT

    TableView1.Rows.Count = 10
    TableView1.Columns.Count = 10
END

PUBLIC SUB TableView1_Data(Row AS Integer, Column AS Integer)
    TableView1.Data.Text = aro[Row][Column]
    IF aro[Row][Column] = 50 THEN TableView1.Data.BackColor = Color.Gray
END
```

## 3.52 Zellen bearbeiten

Das TableView-Steuerelement ist aber nur für die Darstellung der Werte zuständig, man kann die Werte in den Zellen nicht editieren. Will man die Werte direkt in den Zellen bearbeiten, wie man es z.B. von Tabellenkalkulationen kennt, muss man dynamisch eine TextBox über die angeklickte Zelle positionieren, in der dann der Zelleninhalt bearbeitet werden kann. Aber keine Sorge, dass klingt komplizierter als es ist.

Beginnen Sie ein neues Projekt, legen Sie eine neue Form an und platzieren Sie auf dieser Form ein TableView, eine TextBox und ein Button. Danach benötigen Sie noch den folgenden Quellcode: [TableView1 Form1.class](#)

### 3.53 Farbige Zellen

Mit nur geringem Aufwand lässt sich das obige Beispiel so erweitern, das man den einzelnen Zellen verschiedene Hintergrundfarben zuweisen kann. Dabei kann man auf die Zellen des TableViews nicht direkt zugreifen, sondern muss die Farbwerte in einem entsprechenden Array (im folgenden Beispiel *\$hColor*) vorrätig halten und innerhalb der *TableView1\_Data()* Routine dem TableView zur Verfügung stellen. Dazu ändern wir den Quellcode des vorherigen Beispiels wie folgt: [TableView2 Form1.class](#)

### 3.54 Navigieren mit den Cursortasten

Was uns jetzt noch fehlt, ist die Möglichkeit, die jeweils zur Bearbeitung ausgewählte Zelle mit den Cursortasten zu wechseln. Um dies zu ermöglichen, müssen wir nur noch zwei Dinge tun. Erstens werden wir die *TextBox1\_KeyPress()* Funktion so ändern, dass sie auf die entsprechenden Tasten reagiert. Zum Zweiten rufen wir am Anfang der *TextBox1\_Show()* Funktion die *TableView1.MoveTo()* Methode auf, um sicherzustellen, dass die Zelle, die wir bearbeiten wollen, auch im sichtbaren Bereich des TableViews liegt. Der entsprechende Quellcode: [TableView3 Form1.class](#)

Ferner verzichten wir jetzt darauf, die Hintergrundfarbe der TextBox auf Gelb zu setzen und verwenden stattdessen die jeweilige Hintergrundfarbe aus dem Array *\$hColor*.

### 3.55 Wiki-Tabelle

Nach all den Trockenübungen wollen wir uns jetzt einem zumindest halbwegs sinnvollen Beispiel für den Einsatz eines TableViews widmen.

Jeder ambitionierte Wikibook- oder Wikipedia-Autor wird früher oder später in die Verlegenheit geraten, bestimmte Informationen in einer Tabelle darstellen zu wollen. Um dabei etwa die folgende Tabelle zu erzeugen:

Wert1	Wert2	Wert3
Wert4	Wert5	Wert6
Wert7	Wert8	Wert9

benötigt man den folgenden Wiki-Quellcode:

```
{|{{prettytable}}
| Wert1 || Wert2 || Wert3
|-
| Wert4 || Wert5 || Wert6
|-
| Wert7 || Wert8 || Wert9
|}
```

Das wird bei größeren Tabellen sehr schnell unübersichtlich.

Wir brauchen also ein Programm, bei dem wir die Daten bequem in eine Tabelle eingeben können und das uns auf Knopfdruck den notwendigen Wiki-Quellcode liefert.

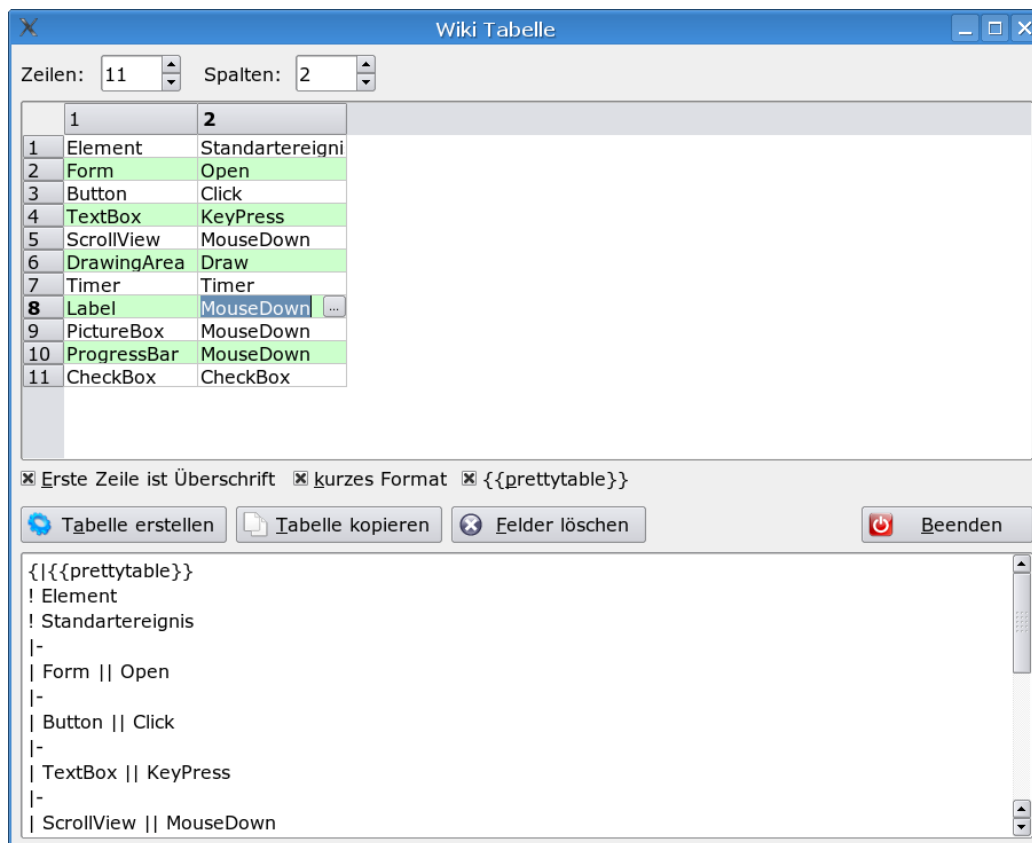


Abbildung 39: Wiki-Tabelle

Quellcode:

- [Form1.class](#)

- [Form1.form](#)
- [Form2.class](#)
- [Form2.form](#)

Erzeugen Sie ein neues Projekt mit Namen **Wiki-Tabelle**, beenden Sie Gambas direkt wieder und kopieren Sie den Quelltext von den vier vorstehenden Seiten in (neue) Dateien mit den entsprechenden Namen (Form1.class, ...) in das gewählte Projektverzeichnis. Anschließend starten Sie Gambas wieder und öffnen das Projekt und definieren im Projektfenster die *Form1* als *Startklasse*.

In der Werkzeugkiste von Gambas gibt es eine Element, welches eine Listendarstellung ermöglicht.

Wie dieses Element funktioniert soll folgendes Beispiel zeigen:

Sie brauchen einen Commandbutton und ein Listviewelement. Holen Sie sich beides mit **F6** aus der Werkzeugkiste.

```
PUBLIC SUB Button1_Click()  
  Listview1.add("1", "Name1")  
  Listview1.add("2", "Name2")  
  Listview1.Add("3", "Name3")  
END
```

Wenn Sie ein zweites Mal auf den Befehlsknopf klicken, gibt es eine Fehlermeldung. Warum?

Im zweiten Beispiel werden die Dateieinträge des aktuellen Unterverzeichnisses in die Liste eingetragen.

Sie brauchen einen Commandbutton und ein Listviewelement. Holen Sie sich beides mit **F6** aus der Werkzeugkiste.

```
PUBLIC SUB Button1_Click()  
  sDir AS String  
  sPath AS String  
  FOR EACH sDir IN Dir(sPath, "**")  
    ListView1.Add(sPath &/ sDir, sDir)  
  NEXT  
END
```

## 3.56 Einfache Meldung

Eine Meldung ist sehr einfach zu programmieren. Geben Sie folgenden Code in eine leere Form ein und starten Sie das Programm.

```
PUBLIC SUB Form_Open()  
    Message("Dies ist eine Meldung", "ButtonText")  
END
```

Alternativ können Sie auch folgenden Code verwenden:

```
PUBLIC SUB Form_Open()  
    Message.Info("Hallo, das ist die Info Meldung", "OK")  
END
```

## 3.57 Meldung mit mehreren Antwortmöglichkeiten

Neben der einfachen Message.Info gibt es vier weitere Meldungsarten:

- Message.Delete (Löschen)
- Message.Error (Fehlermeldung)
- Message.Question (Frage)
- Message.Warning (Warnmeldung)

Bei diesen Meldungsboxen sind bis zu drei Antworten möglich. Außerdem kommt ein Icon zur Anzeige:

- Delete = "Mülltonne",
- Error = "Roter Punkt mit X",
- Question = "?" Fragezeichen,
- Warning = "Gelbes Dreieck mit Ausrufungszeichen"

### 3.57.1 Beispiel

Wie diese Meldungsboxen funktionieren zeigt folgendes Beispiel. Sie brauchen einen Button und ein Labelsteuerelement auf Ihrer Form um das Programm in gang zu bringen. Beides finden Sie in der Werkzeugkiste mit F6.

### 3.57.2 Layout

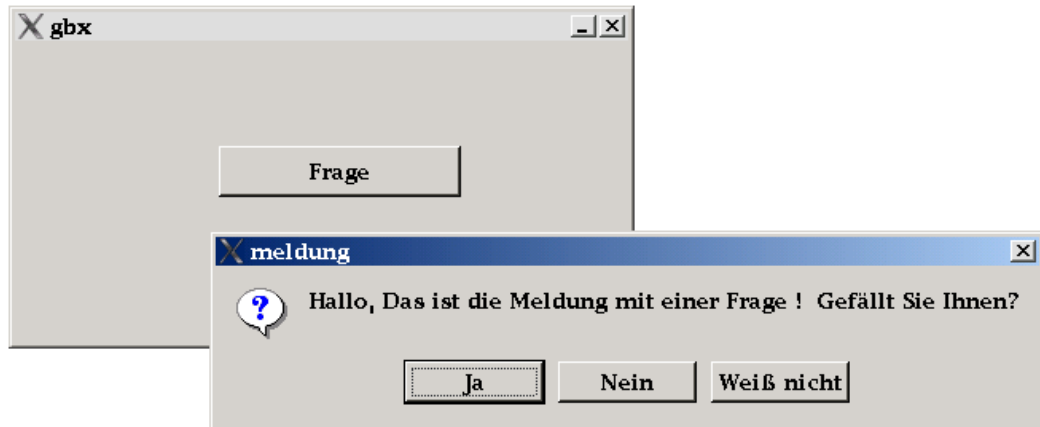


Abbildung 40: Bild:Gambasmeldung.png

### 3.57.3 Code

```
PUBLIC SUB Button1_Click()  
    SELECT Message.Question("Hallo, das ist die Meldung mit einer Frage! Gefällt Sie Ihnen?", "Ja",  
"Nein", "Weiß nicht")  
    CASE 1  
        Label1.Text = "Erfreulich!"  
    CASE 2  
        Label1.Text = "Bedauerlich!"  
    CASE 3  
        Label1.Text = "Sie werden sie bald mögen!"  
    END SELECT  
END
```



# Kapitel 4

## Beispielprogramme

### 4.1 Einleitung

Wenn man ein bisschen mathematisch interessiert ist, kann man eine Programmiersprache wie Gambas sehr gut nutzen, um die Mathematik noch besser zu verstehen.

Umgekehrt kann man durch die Programmierung kleiner mathematischer Programme sehr viel über eine Programmiersprache lernen. Der Vorteil mathematischer Programme ist, dass sie meistens gut überschaubar sind und die Fehlerbeseitigung in kurzer Zeit vollzogen ist.

Außerdem kann man anfangs auch völlig auf eine Grafik verzichten und nur mit dem Direktfenster arbeiten, auch wenn die Grafik ja gerade den Reiz von KDE Programmen ausmacht.

### 4.2 Rechenausdrücke

Um längere Rechenausdrücke in Gambas korrekt zu berechnen, sollte man einige Vereinbarungen für mathematische Zeichen kennen, die in fast allen Basicdialekten gültig sind. Für die Multiplikation, die Division und die Potenzierung werden dabei andere Zeichen gebraucht, als die im Deutschen üblichen.

- Die Addition (Zusammenzählen) erfolgt mit +
  - z.B. `Print 8 + 8`

Die Subtraktion (Abziehen) erfolgt mit -

– z.B. Print 9 - 5

Die Multiplikation (Malnehmen) erfolgt mit dem Sternchen \*

– z.B. Print 5 \* 5

Die Division (Teilen) erfolgt mit dem Schrägstrich /

– z.B. Print 10 / 5

– ergibt 2

– Vorsicht bei Teilung mit oder ohne Rest

Die Potenzierung (Hochzahlen) berechnet man mit dem ^ Zeichen.

– z.B. Print 8 ^ 2

– ergibt 64

Die Klammern legen den Vorrang beim Rechnen fest 5 \* (5 + 5)

– Falsch ist 5(5+5), Richtig ist 5 \* (5 + 5)

Für die Berechnung gelten folgende Regeln der **Vorrangigkeit**:

- Zuerst werden die Klammern berechnet. Die innersten Klammern zuerst.
- Bei den Operationen gilt die Rangfolge: Erst Potenzieren, dann multiplizieren oder dividieren, dann addieren oder subtrahieren.
- Gleichrangige Ausdrücke werden von links nach rechts abgearbeitet.

Es gibt in Gambas nur ein Rechenzeichen für die Quadratwurzel **SQR**. Die höheren **Wurzeln** muss man anders berechnen: Hier gilt x-te Wurzel aus a ist gleich  $a^{(1/x)}$  (a hoch 1 durch x) Beispielsweise wird die 3-te Wurzel aus 27 so berechnet:  $27^{(1/3)} = 3$

Statt

$$\frac{36}{2 \cdot 3 \cdot 6}$$

schreibt man in Gambas: 36/2/3/6 oder  $36/(2*3*6) = 1$

Beachten Sie, dass das **Minuszeichen** einmal als Vorzeichen und einmal als Subtraktionszeichen gebraucht wird. Folgende Rechnungen sind gültig:

$$5 + -2 = 3$$

$$-5 * 3 = -15$$

$$5*-3 = -15$$

$$5/4 = 1,25 \text{ oder } 1.25 \text{ je nach Ländereinstellung}$$

$$20/2*5 = 50$$

---

$20/2/5 = 2$   
 $20/(2*5) = 2$   
 $5*(2+3) = 25$   
 $2*(2+3)/(4+6) = 1$   
 $((2+3)*2)^2 = 100$

## 4.2.1 Addy der Summierer

Wenn man eine Kolonne von Zahlen zusammenzählen will, verliert man schnell den Überblick, welche Zahl man schon eingegeben hat und welche nicht. Man kann dazu natürlich eine große Tabellenkalkulation anwerfen. Diese ist aber etwas kompliziert zu bedienen.

Einfacher soll diese Aufgabe Addy, ein kleines Summierprogramm erledigen. Auch eine nachträgliche Korrektur der Eingabe soll möglich sein.

Die Eingabe der Zahlen soll schnell erfolgen, am besten durch ein Return ohne Mausaktion.

Bei dem Programm können auch negative Zahlen summiert werden. Sie werden dann korrekt abgezogen. Auch das Zusammenzählen von Kommazahlen ist möglich.

Außerdem kann man Zahlenkolonnen via Zwischenspeicher übertragen.

Sehr hilfreich bei der Erstellung dieses Programms ist der Splitbefehl. CHR(10) als Trennzeichen ist in diesem Fall der Zeilenwechsel.

Was brauchen Sie? Holen Sie sich eine neue Form und nennen Sie diese wie Sie wollen. Holen Sie sich eine TextArea, eine TextBox und einen Commandbutton aus der Werkzeugkiste. Löschen Sie den Text aus den Textboxen bei den Eigenschaften (= Properties) Text.

So schaut das Ganze aus:

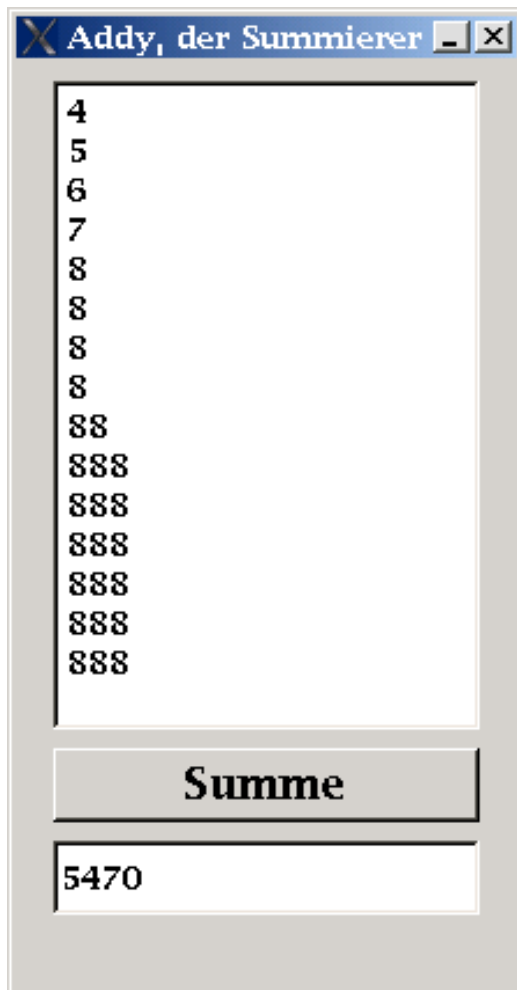


Abbildung 41

Der Programmcode dazu. Er steht hinter dem Befehlsbutton Summe:

```
PUBLIC SUB Button1_Click()  
    DIM text AS String  
    DIM summe AS Float  
    DIM Elt AS String()  
    DIM Sb AS String  
  
    text = TextAreal.Text  
    Elt = Split(text, Chr(10))  
    FOR EACH Sb IN Elt  
        summe = summe + Val(sb)  
    NEXT  
    TextBox1.Text = summe  
END
```

Das Programm ist doch ganz praktisch, oder was meinen Sie? Bei Kommazahlen erfolgt die Ergebnisausgabe noch inkorrekt mit dem englischen Punkt als Komma. Versuchen Sie diesen Fehler zu beheben.

In dem Programm bekommt man eine Fehlermeldung *Type mismatch: Wanted Number, got Null instead* wenn Sie z.B. versuchen Buchstaben statt Zahlen einzugeben.

Der Computer kann hier nur etwas mit Zahlen anfangen, nicht mit Buchstaben. Sie können den Fehler vermeiden, wenn Sie die Eingabe auf Ziffern, Kommas und Minuszeichen begrenzen. Auch ein Abfangen des Fehlers mit Catch ist sicher möglich.

Wollen Sie die Eingaben in Addy speichern. Dann schauen Sie sich doch folgende Erweiterung an: [Gambas: Dateien#Datei laden und speichern mit Addyplus](#)

Das Programm Addy regt an, um über einen kleinen Mittelwert- und Standardabweichungsrechner nachzudenken.

## 4.2.2 Potenzen

Folgendes Beispielprogramm zeigt den Umgang mit Potenzen. Es zeigt auch die dabei möglichen Fehler auf. Es arbeitet im Terminalmodus.

```

STATIC PUBLIC SUB Main()
  PRINT 4^3
  PRINT 3^4
  PRINT 4^1/2
  PRINT 4^(1/2)
  PRINT 8^0.333
  PRINT 8^(1/3)
  PRINT 8^1/3
  PRINT 2^(-2)
  PRINT 1/2^2
END

```

Die Ausgabe schaut so aus:

```

64
81
2
2
1,998614185981
2
2,6666666666667
0,25
0,25

```

Frage dazu: Wozu wird hier der Befehl *Static* verwendet? Geht es auch ohne ihn?

Der Befehl *Static* wird im Terminalmodus automatisch von der GUI vergeben. Löscht man ihn weg, funktioniert das Programm nicht mehr und es erscheint eine Fehlermeldung: *ERROR: #24: No startup method*

### 4.2.3 Modulo oder Modulus (Rest beim Teilen)

Der MOD Befehl entspricht – einfach erklärt – dem Rest beim Teilen. Geht die Division auf, bleibt kein Rest (also 0).

Ein paar Beispiele sind (besonders für Programmieranfänger) sinnvoll für das Verständnis:

- 10 MOD 5 ergibt 0 (Oft gemachter Fehler: Es kommt nicht 2 heraus. MOD ist nicht das selbe wie /)
- 2 MOD 3 ergibt 2
- 6 MOD 4 ergibt auch 2
- 8 MOD 3 ergibt 2

Ein weiteres Beispiel zeigt folgendes Programm im [Terminalbetrieb](#):

```
STATIC PUBLIC SUB Main()  
  DIM a AS Integer  
  DIM b AS Integer  
  
  b = 2  
  FOR a = 10 TO 2 STEP - 1  
    PRINT a MOD b  
  NEXT  
END
```

Setzen Sie verschiedene Werte für b ein und schauen Sie sich die Ausgabe an. Falls Sie b = 0 setzen, dann erfolgt eine Fehlermeldung: *Division by zero*.

### Lernprogramm Modus

Siehe <http://www.madeasy.de/7/prgmodus.htm>

### Quersumme berechnen

Im folgenden Programm wird der Modus Befehl zur Berechnung der Quersumme einer mehrstelligen Zahl benutzt: Holen Sie sich eine neue Form und nennen Sie diese wie Sie wollen. Holen Sie zwei Textboxen aus der Werkzeugkiste.

Löschen Sie den Text aus den Textboxen bei den Eigenschaften (= Properties) Text.

Holen Sie sich einen Commandbutton dazu und benennen Sie diesen um.

Siehe <http://www.madeasy.de/7/prgquer.htm>

Hinter dem Befehlsfeld Quersumme steht folgender Code: Beachten Sie das hier das Divisionszeichen / nicht verwendet wird, sondern das Teilenzeichen ohne Rest \.

```
PUBLIC SUB Button1_Click()
    DIM zahl AS Integer
    DIM Quersumme AS Integer

    zahl = Val(TextBox1.Text)
    DO WHILE zahl <> 0
        Quersumme = Quersumme + zahl MOD 10
        zahl = zahl \ 10
    LOOP
    TextBox2.Text = Str$(Quersumme)
END
```

## Gerade oder Ungerade

Den Modulus Befehl kann man nutzen, um zu überprüfen ob eine ganze Zahl gerade (teilbar durch 2) oder ungerade (nicht teilbar durch 2) ist.

Programm: Sie brauchen 2 Befehlsbuttons und 2 Textboxen auf der Form, um das Programm in Gang zu bekommen.

Benutzen Sie folgenden Code um die Eigenschaften der Steuerelemente automatisch einzurichten:

```
PUBLIC SUB Form_Open()
    Button1.Text = "Zufallszahl erzeugen"
    Button1.Move(16, 8, 176, 32)
    Button2.Text = "Gerade-Ungerade Testen"
    Button2.Move(16, 48, 176, 32)
    TextBox1.Text = ""
    TextBox1.Move(192, 8, 88, 32)
    TextBox2.Text = ""
    TextBox2.Move(192, 48, 88, 32)
END
```

Sie können die Eigenschaften natürlich auch mit der Hand über das Eigenschaftsfenster festlegen. Der eigentliche Code beginnt hier:

```
PUBLIC SUB Button1_Click()
    Randomize()
    textbox1.Text = Str(Int(Rnd(1,101)))
END
```

```
PUBLIC SUB Button2_Click()  
    DIM n AS Integer  
  
    n = Val(textbox1.text)  
  
    IF textbox1.Text = "" THEN textbox1.Text = Str(Int(Rnd(1,101)))  
  
    IF n MOD 2 THEN  
        textbox2.Text = "Ungerade"  
    ELSE  
        textbox2.Text = "Gerade"  
    ENDIF  
END
```

Mit dem ersten Befehl wird eine ganze Zufallszahl zwischen 1 und 100 erzeugt. Mit dem zweiten Befehl wird die Teilbarkeit durch 2 überprüft.

Falls  $n$  eine ungerade Zahl ist, dann ist  $n \text{ MOD } 2$  gleich 1, denn es bleibt ein Rest von 1 übrig. Der Wahrheitswert ist true. Wenn kein Rest übrig bleibt ist der Wahrheitswert false.

Ausführlicher kann man auch schreiben:

```
IF n MOD 2 = true Then
```

Versuchen Sie es zu verstehen und probieren sie das Programm aus.

## 4.3 Übersicht der mathematischen Zeichen in Gambas

siehe [Mathematische Zeichen in Gambas](#)

## 4.4 Zahlendarstellung

### 4.4.1 Einfache Rechnungen mit ganzen Zahlen

Zahlen werden in Gambas zunächst ganz normal wie gewohnt gehandhabt. So ergibt der Ausdruck **PRINT 8 + 8** in der Ausgabe 16.

```
PUBLIC SUB Form_Open()  
    PRINT 8 + 8  
END
```



Will man mit Zahlen im Programm länger arbeiten, muss man sie als Variable definieren. Die einfachste Form dafür ist die Variable Integer. Die Deklaration einer Zahlenvariable erfolgt dann z.B. mit **Dim zahl AS Integer**.

Im folgenden einfachen Beispielprogramm für die Addition aller Ziffern von 1 bis 10 werden 2 Zahlenvariablen i und k definiert. Sie durchlaufen beide eine Schleife von 1 bis 10 und werden addiert. Kopieren Sie das Programm in Gambas und lassen Sie es laufen. Ersetzen Sie das Pluszeichen durch andere mathematische Zeichen und prüfen Sie was passiert.

```
PUBLIC SUB Form_Open()  
  DIM i AS Integer  
  DIM k AS Integer  
  
  FOR i = 1 TO 10  
    FOR k = 1 TO 10  
      PRINT i + k & " ";  
    NEXT  
    PRINT  
  NEXT  
END
```

Durch die erste Zeile mit dem Befehl Form\_Open() startet das Programm sofort beim Aufrufen der Form. Mit den Befehlen DIM werden die Variablen i und k als ganze Zahlen (= Integer) festgelegt. Dann folgen 2 FOR - TO Schleifen, die verschachtelt sind. In der ersten Schleife wird die Variable i von 1 bis 10 hochgezählt. Bevor aber i zu einem höheren Wert springt, muss die innere Schleife für k immer wieder von 1 bis 10 abgearbeitet werden. In der Ausgabe mit dem Printbefehl werden dann i und k mathematisch verknüpft und im Direktfenster mit PRINT ausgegeben. Der Teil & " " fügt immer ein Leerzeichen zwischen die errechneten Zahlen i + k ein. Der Strichpunkt am Ende der Zeile verhindert einen Zeilenwechsel. So wird die Ausgabe in eine Zeile geschrieben.

Erst in der äußeren Schleife für i steht ein einsamer Printbefehl. Dieser bewirkt einen Zeilenvorschub jedesmal wenn ein neuer Wert für i angesagt ist. Der Zeilenwechsel erfolgt also immer nach 10 Zahlen in der Querausgabe. Beachten Sie bitte, dass die Schleifen mit dem Nextbefehl geschlossen werden. Beim Nextbefehl wird in Gambas auf die Angabe Next i oder Next k verzichtet. Der Computer weiß das selbst.

Wenn Sie die 2 Schleifen nicht verstehen, dann ersetzen sie einfach das Pluszeichen durch ein Komma oder einen Strichpunkt. Daraufhin starten Sie das Programm. Jetzt können Sie erkennen wie die Werte von i und k hochgezählt werden. Wenn Sie es immer noch verstehen, empfehle ich Ihnen den Einzelschrittmodus im Programmablauf.

Versuchen Sie das Programm zu vereinfachen und halten einen der Werte z.B.  $i$  konstant. Fügen Sie dazu z.B. eine Zeile  $i = 5$  nach den DIM Befehlen ein und streichen Sie die äußere Schleife für den  $i$  Wert inklusive des Next Befehles.

Ihr Programm sollte dann so ausschauen:

```
PUBLIC SUB Form_Open()  
  DIM i AS Integer  
  DIM k AS Integer  
  
  i = 5  
  FOR k = 1 TO 10  
    PRINT i + k & " ";  
  NEXT  
  PRINT  
END
```

Die Variable  $i$  ist hier eigentlich überflüssig und kann auch noch herausgeworfen werden. Resultat:

```
PUBLIC SUB Form_Open()  
  DIM k AS Integer  
  
  FOR k = 1 TO 10  
    PRINT 5 + k & " ";  
  NEXT  
END
```

Auch die Multiplikation ist einfach und wie gewohnt. Sie müssen dazu nur im obigen Additionsprogramm das Pluszeichen mit dem englischen Multiplikationszeichen, dem Sternchen "\*" ersetzen.

Die Multiplikation zeigt z.B. ein einfaches Einmaleinsprogramm: Siehe auch <http://www.madeasy.de/7/prgschleife2.htm>

```
PUBLIC SUB Form_Open()  
  DIM i AS Integer  
  DIM k AS Integer  
  
  FOR i = 1 TO 10  
    FOR k = 1 TO 10  
      PRINT i * k & " ";  
    NEXT  
    PRINT  
  NEXT  
END
```

## 4.4.2 Exponential Darstellung mit E

Wenn Sie folgendes Programm laufen lassen, dann erscheint in der Ausgabe nicht wie erwartet 10 100 1000 etc sondern ab der Zahl eine Million werden die großen

Zahlen mit 1.E+8 und 1.E+10 abgekürzt. **Der Buchstabe E heißt in diesem Zusammenhang 10 hoch.** Die danach folgende Zahl mit einem Plus oder Minuszeichen davor ist die Zahl mit der die Zehn hochgenommen wird, also die Potenz.

1.E+8 heißt also genauer  $1,0 * 10^8$  oder  $1,0 * 10$  hoch 8 oder  $1,0 * 10^8$

```
PUBLIC SUB Form_Open()
  DIM i AS Integer
  DIM k AS Integer

  i = 10
  FOR k = 1 TO 10
    PRINT k ^ i & " ";
  NEXT
END
```

Noch einmal zur Wiederholung: Das große "E" bedeutet "Mal 10 hoch". Die Ganzzahl, die folgt gibt die Potenz von Zehn an mit der die Zahl zu multiplizieren ist. Eine andere Möglichkeit die Ganzzahl, die dem "E" folgt, zu betrachten ist, dass sie die Richtung und die Anzahl der Stellen angibt, in der der Dezimalpunkt zu verschieben ist. Positive Ganzzahlen bedeuten eine Rechtsverschiebung; negative Ganzzahlen bedeuten eine Linksverschiebung.

### 4.4.3 Rechenfehler

Übrigens ergibt das folgende Programm einen Rechenfehler.

```
PUBLIC SUB Form_Open()
  PRINT 1000000 * 1000000
END
```

**Rechnen Sie alle Ihre Gambas Programme mehrfach nach, da öfter unerwartete Ergebnisse durch Variablenumwandlung und durch die im Englischen und Deutschen verschiedene Verwendung des Punktes und des Kommas in Dezimalbrüchen auftreten.**

Beispiel: Deutsch  $1/2 = 0,5$  Englisch  $1/2 = 0.5$

Dieses Problem tritt aber in den meisten anderen Programmiersprachen auch auf und ist nicht Gambas spezifisch.

Die deutschen Mathematiker sollten sich bald möglichst an die englische Notation anpassen und auch gleich die Zeichen für die Multiplikation, Division und Potenzierung an die Computernotierung ändern.

Eine Länderliste mit ihren Dezimaltrennzeichen findet sich hier: <http://www.f-i-t.net/dezimalTrennz/>

## 4.5 Hochzählen, Runterzählen, Abschneiden, Runden etc INC, DEC, INT, FRAC

### 4.5.1 INC Hochzählen, 1 dazuzählen

Der Befehl INC zählt zu jeder numerischen Variable eins dazu. INC steht für Increment, d.h. heraufzählen.

Beispiel im Terminalmodus (d.h ohne Grafik)

```
STATIC PUBLIC SUB Main()  
  DIM X AS Integer  
  X = 7  
  INC X  
  PRINT X  
END
```

Die Ausgabe erfolgt im Direktfenster und lautet: 8

Zweites Beispiel mit einem Array

```
STATIC PUBLIC SUB Main()  
  DIM A[3, 3] AS Float  
  DIM X AS Integer  
  DIM Y AS Integer  
  
  X = 2  
  Y = 1  
  A[X, Y] = Pi  
  INC A[X, Y]  
  PRINT A[X, Y]  
END
```

Ausgabe: 4.14159265359

Man kann natürlich statt des INC Befehls auch einfach eins dazuzählen

```
INC X
```

Ist dasselbe wie

```
X = X + 1
```

Folgendes ist falsch und funktioniert nicht:

```
X = INC X
```

### 4.5.2 DEC Herunterzählen

Das Gegenteil von INC macht der Befehl DEC. Er zieht von der Zahl die 1 ab. DEC heißt auf englisch Decrement, d.h. herunterzählen.

### Beispiel im Terminalmodus (d.h. ohne Grafik)

```
STATIC PUBLIC SUB Main()  
  DIM X AS Integer  
  X = 7  
  DEC X  
  PRINT X  
END
```

Die Ausgabe erfolgt im Direktfenster und lautet: 6

```
STATIC PUBLIC SUB Main()  
  DIM A[3, 3] AS Float  
  DIM X AS Integer  
  DIM Y AS Integer  
  
  X = 2  
  Y = 1  
  A[X, Y] = Pi  
  DEC A[X, Y]  
  PRINT A[X, Y]  
END
```

Ausgabe: 2,14159265359

## 4.6 Vorzeichen und Absolutwert Sgn, Abs

### 4.6.1 Sgn (Signum) Vorzeichen einer Zahl

Der Befehl **Sgn** liefert einen Wert, der für das Vorzeichen einer Zahl steht. Das Argument **Zahl** kann ein beliebiger zulässiger numerischer Ausdruck sein. Dessen Vorzeichen bestimmt den von der Sgn-Funktion ausgegebenen Wert:

- Ist die Zahl  $> 0$  liefert  $\text{Sgn}(\text{Zahl})$  den Wert 1.
- Ist die Zahl  $= 0$  liefert  $\text{Sgn}(\text{Zahl})$  den Wert 0.
- Ist die Zahl  $< 0$  liefert  $\text{Sgn}(\text{Zahl})$  den Wert -1.

Einfache Beispiele:

```
PRINT Sgn(Pi)
```

Ergebnis: 1

```
PRINT Sgn(-Pi)
```

Ergebnis: -1

```
PRINT Sgn(0)
```

Ergebnis: 0

**Beispiel:**

```
PUBLIC SUB Form_Open()
  DIM x AS Integer
  DIM y AS Float

  FOR x = 1 TO 100
    y = Rnd(-5, 5)
    'Es werden 100 Zufallszahlen zwischen -5 und +5 erzeugt.

    SELECT CASE Sgn(y) '...auswerten.
      CASE 0 ' Wenn Zahl Null.
        PRINT y & " " & Sgn(y) & " Signum = Null."
      CASE 1 ' Wenn Zahl positiv.
        PRINT y & " " & Sgn(y) & " Signum = positive Zahl."
      CASE -1 ' Wenn Zahl negativ.
        PRINT y & " " & Sgn(y) & " Signum = negative Zahl."
    END SELECT
  NEXT
END
```

**Oder etwas anders:**

```
PUBLIC SUB Form_Open()
  DIM x AS Integer
  DIM y AS Float
  DIM pos AS Integer
  DIM neg AS Integer
  DIM nul AS Integer

  pos = 0
  neg = 0
  nul = 0

  FOR x = 1 TO 100
    y = Rnd(-5, 5)
    'Es werden 100 Zufallszahlen zwischen -5 und +5 erzeugt.

    SELECT CASE Sgn(y) '...auswerten.
      CASE 0 ' Wenn Zahl null.
        INC nul
      CASE 1 ' Wenn Zahl positiv.
        INC pos
      CASE -1 ' Wenn Zahl negativ.
        INC neg
    END SELECT
  NEXT
  PRINT nul,pos,neg
END
```

**Es geht natürlich auch ohne den INC Befehl:**

```
STATIC PUBLIC SUB Main()
  DIM x AS Integer
  DIM y AS Float
  DIM pos AS Integer
  DIM neg AS Integer
  DIM nul AS Integer
```

```

pos = 0
neg = 0
nul = 0

FOR x = 1 TO 100
  y = Rnd(-5, 5)
  'Es werden 100 Zufallszahlen zwischen -5 und +5 erzeugt.

  SELECT CASE Sgn(y) '...auswerten.
    CASE 0 ' Wenn Zahl null.
      nul = nul + 1
    CASE 1 ' Wenn Zahl positiv.
      pos = pos + 1
    CASE -1 ' Wenn Zahl negativ.
      neg = neg + 1
  END SELECT
NEXT
PRINT nul, pos, neg
END

```

Wie zu erwarten gibt es fast keine Treffer auf der Null, Erhöhen Sie die Zahl x und schauen Sie nach, ob irgendwann auch einmal die Null getroffen wird.

## 4.6.2 Absolutwert einer Zahl ABS

Der Befehl ABS liefert den Absoluten Betrag einer Zahl, d.h das Vorzeichen wird weggestrichen, wenn es negativ ist. In der Mathematik verwendet man dafür 2 senkrechte Striche  $|-2| = 2$ .

Kleine Beispiele dazu:

```
Print abs(-2)
```

Ergebnis: 2

```
Print abs(2)
```

Ergebnis: 2

```
Print abs(0)
```

Ergebnis: 0

Siehe auch: <http://www.madeasy.de/7/prgabs.htm>

Ein etwas längeres Beispiel im Terminalmodus:

```

STATIC PUBLIC SUB Main()
  DIM i AS Integer
  FOR i = -10 TO 10
    PRINT i & " => Abs(i) => " & Abs(i)
  NEXT
END

```

## 4.7 Der Val Befehl - Aus einem Text etwas Sinnvolles machen

Val() wandelt einen String (= Folge von Asciizeichen) in einen Datentyp um, der dem Inhalt des Strings entspricht. Wenn der String aussieht wie eine Fließkommazahl, dann macht Val() eine Fließkommazahl daraus. Val() benutzt die lokale Sprache ihrer Maschine (hier also deutsch), die in "locales" festgelegt wurde.

Sie können Ihre Einstellung mit folgendem Konsolenbefehl überprüfen: "locales" oder "echo \$LANG".

Auf einer deutschen Maschine erhält man: de\_DE@euro

Vorsicht: Mit Kommazahlen und der Ländereinstellung! Der Val Befehl ist fehlerträchtig!

Steht in einer Textbox1 6,6 mit deutscher Ländereinstellung, dann liefert Val(textbox1.text) eine Floatvariable mit dem Wert 6,6.

Steht in der Textbox 6.6 dann liefert Val(textbox1.text) ein Datum 06.06.04

Wollen Sie die Zahl wieder in einen deutschen Text verwandeln, dann nutzen Sie Str(zahl) zur Umwandlung.

Probieren Sie die Ländereinstellungen Ihres PCs mit folgendem Programm einfach einmal aus. Die Ausgabe erfolgt im Direktfenster:

```
PUBLIC SUB Form_Open()  
  PRINT Val("6.6.99")  
  PRINT Val("09/06/72 01:00")  
  PRINT Val("3.1415")  
  PRINT Val("3,1415")  
  PRINT Val("-25")  
  PRINT Val("True")  
  PRINT Val("False")  
  PRINT IsNull(Val("Gambas"))  
END
```

Bei deutscher Ländereinstellung ergibt sich:

```
06.06.99  
  
31415  
3,1415  
-25  
True  
False  
True
```

Weiteres Beispielprogramm

Siehe auch: <http://www.madeasy.de/7/prgval.htm>



Wenn Sie einen Textstring eingeben, dann wandelt das Programm den String mit Val um und die Art der umgewandelten Variable wird überprüft.

Das Ergebnis wird ausgegeben.

```

PUBLIC SUB _new()
    TextLabell.Text = "Tippen Sie etwas ein, das wie ein Datum, eine Zahl, eine Kommazahl oder eine
    Boolesche Variable aussieht"
END

PUBLIC SUB Button1_Click()
    x AS Variant
    x = Val(TextBox1.Text)

    IF IsDate(x) THEN Labell.Text="Das ist eine Datumsangabe!"
    IF IsBoolean(x) THEN Labell.Text="Das ist eine Boolesche Variable!"
    IF IsInteger(x) THEN Labell.Text="Das ist eine Zahl!"
    IF IsFloat(x) THEN Labell.Text="Das ist eine Kommazahl!"
    IF IsString(x) THEN Labell.Text="Das ist ein String!"
END

```

Vorsicht: Bei der Eingabe von True, wird zwar eine Boolesche Variable erkannt, der Wert wird aber in einer zweiten Textbox nicht ausgegeben. Bei der Eingabe von False, wird zwar eine Boolesche Variable erkannt, der Wert wird aber in einer zweiten Textbox nicht ausgegeben.

### 4.7.1 Theorie

Ausdruck = Val(String)

Konvertiert einen String in einen booleschen Wert, eine Zahl oder ein Datum, abhängig vom Inhalt des Strings. Bei der Umwandlung von Zahlen und Datumsangaben werden die gegenwärtigen local-Einstellungen berücksichtigt.

Die Umwandlung geschieht folgendermaßen:

- Wenn der String als *Datum und Zeit* (mit Datums- und Zeit-Trennzeichen) interpretiert werden kann, dann wird *Datum und Zeit* zurückgegeben.
- Andernfalls, wenn der String als *Fließkommazahl* interpretiert werden kann, dann wird eine *Fließkommazahl* zurückgegeben.
- Andernfalls, wenn der String als *Ganzzahl* interpretiert werden kann, dann wird eine *Ganzzahl* zurückgegeben.
- Andernfalls, wenn der String *TRUE* oder *FALSE* ist, dann wird der entsprechende *boolesche Wert* zurückgegeben.
- Ansonsten wird *NULL* zurückgegeben.

## 4.8 Eulersche Zahl berechnen

Siehe auch <http://www.madeasy.de/7/prge.htm> und <http://www.madeasy.de/2/e.htm>

Mit der folgenden Formel kann man die Eulersche Zahl berechnen.

$$e = 1 + 1/1! + 1/2! + 1/3! + 1/4! + \dots + 1/n!$$

Die Formel wird in ein einfaches Programm eingebaut. Die unendlich Folge wird im Beispiel nach 15 Schleifendurchgängen abgebrochen. Man kann die Zahl aber auch noch genauer berechnen.

Das Programm ist als Terminal Programm ohne Grafik codiert.

```
STATIC PUBLIC SUB Main()
  E AS Float
  QF AS Float
  F AS Integer
  K AS Integer

  E = 1
  F = 1
  FOR K = 1 TO 15
    F = F * K
    QF = CFloat(1)/F
    E = E + QF
    PRINT E
  NEXT
END
```

## 4.9 Fibonaccifolge berechnen

Siehe auch: <http://www.madeasy.de/7/prgfibo.htm>

Mit dem folgenden Programm kann man die Fibonaccifolge berechnen.

Es ist als Terminal Programm ohne Grafik codiert.

```
STATIC PUBLIC SUB Main()
  a AS Integer
  b AS Integer
  x AS Integer

  a = 1
  b = 1
  FOR x = 1 TO 23
    PRINT 2*x-1, a
    PRINT 2*x, b
    a = a + b
    b = a + b
  NEXT
END
```

```
NEXT  
END
```

**Ausgabe:**

```
1 1  
2 1  
3 2  
4 3  
5 5  
6 8  
7 13  
8 21  
9 34  
10 55  
11 89  
12 144  
13 233  
14 377  
15 610  
16 987  
17 1597  
18 2584  
19 4181  
20 6765  
21 10946
```

Versuchen Sie an Hand des obigen Codes, das Ganze als **Funktion** zu programmieren:

Eingabe der Funktion soll sein `fibonacci(n)`. Herauskommen soll die Fibonaccizahl, die zu dem entsprechenden N-Wert gehört. Allerdings muss man sicher sehr bald eine Größenbegrenzung einführen, da die Zahlen sehr schnell sehr groß werden.

## 4.10 Lucasfolge berechnen

Die Lucasfolge hat die gleiche Bildungsregel wie die Fibonacci-Folge. Sie unterscheidet sich allerdings in den beiden Anfangsgliedern. Statt 0 und 1 lauten die beiden Anfangsglieder 2 und 1.

Mit dem folgenden Programm kann man die Lucasfolge berechnen.

Es ist als Terminal Programm ohne Grafik codiert.

```
STATIC PUBLIC SUB Main()  
  a AS Integer  
  b AS Integer  
  x AS Integer  
  
  a = 2  
  b = 1  
  FOR x = 1 TO 23  
    PRINT 2*x-1, a
```

```
      PRINT 2*x, b
      a = a + b
      b = a + b
NEXT
END
```

### Ausgabe

```
1 2
2 1
3 3
4 4
5 7
6 11
7 18
8 29
9 47
10 76
11 123
12 199
13 322
14 521
15 843
16 1364
17 2207
18 3571
19 5778
20 9349
21 15127
22 24476
23 39603
```

## 4.11 Das Summenzeichen $\Sigma$ programmieren

Die Mathematik bereitet dem Anfänger schon deswegen erhebliche Schwierigkeiten, weil sie dazu neigt, neue komplizierte Zeichen zu benutzen. Ein Beispiel ist das Summenzeichen. Wenn man es verstehen will, sollte man es einfach programmieren. Programmiertechnisch ist es eine **For Next Schleife mit Aufsummierung des Ergebnisses aus jedem Schleifendurchlauf**.

Wie kann man das Zeichen programmieren?

Dazu benutzen wir 2 Beispiele:

### 4.11.1 Beispiel 1

$$\sum_{i=-2}^3 i \quad (4.1)$$

Das Programm dazu schaut so aus: Sie brauchen eine Textbox und einen Commandbutton, um es in Gang zu bringen.

Code:

```
PUBLIC SUB Form_Open()
    Button1.Text = "Berechne die Summe"
    TextBox1.Text = ""
END

PUBLIC SUB Button1_Click()
    i AS Integer
    'i ist die Laufvariable von -2 bis 3
    s AS Variant
    's ist die Summe
    s = 0
    FOR i = -2 TO 3
        s = s + i
        'PRINT i, s
    NEXT
    textbox1.Text = Str(s)
END
```

Verändern Sie das Programm so, dass Sie den Anfangswert und den Endwert der Laufvariablen *i* in eine Textbox eingeben können. Fügen Sie dazu in das Programm 2 neue Variablen ein

```
a as integer
' a = Anfangswert
e as integer
' e = Endwert
```

Ändern Sie die Schleife in

```
For i = a to e
```

Vorher müssen Sie noch *a* und *e* Werte zuweisen:

```
a = 0
e = 6
```

Oder:

```
a = val(TextBox2.Text)
e = val(TextBox3.Text)
```

## 4.11.2 Beispiel 1a

$$\sum_{i=1}^{10} i * i \quad (4.2)$$

Das Programm dazu schaut so aus: Sie brauchen eine Textbox und einen Commandbutton, um es in Gang zu bringen.

**Code:**

```
PUBLIC SUB Form_Open()  
    Button1.Text = "Berechne die Summe"  
    TextBox1.Text = ""  
END  
  
PUBLIC SUB Button1_Click()  
    i AS Integer  
    'i ist die Laufvariable von 1 bis 10  
    s AS Variant  
    s = 0  
    FOR i = 1 TO 10  
        s = s + i*i  
        'PRINT i, s  
    NEXT  
    textbox1.Text = Str(s)  
END
```

Verändern Sie das Programm so, dass Sie den Anfangswert und den Endwert der Laufvariablen *i* in eine Textbox eingeben können. Fügen Sie dazu in das Programm 2 neue Variablen ein

```
a as integer  
' a = Anfangswert  
e as integer  
' e = Endwert
```

Ändern Sie die Schleife in

```
For i = a to e
```

Vorher müssen Sie noch *a* und *e* Werte zuweisen:

```
a = 1  
e = 10
```

Oder:

```
a = val(TextBox2.Text)  
e = val(TextBox3.Text)
```

### 4.11.3 Beispiel 2

Berechnet werden soll

$$\sum_{i=0}^6 \frac{i}{i+1} \quad (4.3)$$

Das Programm dazu schaut so aus:

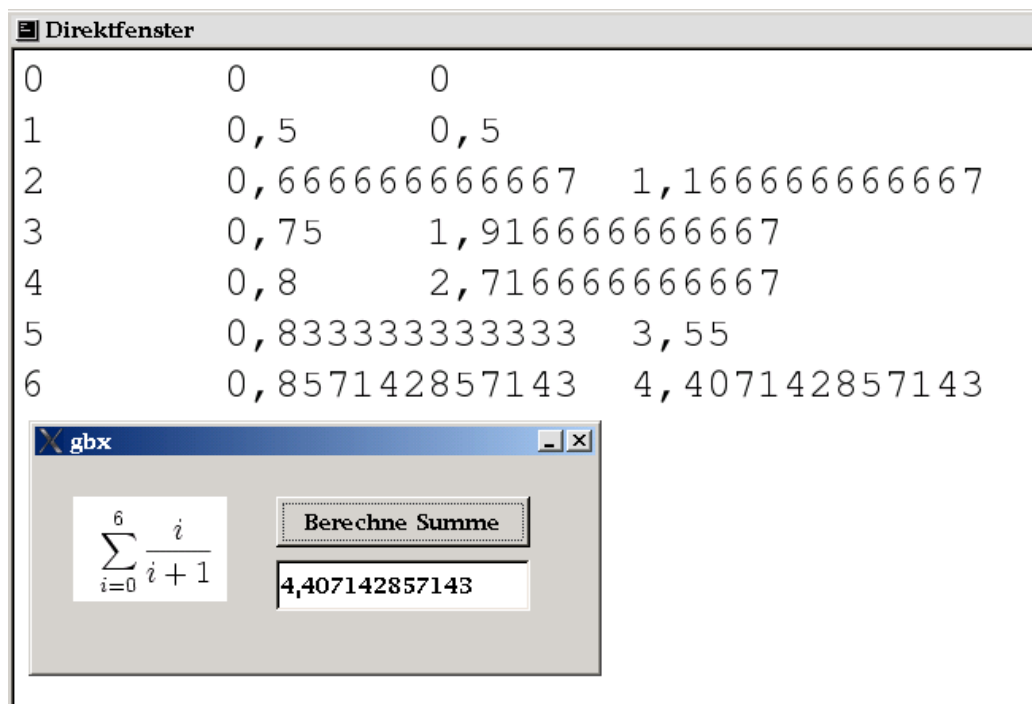


Abbildung 42: Bild:Summezeichen1.png

Die Zwischenwerte wurden dabei im Direktfenster mit Print ausgegeben. Sie brauchen eine Textbox und einen Commandbutton, um es in Gang zu bringen.

```

PUBLIC SUB Form_Open()
    Button1.Text = "Berechne Summe"
    Textbox1.Text = ""
END

PUBLIC SUB Button1_Click()
    i AS Integer
    'i ist die Laufvariable von 0 bis 6
    s AS Variant
    s = 0
    FOR i = 0 TO 6
        s = s + i / (i + 1)
        PRINT i, (i / (i + 1)), s
        '0 + 1/2 + 2/3 + 3/4 + 4/5 + 5/6 + 6/7 = 210/420 + 280/420 + 315/420 + 336/420 + 350/420 +
360/420
    NEXT
    textbox1.Text = Str(s)
END

```

Man erkennt den Unterschied der Computerberechnung und der eigenen Berechnung auf dem Papier.  $0 + 1/2 + 2/3 + 3/4 + 4/5 + 5/6 + 6/7 = 210/420 + 280/420 + 315/420 + 336/420 + 350/420 + 360/420$  Auf dem Papier sucht man den kleinsten

gemeinsamen Nenner. Dann erweitert man die Brüche entsprechend. Die Zähler kann man dann addieren. Ergebnis  $1851/420 = 617/140 = 4.407142857142857$

Der entscheidende Vorteil des Computerprogrammes ist die leichte Erweiterbarkeit. Man kann den Anfangswert 0 für  $i$  als Variable allgemein lassen. Man kann den Endwert für  $i = 6$  allgemein lassen. Dann eignet sich das Programm nicht nur für eine spezielle Summe, sondern gleich für beliebige Summen derselben Art.

```
PUBLIC SUB Form_Open()
    Button1.Text = "Berechne Summe"
    Textbox1.Text = ""
END

PUBLIC SUB Button1_Click()
    i AS Integer
    'i ist die Laufvariable von 0 bis 6
    a as integer
    'a ist der Anfangswert für i
    z as integer
    'z ist der Endwert für i
    s AS Variant

    s = 0
    a = 0
    z = 6
    FOR i = a TO z
        s = s + i / (i + 1)
        'PRINT i, (i / (i + 1)), s
    NEXT
    textbox1.Text = Str(s)
END
```

## 4.12 Zahlen zählen

Dieses Programm zählt aus einer langen Schlange von Dezimalzahlen die Häufigkeit von 1,2,3,4,5,6,7,8,9,0. Es ist in typischer Anfängermanier programmiert. Das ist keine Schande, denn es erfüllt seinen Zweck. Mit einem Array könnte man das Programm wahrscheinlich deutlich verkürzen.

Die Ergebnisausgabe erfolgt wie üblich im Direktfenster.

Um das Programm in Gang zu bringen brauchen Sie eine Textarea und 2 Commandbuttons auf ihrer Form.



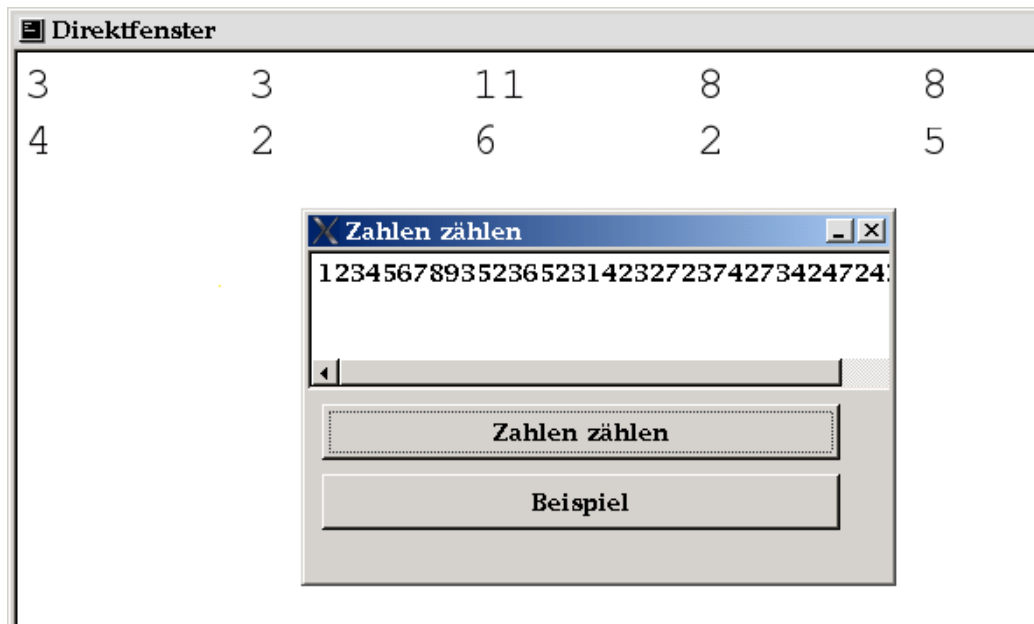


Abbildung 43: Screenshot

```

PUBLIC SUB Form_Open()
    Button1.Text = "Zahlen zählen"
    Button2.Text = "Beispiel"
    Textareal.Text = ""
    ME.Text = "Zahlen zählen"
END

```

```

PUBLIC SUB Button1_Click()
    s AS String
    n AS Integer
    Tempo AS String
    char AS String
    z0 AS Integer
    z1 AS Integer
    z2 AS Integer
    z3 AS Integer
    z4 AS Integer
    z5 AS Integer
    z6 AS Integer
    z7 AS Integer
    z8 AS Integer
    z9 AS Integer

    s = textareal.Text
    z0 = 0
    z1 = 0
    z2 = 0
    z3 = 0
    z4 = 0
    z5 = 0
    z6 = 0

```

```
z7 = 0
z8 = 0
z9 = 0

Tempo = RTrim$(s)
Tempo = LTrim$(Tempo)
FOR n = 1 TO Len(Temp)
  char = Mid$(Tempo, n, 1)
  SELECT CASE char
    CASE "0"
      z0 = z0 + 1
    CASE "1"
      z1 = z1 + 1
    CASE "2"
      z2 = z2 + 1
    CASE "3"
      z3 = z3 + 1
    CASE "4"
      z4 = z4 + 1
    CASE "5"
      z5 = z5 + 1
    CASE "6"
      z6 = z6 + 1
    CASE "7"
      z7 = z7 + 1
    CASE "8"
      z8 = z8 + 1
    CASE "9"
      z9 = z9 + 1
  END SELECT
NEXT
PRINT z0, z1, z2, z3, z4
PRINT z5, z6, z7, z8, z9
END

PUBLIC SUB Button2_Click()
  textareal.Text = "1234567893523652314232723742734247247899990001223445"
END
```

## 4.13 Funktionsdarstellung

Leider funktioniert der VB Befehl Scale in Gambas noch nicht. Trotzdem gibt es schon ein paar brauchbare Funktionsdarstellungen: <http://www.madeasy.de/7/prgkoord.htm> <http://www.madeasy.de/7/prgfunkt.htm>

## 4.14 Trigonometrische Funktionen in Gambas

Acsh / ACosh, Acs / ACos, Asn / ASin, Asnh / ASinh, Atn / ATan, Atnh / ATanh, Cos, Cosh, Deg, Rad, Pi, Sin, Sinh

### 4.14.1 Beispiel Programm für den sin und cos ohne Grafik

```

STATIC PUBLIC SUB Main()
  DIM a AS Float
  FOR a = - 10 TO 10 STEP 0.1
    PRINT a," Sin ",Sin(a),"      Cos ",Cos(a)
  NEXT
END

```

In der Ausgabe stecken noch einige Fehler. Der Wert  $a = 0$  wird beispielsweise nicht korrekt ausgegeben. Auch die Werte  $a = 0,1$  und  $a = 10$  machen Schwierigkeiten.

### 4.14.2 Grafische Ausgabe der Sinusfunktion

Siehe <http://www.madeasy.de/7/prgsinohne.htm>

Dieses Grafikprogramm zeigt die Programmierung der Sinusfunktion: Sie brauchen einen Befehlsknopf und ein Zeichenfeld. Stellen Sie die Höhe des Zeichenfeldes bei den Eigenschaften auf 100 ein (Height).

Hinter dem Befehl Sinuskurve steht folgender Code:

```

PUBLIC SUB Button1_Click()
  DIM dymax AS Integer
  ' maximaler gezeichneter y - Wert = Height drawymax
  DIM dymin AS Integer
  ' minimaler gezeichneter y - Wert = 0 drawymin
  DIM ymax AS Integer
  DIM ymin AS Integer
  'gewünschtes Koordinatensystem mit ymax und ymin
  DIM y AS Float
  'wechselnder Originalwert von y
  DIM dy AS Float
  'gezeichneter Wert von y
  DIM dyi AS Integer
  'Ganzzahlwert von dy
  DIM x AS Float
  DIM dxi AS Integer
  'Ganzzahlwert von dx

  ' Beispiel dymax = 100 und dymin = 0
  dymax = 100
  dymin = 0
  ymax = 1
  ymin = -1

  FOR x = 0 TO 12 STEP 0.1
    y = Sin(x)
    'Formel
    'Verhältnis Teilstrecke / Gesamtstrecke in beiden Maßstäben sind gleich
    '(dy - dymin) / (dymax - dymin) = (y - ymin) / (ymax - ymin)
    '==> dy - dymin = (y - ymin) / (ymax - ymin) * (dymax - dymin)
    '==> dy = (y - ymin) / (ymax - ymin) * (dymax - dymin) + dymin
  
```

```
dy = CFloat(y - ymin) / (ymax - ymin) * (dymax - dymin) + dymin
'PRINT x,y,dy
dyi = Fix(dy)
dxi = Fix(x *30)
Draw.Begin(DrawingArea1)
Draw.Point(dxi,dyi)
Draw.End
NEXT
END
```

Spielen Sie mit den einzelnen Größen etwas herum und beobachten Sie wie sich die Grafik ändert. Verändern Sie das Programm so, dass der Sinus korrekt mit 0 anfängt und erst ins positive wächst. Was ist hier noch falsch? (dy Werte werden noch von oben nach unten aufgetragen und nicht umgekehrt.) Versuchen Sie die Umrechnungen für dyi alle in eine Formel zu bekommen.

## 4.15 Logik

Siehe [Gambas: Logische Operatoren](#)

## 4.16 Primzahlen berechnen

Das folgende Programm zeigt die Suche nach Primzahlen mit Hilfe des [Siebs von Eratosthenes](#). Die Ergebnisausgabe erfolgt im Direktfenster.

Um es in Gang zu bringen brauchen Sie 2 Commandbuttons und 1 Textbox auf Ihrer Form.

```
PUBLIC SUB Form_Open()
    ME.Text = "Primzahlberechnung"
END

PUBLIC SUB Button1_Click()
    m AS Integer
    f AS Integer
    n AS Integer
    A$ AS String

    if textbox1.Text = "" Then textbox1.text = "1"
    m = Val(textbox1.Text)
    'holt sich aus dem Textfeld1 die erste Zahl zum Testen

    IF m / 2 = Int(m / 2) THEN m = m - 1
    'Falls diese Zahl ohne Rest durch 2 teilbar ist, also eine gerade Zahl ist
    'geht das Programm noch eine Zahl rückwärts um eine ungerade Zahl zu bekommen
```

```

IF textbox1.Text = "" THEN m = 6
'Falls keine Startzahl eingegeben wurde wird die Startzahl m = 6 vergeben.

FOR m = m TO m + 1000 STEP 2
'Hauptschleife
'Die nächsten 1000 ungeraden Zahlen in einer Schleife durchlaufen lassen
'm ist die Variable für die ungeraden Zahlen

f = 1
'f ist die Variable für die Faktorentestung,
'm teilbar durch f oder nicht

n = m
'n = ist die Testzahl, bei der noch nicht klar ist ob sie eine Primzahl ist

DO WHILE f < Sqr(n)
'solange der Teiler f kleiner als die Wurzel
'aus der Testzahl n ist, muss getestet werden
f = f + 2
'Teiler von f = 1 beginnend um jeweils 2 vermehren, 3,5,7,9 etc erster Test also mit f = 3
DO WHILE n / f = Int(n / f)
'Teiler f testen solange bis n / f ohne Rest teilbar
'PRINT "m = "; m; " n = "; n; " f = "; f
n = Int(n / f)
'Die Testzahl n verkleinern auf die Zahl n/f
LOOP
LOOP

A$ = " 1" & Chr(13) & " 2" & Chr(13) & " 3"
IF m < 7 THEN PRINT A$
'Chr(13) = Zeilenwechsel
'Am Anfang zwischen 1 und 5 gibt es Probleme mit der Ausgabe
'deswegen werden die ersten drei Zeilen ersetzt durch A$

IF n = m THEN PRINT n
'Wenn die Testzahl n nicht teilbar war durch f
'ist es eine Primzahl und kann ausgedruckt werden
NEXT
CATCH
message.Info("Bitte eine Zahl eingeben")
END

```

## 4.17 Logarithmus

Mit dem Logarithmus kann man sehr große Zahlen übersichtlich und klein machen. Aus 1 000 000 wird beispielsweise  $10^6$  (Sprich 10 hoch 6 =  $10 * 10 * 10 * 10 * 10 * 10$ ). Dabei ist die Hochzahl 6 der Logarithmus von 1 000 000 zur hier gewählten Basiszahl 10. Der Begriff Hochzahl und Exponent sind identisch.

Die Logarithmusfunktion  $y = \log_{\langle \text{sub} \rangle b \langle \text{sub} \rangle} (x)$  liefert für jede beliebige positive Zahl  $x$  einen Wert. So ist der Logarithmus der Zahl 1 234 567 zur Basis 10 gleich  $\log_{\langle \text{sub} \rangle 10 \langle \text{sub} \rangle} (1234567) = 6.09151466408626$  Insbesondere für Werte, die sehr klein oder sehr groß sind oder die einen sehr großen Wertebereich

einnehmen können, haben sich Logarithmen sehr bewährt. Da die Logarithmusfunktion mit größer werden  $x$  Werten stetig wächst, bleiben logische Zusammenhänge, die für den  $x$  Wert gelten meist auch für den Logarithmus von  $x$  erhalten.

#### 4.17.1 Anwendung von Logarithmen:

- pH = Säurewert von chemischen Lösungen
- dB = Dezibel = Messung der Lautstärke
- bit = Informationseinheit = Messung der Informationsmenge
- Logarithmuspapier in der Statistik

Siehe auch:

- <http://www.madeasy.de/2/log.htm>
- [Mathematik: Schulmathematik: Logarithmieren](#)
- [w:Logarithmus](#)

#### 4.17.2 Beispielprogramm Log Befehl

Um den Befehl Log zu verstehen, kann man ihn einfach für verschiedene Werte berechnen:  $L = \log(x)$   $L$  ist der natürliche Logarithmus von  $x$  zur Basis [e](#), der [Eulerschen Zahl](#)

$e = 2,718281828459$

Man schreibt sich ein Miniprogramm mit einer Form und einem Befehlsbutton:

Der Code dazu lautet:

```
PUBLIC SUB Form_Open()  
    Button1.Text = "log"  
END  
  
PUBLIC SUB Button1_Click()  
    x AS Integer  
    FOR x = 1 TO 10  
        PRINT x, Log(x)  
    NEXT  
END
```

Die Ausgabe erfolgt im Direktfenster. Sie lautet:

```
1 0  
2 0,69314718056  
3 1,098612288668
```

```
4 1,38629436112
5 1,609437912434
6 1,791759469228
7 1,945910149055
8 2,07944154168
9 2,197224577336
10 2,302585092994
```

Wenn Sie Ihre Schleife für x mit einer negativen Zahl anfangen lassen, erhalten Sie eine Fehlermeldung:

```
Mathematic Error
```

Denn Logarithmus einer negativen Zahl gibt es nicht.

### Genauere Version

Wenn man die Schleife in genaueren Schritten laufen lassen will, benutzt man folgende Codevariante:

```
PUBLIC SUB Form_Open()
    Button1.Text = "log"
END

PUBLIC SUB Button1_Click()
    x AS Float
    FOR x = 0.1 TO 10 STEP 0.1
        PRINT x, Log(x)
    NEXT
END
```

Ausgabe im Direktfenster:

```
0,1 -2,302585092994
0,2 -1,609437912434
0,3 -1,203972804326
0,4 -0,916290731874
0,5 -0,69314718056
0,6 -0,510825623766
0,7 -0,356674943939
0,8 -0,223143551314
0,9 -0,105360515658
1 -1,110223024625E-16
1,1 0,095310179804
1,2 0,182321556794
1,3 0,262364264467
1,4 0,336472236621
```

etc...

### 4.17.3 Beispielprogramm Logbefehl mit beliebiger Eingabe

Wenn man den Log Befehl mit einer beliebigen positiven Zahl testen will, kann man folgendes Programm nutzen.

Sie brauchen 2 Textboxen und 1 Befehlsknopf (Commandbutton) auf Ihrer Form. Machen Sie die Textbox2 groß genug, damit alle Nachkommastellen Platz haben.

Code:

```
PUBLIC SUB Form_Open()  
    ME.Text = "Was macht der Log Befehl"  
    Textbox1.Text = ""  
    Textbox2.Text = ""  
    Button1.Text = "Log"  
END  
  
PUBLIC SUB Button1_Click()  
    x AS Float  
    L AS Float  
  
    IF Textbox1.Text = "" THEN textbox1.text = "1"  
    IF Val(textbox1.Text) <= 0 THEN  
        message.info = "Unzulässige Eingabe"  
        RETURN  
    ENDIF  
  
    'Umformen der Texteingabe 1 in eine genaue Zahl  
    x = Val(textbox1.Text)  
    L = Log(x)  
    'Ausgabe des Ergebnises als String im Textfeld 2  
    textbox2.Text = Str(L)  
END
```

Im Code dieses Programms steckt noch ein Fehler. Die MessageBox (Meldung) wurde falsch programmiert:

Statt:

```
IF Val(textbox1.Text) <= 0 THEN message.info = "Unzulässige Eingabe"
```

Sollten Sie schreiben:

```
IF Val(textbox1.Text) <= 0 THEN Message.info("Unzulässige Eingabe Zahl kleiner Null", "OK")
```

Wie kann man noch Fehleingaben von Buchstaben verhindern?

### 4.17.4 Beispielprogramm Dualer Logarithmus

Der Duale Logarithmus nutzt als Basis die Zahl 2. Um ihn mit dem Befehl log zu erhalten, muss man den natürlichen in den dualen umrechnen. dazu nutzt man folgende Formel:

$$L = \log_{2}x = \log_{e}x / \log_{e}2$$



Code:

```

PUBLIC SUB Form_Open()
    ME.Text = "Dualer Logarithmus"
    Textbox1.Text = ""
    Textbox2.Text = ""
    Button1.Text = "Log2"
END

PUBLIC SUB Button1_Click()
    x AS Float
    L AS Float

    x = Val(textbox1.Text)
    L = Log(x)/Log(2)
    'Ausgabe des Ergebnisses als String im Textfeld 2
    textbox2.Text = Str(L)
END

```

#### 4.17.5 Beispielprogramm Dekadischer Logarithmus

Der dekadische Logarithmus nutzt als Basis die Zahl 10. Um ihn mit dem Befehl `log` zu erhalten, muss man den natürlichen in den dekadischen umrechnen. dazu nutzt man folgende Formel:

$$L = \log_{10} x = \frac{\log_e x}{\log_e 10}$$

Code:

```

PUBLIC SUB Form_Open()
    ME.Text = "Dualer Logarithmus"
    Textbox1.Text = ""
    Textbox2.Text = ""
    Button1.Text = "Log2"
END

PUBLIC SUB Button1_Click()
    x AS Float
    L AS Float

    x = Val(textbox1.Text)
    L = Log(x)/Log(10)
    'Ausgabe des Ergebnisses als String im Textfeld 2
    textbox2.Text = Str(L)
END

```

#### 4.17.6 Beispielprogramm Logarithmusrechner

Zum Berechnen von Logarithmen kann man folgendes kleine Programm nutzen. Um es in Gang zu bringen braucht man:

- 5 Commandbuttons

- 4 Textboxen
- 3 Textlabelfelder

Abbildung 44: Bild:Prglogrechner.png

Der Code:

```

PUBLIC SUB Form_Open()
    'automatisches Benennen der Steuerelemente beim Programmstart
    Textbox1.Text = ""
    Textbox2.Text = ""
    Textbox3.Text = ""
    textbox4.Text = ""
    Button1.Text = "Basis 2"
    Button2.Text = "Basis e"
    Button3.Text = "Basis 10"
    Button4.Text = "Berechnen"
    Button5.Text = "Ende"
    Textlabel1.Text = "Der Logarithmus der Zahl"
    Textlabel2.Text = "zur Basis "
    Textlabel3.Text = "beträgt: "
    Textlabel1.Alignment = 66
    Textlabel2.Alignment = 66
    Textlabel3.Alignment = 66
    Textlabel1.BackColor = &Hffffff&
    Textlabel2.BackColor = &HFFFFFF&
    Textlabel3.BackColor = &HFFFFFF&
    Textlabel1.Border = 1

```

```
Textlabel2.Border = 1
Textlabel3.Border = 1
Form1.Move(174,455,393,242)
Form1.Border = Window.Fixed
Button1.Move(80,64,64,32)
Button2.Move(160,64,64,32)
Button3.Move(240,64,64,32)
Button4.Move(272,112,112,32)
Button5.Move(272,192,112,32)
TextBox1.Move(240,16,144,40)
TextBox2.Move(120,112,144,32)
TextBox3.Move(120,152,144,32)
TextBox4.Move(24,192,240,32)
TextLabel1.Move(8,16,216,40)
TextLabel2.Move(24,112,96,32)
TextLabel3.Move(24,152,96,32)
END

PUBLIC SUB Button4_Click()
    x AS Float
    y AS Float
    L AS Float

    textbox4.Text = ""
    'Fehleingaben abfangen
    IF TextBox1.Text = "" OR TextBox2.Text = "" THEN textbox4.Text = "Eingabe fehlt!"
    IF TextBox1.Text = "" OR TextBox2.Text = "" THEN RETURN
    IF Val(textbox1.Text) <= 0 THEN textbox4.Text = "Unzulässige Eingabe"
    IF Val(textbox1.Text) <= 0 THEN RETURN
    IF Val(textbox2.Text) <= 1 THEN textbox4.Text = "Unzulässige Eingabe"
    IF Val(textbox2.Text) <= 1 THEN RETURN

    'Umformen der Texteingabe 1 und 2 in eine genaue Zahl
    x = Val(textbox1.Text)
    y = Val(textbox2.Text)

    'allgemeine Logarithmusformel
    L = Log(x) / Log(y)

    'Ausgabe des Ergebnisses im Textfeld 3
    textbox3.Text = Str(L)
END

PUBLIC SUB Button1_Click()
    textbox2.Text = 2
END

PUBLIC SUB Button3_Click()
    textbox2.Text = 10
END

PUBLIC SUB Button2_Click()
    textbox2.Text = "2,718282"
END

PUBLIC SUB Button5_Click()
    ME.Close
END
```

## 4.18 Zinsrechnung

Die im Folgenden aufgeführten Formeln für die Zinsrechnung verwenden Symbole wie folgt:

1. Anfangskapital:  $K_{<sub>0</sub>}$  (Kapital nach 0 Jahren)
2. Endkapital:  $K_{<sub>n</sub>}$  (Kapital nach n Jahren)
3. Laufzeit (ganze Jahre): n Eingabe in Jahren
4. Laufzeit (Tage) : t Eingabe in Tagen
5. Zinssatz als Dezimalangabe:  $i = \frac{P}{100\%}$  (pro Zinsperiode)

### 4.18.1 Beispielprogramm Einfache Zinsen ohne Zinseszinsen (lineare Verzinsung)

Um einfache Zinsen auszurechnen verwenden wir die folgende Formel:

$$K_n = K_0 * (i * n + 1)$$

Für das Beispielprogramm braucht sie :

- 4 Valueboxen
- 1 Befehlsknopf (Commandbutton)

oder verwenden sie den folgenden .form Code

```
# Gambas Form File 2.0
{ Form Form
  MoveScaled(0,0,53,31)
  Text = ("")
  { Label1 Label
    MoveScaled(45,8,6,3)
    Text = ("Jahre")
  }
  { Label2 Label
    MoveScaled(45,13,4,3)
    Text = ("%")
  }
  { Label3 Label
    MoveScaled(1,3,14,3)
    Text = ("Anfangskapital")
  }
  { Label4 Label
    MoveScaled(4,8,11,3)
    Text = ("Laufzeit")
  }
  { Label5 Label
    MoveScaled(4,13,11,3)
```

```

    Text = ("Zinssatz")
}
{ Label6 Label
  MoveScaled(1,25,14,4)
  Text = ("Endkapital")
}
{ Button1 Button
  MoveScaled(7,18,38,5)
  Text = ("Berechnen")
}
{ ValueBox1 ValueBox
  MoveScaled(16,3,28,3)
}
{ Label7 Label
  MoveScaled(45,3,6,3)
  Text = ("Euro")
}
{ ValueBox2 ValueBox
  MoveScaled(16,8,28,3)
}
{ ValueBox3 ValueBox
  MoveScaled(16,13,28,3)
}
{ ValueBox4 ValueBox
  MoveScaled(16,25,28,4)
}
{ Label8 Label
  MoveScaled(45,25,6,3)
  Text = ("Euro")
}
}
}

```

## Der .class Code

```

' Gambas class file

'K (Anfangskapital)      = ValueBox1.Value
'n (Laufzeit in Jahren) = ValueBox2.Value
'i (Zinssatz)           = ValueBox3.Value / 100%
'Kn (Endkapital)        = ValueBox4.Value

PUBLIC SUB Button1_Click()

  'Endkapital berechnen mit der Formel:
  ' Kn = K*(i*n+1)
  ValueBox4.Value = (ValueBox1.Value * (ValueBox3.Value / 100 * ValueBox2.Value + 1))

END

```

## 4.18.2 Beispielprogramm Zinseszinsrechnung (exponentielle Verzinsung)

Um Zinsen mit Zinseszins auszurechnen verwenden wir die folgende Formel:

$$K_n = K_0 * (i + 1)^n$$

Für das Beispielprogramm brauchen sie :

- 5 Valueboxen
- 1 Checkboxe
- 1 Befehlsknopf (Commandbutton)

oder verwenden sie den folgenden .form Code

```
# Gambas Form File 2.0

{ Form Form
  MoveScaled(0,0,53,39)
  Text = ("")
  { Label1 Label
    MoveScaled(45,8,6,3)
    Text = ("Jahre")
  }
  { Label2 Label
    MoveScaled(45,13,4,3)
    Text = ("%")
  }
  { Label3 Label
    MoveScaled(1,3,14,3)
    Text = ("Anfangskapital")
  }
  { Label4 Label
    MoveScaled(4,8,11,3)
    Text = ("Laufzeit")
  }
  { Label5 Label
    MoveScaled(4,13,11,3)
    Text = ("Zinssatz")
  }
  { Label6 Label
    MoveScaled(1,28,14,4)
    Text = ("Endkapital")
  }
  { Button1 Button
    MoveScaled(7,22,38,5)
    Text = ("Berechnen")
  }
  { ValueBox1 ValueBox
    MoveScaled(16,3,28,3)
  }
  { Label7 Label
    MoveScaled(45,3,6,3)
    Text = ("Euro")
  }
  { ValueBox2 ValueBox
    MoveScaled(16,8,28,3)
  }
  { ValueBox3 ValueBox
    MoveScaled(16,13,28,3)
  }
  { ValueBox4 ValueBox
    MoveScaled(16,28,28,4)
  }
  { Label8 Label
    MoveScaled(45,28,6,3)
```

```

    Text = ("Euro")
}
{ Label9 Label
  MoveScaled(1,34,14,3)
  Text = ("Zinsen")
}
{ ValueBox5 ValueBox
  MoveScaled(16,34,28,3)
}
{ Label10 Label
  MoveScaled(45,34,6,3)
  Text = ("Euro")
}
{ CheckBox1 CheckBox
  MoveScaled(16,18,27,2)
  Text = ("Mit Zinseszins")
}
}
}

```

## Der .class Code

```

' Gambas class file

'K (Anfangskapital)      = ValueBox1.Value
'n (Laufzeit in Jahren) = ValueBox2.Value
'i (Zinssatz)           = ValueBox3.Value / 100%
'Kn (Endkapital)        = ValueBox4.Value
'Z (Zinsen/Zinswert)    = ValueBox5.Value

PUBLIC SUB Button1_Click()

  IF CheckBox1.Value = FALSE THEN
    'Normale Zinsen berechnen

    'Endkapital berechnen, mit der Formel:
    '  $Kn = K \cdot (i \cdot n + 1)$ 
    ValueBox4.Value = Round(ValueBox1.Value * (ValueBox3.Value / 100 * ValueBox2.Value + 1), -2)
    'Zinsen/Zinswert berechnen, mit der Formel:
    '  $Z = K \cdot (i \cdot n + 1) - K = Kn - K$ 
    ValueBox5.Value = Round(ValueBox4.Value - ValueBox1.Value, -2)

  ELSE
    'Zinseszins berechnen

    'Endkapital berechnen, mit der Formel:
    '  $Kn = K \cdot (i + 1)^n$ 
    ValueBox4.Value = Round(ValueBox1.Value * (ValueBox3.Value / 100 + 1) ^ ValueBox2.Value, -2)
    'Zinsen/Zinswert berechnen, mit der Formel:
    '  $Z = K \cdot (i + 1)^n - K = Kn - K$ 
    ValueBox5.Value = Round(ValueBox4.Value - ValueBox1.Value, -2)

  ENDIF

END

```

–Held03 23:57, 3. Jun. 2009 (CEST)

## 4.19 Beschreibende Statistik

### 4.19.1 Mittelwert

Das arithmetische Mittel (auch Durchschnitt) ist der am häufigsten benutzte Mittelwert und wird deshalb auch als Standardmittelwert bezeichnet.

Den arithmetischen Mittelwert einer Zahlenreihe können Sie berechnen, wenn Sie alle Elemente der Reihe zusammenzählen und die erhaltene Summe durch die Zahl der Elemente teilen.

Mittelwert = Summe der Werte / Anzahl der Werte

Beispiel für das arithmetische Mittel von 3,4 und 5:

Mittelwert = (3 + 4 + 5) / 3 = 4

Im folgenden Programmbeispiel wird dies umgesetzt.

Holen Sie sich aus der Werkzeugkiste eine Textarea, eine Textbox und einen Commandbutton. Diese ordnen Sie wie unten auf der Form an: Die Zahlen sind nur als Beispiel gedacht.

```

Textareal
*****
* 5 *
* 5 *
* 4 *
* 5 *
* *
* *
*****
Befehlsbutton1:
*****
*Mittelwert*
*****
Textbox1
*****
*4.75 *
*****

```

Der Code hinter dem Commandbutton *Mittelwert* schaut so aus und erinnert stark an ADDY, den Summierer:

```

PUBLIC SUB Button1_Click()
DIM text AS String
DIM summe AS Float
DIM mw AS Float
DIM elt AS String[]
DIM sb AS String
text = textareal.Text
elt = Split(text,Chr(10))
FOR EACH sb IN elt
summe = summe + Val(sb)
NEXT
mw = summe / elt.Length
textbox1.Text = mw

```



END

Das Programm nutzt als wesentliche Tricks einen Stringarray, den Befehl Split und das Zeichen CHR(10) um die einzelnen Zahlenglieder zu trennen.

### **4.19.2 Varianz und Standardabweichung**

Das Programm zur Berechnung des Mittelwertes kann man erweitern, so daß auch die Varianz und die Standardabweichung errechnet werden. Ein Testdatensatz ist auch dabei.

Bei der Gaußschen Normalverteilung ergibt sich, dass die Wahrscheinlichkeit etwa  $2/3$  dafür beträgt, dass die Messwerte innerhalb von  $\pm$  einer Standardabweichung vom Erwartungswert liegen und damit jeweils etwa  $1/6$  dafür, dass sie unterhalb bzw. oberhalb dieses Intervalls liegt.

Sie brauchen 3 Commandbuttons , eine Textarea und 2 Textboxen um es in Gang zu bringen.



Abbildung 45: Screenshot Mittelwert, Varianz und Standardabweichung berechnen

Der Code schaut so aus:

```
PUBLIC SUB Button1_Click()
```

```

DIM text AS String
DIM summe AS Float
DIM mw AS Float
DIM varianz AS Float
DIM sigma AS Float
DIM Liste AS String[]
DIM Einzelwert AS String
text = textareal.Text
Liste = Split(text,Chr(10))
FOR EACH Einzelwert IN Liste
    summe = summe + Val(Einzelwert)
NEXT
mw = summe / Liste.Length
textbox1.Text = mw
varianz = 0
FOR EACH Einzelwert IN Liste
    varianz = varianz + ((Val(Einzelwert) - mw)^2)
    'PRINT Einzelwert,mw,((Val(Einzelwert) - mw)^2)
    'PRINT varianz
NEXT
varianz = varianz / (Liste.Length - 1)
'PRINT varianz
sigma = Sqr(varianz)
textbox2.Text = sigma
END
-----
PUBLIC SUB Button2_Click()
'Testdatensatz in die Textarea füllen.
textareal.Text = "114,3"
textareal.Text = textareal.Text & Chr(10) & "135,7"
textareal.Text = textareal.Text & Chr(10) & "104,8"
textareal.Text = textareal.Text & Chr(10) & "118,5"
textareal.Text = textareal.Text & Chr(10) & "125,7"
textareal.Text = textareal.Text & Chr(10) & "121,4"
textareal.Text = textareal.Text & Chr(10) & "122,4"
textareal.Text = textareal.Text & Chr(10) & "96,8"
textareal.Text = textareal.Text & Chr(10) & "118,9"
textareal.Text = textareal.Text & Chr(10) & "120"
textareal.Text = textareal.Text & Chr(10) & "112,2"
textareal.Text = textareal.Text & Chr(10) & "127,9"
textareal.Text = textareal.Text & Chr(10) & "122,8"
textareal.Text = textareal.Text & Chr(10) & "128,9"
textareal.Text = textareal.Text & Chr(10) & "120,3"
'mw = 119,373
'varianz = 92,6016
'sigma = 9,62297
END
-----
PUBLIC SUB Button3_Click()
ME.Close
END

```

Einem Profiprogrammierer dreht es den Magen um, wenn er sieht, wie der Testdatensatz geladen wird. Aber es funktioniert. Und als kleine programmiertechnische Herausforderung kann man das Laden des Testdatensatzes mit dem Array Befehl deutlich verkürzen.

Siehe [Gambas: Arrays](#)

### 4.19.3 Der Median

Der Medianwert oder Zentralwert einer Werteliste liegt zwischen zwei gleich großen Hälften der Liste. 50 % der Werte sind größer/gleich als der Median, 50 % sind kleiner/gleich als der Median. Im Gegensatz zum arithmetischen Mittelwert ist er durch Extremwerte (Ausreißer) in der Werteliste kaum beeinflussbar.

Will man den Median programmieren muß man:

- 1. Die Werteliste sortieren
- 2. Überprüfen ob es sich um eine gerade Anzahl oder eine ungerade Anzahl von Werten handelt.
- 3. Die entsprechenden Formeln anwenden, um den Median zu berechnen.

– Bei ungerader Zahl an Werten

$$x_{\text{median}} = x[(n+1)/2]$$

- – Bei gerader Zahl an Werten

$$x_{\text{median}} = 1/2 * (x[n/2] + x[n/2+1])$$

Beispiel 1:

sortierte Werteliste:

```
11
12
13
14
15
```

Die Zahl der Werte ist ungerade ,  $n = 5$

$$(n+1)/2 = 6/2 = 3$$

$$x_{\text{median}} = x[3] = 13$$

Beispiel 2.

sortierte Werteliste:

```
11
12
13
14
15
16
```

Die Zahl der Werte ist gerade ,  $n = 6$

$$n/2 = 6/2 = 3$$

$$x[3] = 13$$

$$x[4] = 14$$

$$x_{\text{median}} = 1/2 * (13+14) = 13,5$$

Sie können obige Werteliste direkt in Ihr Beispielprogramm übernehmen: Nutzen Sie dazu den Zwischenspeicher: Markieren, Strg+C drücken. In Ihrem Programm mit Strg+V wieder abladen. Sortieren und Median ausrechnen. Klappt es ?

**Beispielprogramm:** Sie brauchen 2 Textareas , 3 Commandbuttons und eine Textbox, um das Programm zu starten:

```
' Gambas CLASS file
liste AS String[]
PUBLIC SUB Form_Open()
  ME.Text = "Zentralwert berechnen"
END
PUBLIC SUB Button1_Click()
'umsortieren
c AS Integer
j AS Integer
n AS Integer
y AS Variant
liste AS String[]
element AS String
txt AS String
text AS String
text = Textareal.Text
liste = Split(text,Chr(10))
y = 0
n = liste.length
REPEAT
c = 0
FOR j = 0 TO n - 2
  'PRINT j,y,liste[0],ar[1],ar[2],ar[3],ar[4]
  IF Val(liste[j]) > Val(liste[j + 1]) THEN
    y = Val(liste[j])
    liste[j] = liste[j + 1]
    liste[j + 1] = Str(y)
    c = 1
  ENDIF
NEXT
UNTIL c = 0
FOR EACH element IN liste
txt = txt & Str(element) & Chr(10)
NEXT
PRINT txt
textarea2.Text = ""
txt = Mid$(txt,1,-1)
'Der letzte Zeilenumbruch chr(10) muß weg
textarea2.Text = txt
END
PUBLIC SUB Button2_Click()
'median berechnen, vorher umsortieren notwendig !!
text AS String
median AS Float
liste AS String[]
posten AS String
text = Textarea2.Text
liste = Split(text,Chr(10))
'Länge bestimmen und richtige Formel verwenden,
'je nachdem ob die Länge gerade oder ungerade ist.
IF liste.Length MOD 2 THEN
  'PRINT liste.Length MOD 2 & " Ungerade"
  'PRINT (liste.length + 1)/2
  'PRINT liste[(liste.length + 1)/2 - 1]
```

```
median = Val(liste[(liste.length + 1)/2 - 1])
'Der Array beginnt mit dem Element 0 nicht mit 1 !
ELSE
'PRINT liste.Length MOD 2 & " Gerade"
median = (Val(liste[liste.length/2 - 1]) + Val(liste[liste.length/2]))/2
'Der Array beginnt mit dem Element 0 nicht mit 1 !
ENDIF
textbox1.Text = Str(median)
END
PUBLIC SUB Button3_Click()
'Ungerade Werteliste als Beispiel
textareal.Text = "114,3"
textareal.Text = textareal.Text & Chr(10) & "135,7"
textareal.Text = textareal.Text & Chr(10) & "104,8"
textareal.Text = textareal.Text & Chr(10) & "118,5"
textareal.Text = textareal.Text & Chr(10) & "125,7"
textareal.Text = textareal.Text & Chr(10) & "121,4"
textareal.Text = textareal.Text & Chr(10) & "122,4"
textareal.Text = textareal.Text & Chr(10) & "96,8"
textareal.Text = textareal.Text & Chr(10) & "118,9"
textareal.Text = textareal.Text & Chr(10) & "120"
textareal.Text = textareal.Text & Chr(10) & "112,2"
textareal.Text = textareal.Text & Chr(10) & "127,9"
textareal.Text = textareal.Text & Chr(10) & "122,8"
textareal.Text = textareal.Text & Chr(10) & "128,9"
textareal.Text = textareal.Text & Chr(10) & "120,3"
'median = 120,3
END
```

## 4.20 Gausssche Normalverteilung

Das folgende Programm berechnet die Werte der Gaussschen Normalverteilung zwischen -5 und 5. Siehe auch: <http://www.madeasy.de/2/gauss.htm>

```
PUBLIC SUB Form_Open()
DIM x AS Float
DIM y AS Float
DIM t AS String
FOR x = -5 TO 5 STEP 0.1
y = 2.718 ^ (-x * x)
t = t & Str(x) & " " & Str(y) & Chr(13) & Chr(10)
NEXT
PRINT t
END
```

## 4.21 Tests

### 4.21.1 Vierfeldertest

Der Vierfeldertest ist einfaches, aber sehr wichtiges Werkzeug zur statistischen Bewertung von Merkmalen in 2 Gruppen. Siehe auch <http://www.madeasy.de/7/4feld.htm>

Zur Ausführung des Programmes brauchen Sie:

- 4 Labelfelder
- 4 Textboxen
- 1 Textarea
- 3 Commandbuttons.

Der Code lautet:

```
PUBLIC SUB Form_Open()  
  ME.Text = "Vierfeldertest"  
END  
PUBLIC SUB Button1_Click()  
  ME.Close  
END  
PUBLIC SUB Button2_Click()  
  textbox1.Text = ""  
  textbox2.Text = ""  
  textbox3.Text = ""  
  textbox4.Text = ""  
  textareal.Text = ""  
END  
PUBLIC SUB Button3_Click()  
  ' Rem Fehleroutine bei leeren Feldern einbauen  
  DIM N1 AS Float  
  DIM N2 AS Float  
  DIM M1 AS Float  
  DIM M2 AS Float  
  DIM R1 AS Float  
  DIM NN AS Float  
  DIM N AS Float  
  DIM E1 AS Float  
  DIM MM AS Float  
  DIM E2 AS Float  
  DIM R2 AS Float  
  DIM E3 AS Float  
  DIM E4 AS Float  
  DIM D1 AS Float  
  DIM C1 AS Float  
  DIM D2 AS Float  
  DIM C2 AS Float  
  DIM D3 AS Float  
  DIM C3 AS Float  
  DIM D4 AS Float  
  DIM C4 AS Float
```

```

DIM C AS Float
DIM S AS Float
DIM P AS Float
textareal.Text = ""
N1 = Val(textbox1.Text)
N2 = Val(textbox2.Text)
M1 = Val(textbox3.Text)
M2 = Val(textbox4.Text)
'ERWARTUNGSWERT EN1
R1 = N1 + M1
NN = N1 + N2
N = N1 + N2 + M1 + M2
E1 = R1 * NN / N
textareal.Text = textareal.Text & "Erwartungswert E1 =" & Str(E1) & Chr(13) & Chr(10)
'Rem ERWARTUNGSWERT EN2
MM = M1 + M2
E2 = R1 * MM / N
textareal.Text = textareal.Text & "Erwartungswert E2 =" & Str(E2) & Chr(10) & Chr(13)
'Rem ERWARTUNGSWERT EN3
R2 = N2 + M2
E3 = R2 * NN / N
textareal.Text = textareal.Text & "Erwartungswert E3 =" & Str(E3) & Chr(13) & Chr(10)
'Rem ERWARTUNGSWERT EN4
E4 = R2 * MM / N
textareal.Text = textareal.Text & "Erwartungswert E4 =" & Str(E4) & Chr(13) & Chr(10)
'Rem PROBE=0?
S = E1 + E2 + E3 + E4
P = S - N
textareal.Text = textareal.Text & "Probe" & Str(S) & "-" & Str(N) & "=" & Str(P) & "=0?" & Chr(13)
& Chr(10)
'Rem CHI^2
D1 = (N1 - E1) ^ 2
C1 = D1 / E1
D2 = (M1 - E2) ^ 2
C2 = D2 / E2
D3 = (N2 - E3) ^ 2
C3 = D3 / E3
D4 = (M2 - E4) ^ 2
C4 = D4 / E4
C = C1 + C2 + C3 + C4
textareal.Text = textareal.Text & "CHI-QUADRAT=" & Str(C) & Chr(13) & Chr(10)
IF C - 3.841 > 0 THEN textareal.Text = textareal.Text & "Bei P=0.05 besteht ein signifikanter
Unterschied"
IF C - 3.841 <= 0 THEN textareal.Text = textareal.Text & "Bei P=0.05 besteht kein signifikanter
Unterschied "
END

```

## 4.21.2 Runtest

Wenn man eine Messung aus Sicherheitsgründen mehrfach durchführt, kann man mit dem Runtest überprüfen, ob die erhaltenen einzelnen Messwerte rein zufällig um den wahren Wert streuen oder ob die Messung nichtzufällige Werte liefert. Allgemein kann man mit dem Runtest auch gut Zufallsgeneratoren wie zb einen Würfel oder eine Münze durch mehrfache Wiederholung auf ihre wirkliche Zufälligkeit überprüfen.



Siehe dazu auch: <http://de.wikipedia.org/wiki/Run-Test>

### Überprüfung einer Münze auf Zufälligkeit

Durch mehrfachen Münzwurf wurde folgende Ergebnisreihe gewonnen.

0100100101110111010001001101100101101

Dabei steht die 0 für Wappen und 1 für Zahl der Münze.

**Ein Run ist dann eine Folge von Würfeln mit demselben Ergebnis.**

Folgende Runs sind in der Folge

0  
1  
00  
1  
00  
1  
0  
111  
0  
1  
000  
1  
00  
11  
0  
11  
00  
1  
0  
11  
0  
1

Insgesamt sind es 24 Runs, d.h. Folgen mit demselben Zeichen 0 oder 1.

Wie kann man diese Serie jetzt auf Ihre Zufälligkeit überprüfen ?

Dazu braucht man einige Formeln und Parameter für den Runtest.

- Man braucht die Gesamtzahl der Würfe  $n$ .
- Man braucht die Gesamtzahl der Runs  $R$ .
- Man braucht die Zahl der 1er Würfe  $n_1$ .
- Man braucht die Zahl der 0er Würfe  $n_2$ .
- Man braucht den Erwartungswert  $ER$  für die statistisch erwartete Zahl der Runs.

$$ER = 2 * n_1 * n_2 / (n + 1)$$

- Man braucht die Varianz  $\text{varR}$  und die Wurzel der Varianz.

$$\text{varR} = 2 \cdot n_1 \cdot n_2 \cdot (2 \cdot n_1 \cdot n_2 - n) / (n \cdot n \cdot (n_1 + n_2 - 1))$$

- Man braucht ein Signifikanzniveau  $p = 0,05$
- Man braucht eine Prüfgröße  $z$ . Diese muß zwischen  $-1,96$  und  $+1,96$  liegen, dann kann man von einer zufälligen Folge ausgehen.

$$z = (R - ER) / \text{sqrt}(\text{varR})$$

### **Wie kann man das alles Programmieren ?**

Beispiel: Sie brauchen

- 4 Textareas,
- 13 Befehlsbuttons
- 10 Textboxen

auf Ihrer Form, um das Programm in Gang zu bringen.

Die Bedienung des Programmes ist gewöhnungsbedürftig. Sie verläuft so ziemlich im Uhrzeigersinn. Man kann die Bedienerführung verbessern, wenn man ein paar Errorroutinen mit CATCH einbaut und auf die richtige Bedienung hinweist. Außerdem ist der SETFOCUS Befehl hilfreich, der den nächsten Befehlsknopf markiert, den man drücken muß. Mit ein paar vernünftigen Unterprogrammen und einem Start des Programmes mit einem Knopfdruck wird alles einfacher und weniger fehlerträchtig. Das kommt dann in die verbesserte Version 2.

## Layout

114,3	96,8	-6	0
135,7	104,8	15,4	1
104,8	112,2	-15,5	0
118,5	114,3	-1,8	0
125,7	118,5	5,4	1
121,4	118,9	1,1	1
122,4	120	2,1	1
96,8	120,3	-23,5	0
118,9	121,4	-1,4	0
120	122,4	-0,3	0
112,2	122,8	-8,1	0
127,9	125,7	7,6	1
122,8	127,9	2,5	1
128,9	128,9	8,6	1
120,3	135,7	0	1
Testwerte	Umsortieren	Median abziehen	In 1,0 Folge umwandeln
Testwerte 2	120,3	10 Zahl	7 8 15
	Median		UM 8,46666666667
			U=Runs 6
Bewertung			Varianz
Die Meßwerte sind mit einem $p < 0,05$ zufällig erhoben worden!			3,448888888889
			1,857118436958
			z
Ende			-1,328222593443

Abbildung 46: Bild:runtest1.png

## Der Code

```

PUBLIC SUB Form_Open()
  ME.Text = "Runtest durchführen"
  'Der Run-Test ist ein nichtparametrischer Test
  'auf Stationarität bzw. Nicht-Korrelation
  'IN einer Zeitreihe oder anderen Sequenz.
  'Die abzulehnende Nullhypothese ist hier,
  'dass aufeinanderfolgende Werte unkorreliert sind.
  'Ein run oder "Lauf" ist definiert als eine Folge
  'von gleichen Symbolen IN einer Symbolsequenz
  'mit endlichem Alphabet.
  'Aus einer Symbolsequenz muss zunächst eine numerische Reihe
  'erzeugt werden
  'Zunächst wird der Median der Reihe berechnet
  'und von allen Werten abgezogen.
  'IN der resultierenden Reihe werden negative Zahlen
  'durch ein '-' und positive durch ein '+' ersetzt.
  'Für Stichprobenumfänge n1,n2 > 10 ist
  '(mit n1,n2 gleich der Anzahl der "plus" und der "minus")
  'die Wahrscheinlichkeitsverteilung der runs ungefähr normal
  'mit dem Mittelwert UM = 2 * n1*n2 / (n1+n2) + 1
  'und Varianz

```

```

'varianz = 2*n1*n2*(2*n1*n2 - n1 - n2) / (n1 + n2) / (n1 + n2) / (n1 + n2 - 1)
'Die Testgröße z berechnet sich nun nach:
'z = ( U - UM ) / Wurzel aus Varianz
'wobei U die gemessene Anzahl der "runs" von '-'
'und '+' in der Reihe ist.
'z ist ungefähr standardnormalverteilt.
'Beispiel
'13 3 14 14 1 14 3 8 14 17 9 14 13 2 16 1 3 12 13 14
' 13
' 3
' 14
' 14
' 1
' 14
' 3
' 8
' 14
' 17
' 9
' 14
' 13
' 2
' 16
' 1
' 3
' 12
' 13
' 14
'Median 13    Abzug von der Orginalreihe
'0 -10 1 1 -12 1 -10 -5 1 4 -4 1 0 -11 3 -12 -10 -1 0 1
'Umwandlung in + und -
'+ - + + - + - - + + - + + - + - - - + +
'1 2 3  4 5 6  7  8 9  101112  13
'U ist die Zahl der Runs
'U = 13 runs bzw U: 13
'n1 = Pluszeichen '+': 11
'n2 = Minuszeichen '-': 9
'UM = (2*11*9)/(11+9) + 1 = 10,9
'varianz '= ... = 4,6 ????'
'σU = 2,1
'z = (13 - 10,9) / 2,1 = 1.0
'Entscheidung: Die Hypothese wird nicht abgelehnt.
'Die Elemente der Stichprobe sind vermutlich zufällig entnommen worden.
'Bei einem Signifikanzniveau von 0,05 wird H0 abgelehnt, wenn |z| > 1,96. Dies ist nicht der Fall.
'Entscheidung: Die Hypothese wird nicht abgelehnt. Die Elemente der Stichprobe sind vermutlich
zufällig
'entnommen worden.
'that most of the z-scores for run lengths 1, 2, and 3
'have an absolute value greater than 1.96.
'This IS strong evidence that these data are
'IN fact NOT random.
'Siehe auch: http://www.itl.nist.gov/div898/handbook/eda/section3/eda35d.htm
'http://www.wu-wien.ac.at/usr/stat2/melichar/spsspul/spssk3-9.htm
'Bradley, (1968). Distribution-Free Statistical Tests, Chapter 12.
'
END
liste AS String[]
PUBLIC SUB Button1_Click()
    DIM text AS String
    DIM median AS Float
    DIM liste AS String[]
    DIM posten AS String

```

```

text = Textarea2.Text
liste = Split(text,Chr(10))
'umsortieren
'Länge bestimmen und richtige Formel verwenden,
'je nachdem ob die Länge gerade oder ungerade ist.
IF liste.Length MOD 2 THEN
PRINT liste.Length MOD 2 & " Ungerade"
PRINT (liste.length + 1)/2
PRINT liste[(liste.length + 1)/2 - 1]
median = Val(liste[(liste.length + 1)/2 - 1])
'Der Array beginnt mit dem Element 0 nicht mit 1 !
ELSE
PRINT liste.Length MOD 2 & " Gerade"
median = (Val(liste[list.length/2 - 1]) + Val(liste[list.length/2]))/2
'Der Array beginnt mit dem Element 0 nicht mit 1 !
ENDIF
textbox1.Text = Str(median)
Button4.setfocus
CATCH
IF textareal.Text = "" THEN message.info("Sie müssen im Textfeld 1 eine Werteliste eingeben")
IF textarea2.Text = "" THEN message.info("Sie müssen erst umsortieren drücken")
RETURN
END
PUBLIC SUB Button2_Click()
'umsortieren
c AS Integer
j AS Integer
n AS Integer
y AS Variant
DIM Liste AS String[]
element AS String
txt AS String
text AS String
text = Textareal.Text
liste = Split(text,Chr(10))
y = 0
n = liste.length
REPEAT
c = 0
FOR j = 0 TO n - 2
'PRINT j,y,liste[0],ar[1],ar[2],ar[3],ar[4]
IF Val(liste[j]) > Val(liste[j + 1]) THEN
y = Val(liste[j])
liste[j] = liste[j + 1]
liste[j + 1] = Str(y)
c = 1
ENDIF
NEXT
UNTIL c = 0
FOR EACH element IN liste
txt = txt & Str(element) & Chr(10)
NEXT
PRINT txt
textarea2.Text = ""
txt = Mid$(txt,1,-1)
'Der letzte Zeilenumbruch muß weg
textarea2.Text = txt
Button1.setfocus
END
PUBLIC SUB Button3_Click()
textareal.Text = "114,3"
textareal.Text = textareal.Text & Chr(10) & "135,7"

```

```
textareal.Text = textareal.Text & Chr(10) & "104,8"  
textareal.Text = textareal.Text & Chr(10) & "118,5"  
textareal.Text = textareal.Text & Chr(10) & "125,7"  
textareal.Text = textareal.Text & Chr(10) & "121,4"  
textareal.Text = textareal.Text & Chr(10) & "122,4"  
textareal.Text = textareal.Text & Chr(10) & "96,8"  
textareal.Text = textareal.Text & Chr(10) & "118,9"  
textareal.Text = textareal.Text & Chr(10) & "120"  
textareal.Text = textareal.Text & Chr(10) & "112,2"  
textareal.Text = textareal.Text & Chr(10) & "127,9"  
textareal.Text = textareal.Text & Chr(10) & "122,8"  
textareal.Text = textareal.Text & Chr(10) & "128,9"  
textareal.Text = textareal.Text & Chr(10) & "120,3"  
'mw = 119,373  
'varianz = 92,6016  
'sigma = 9,62297  
'median  
Button2.setfocus  
END  
PUBLIC SUB Button4_Click()  
'median abziehen  
m AS Float  
d AS Float  
n AS Integer  
y AS Variant  
DIM Liste AS String[]  
element AS String  
txt AS String  
text AS String  
text = Textareal.Text  
m = Val(textbox1.Text)  
liste = Split(text,Chr(10))  
y = 0  
n = liste.length  
FOR EACH element IN liste  
d = Val(element) - m  
'PRINT element,m,d  
txt = txt & Str(d) & Chr(10)  
NEXT  
PRINT txt  
textarea3.Text = ""  
txt = Mid$(txt,1,-1)  
'Der letzte Zeilenumbruch muß weg  
textarea3.Text = txt  
Button6.setfocus  
END  
PUBLIC SUB Button5_Click()  
'in 1,0 Folge umwandeln  
e AS Integer  
DIM Liste AS String[]  
element AS String  
txt AS String  
text AS String  
text = Textarea3.Text  
liste = Split(text,Chr(10))  
'n = liste.length  
FOR EACH element IN liste  
IF Val(element) < 0 THEN  
e = 0  
ELSE  
e = 1  
ENDIF
```

```
txt = txt & Str(e) & Chr(10)
NEXT
PRINT txt
textarea4.Text = ""
txt = Mid$(txt,1,-1)
'Der letzte Zeilenumbruch muß weg
textarea4.Text = txt
END
PUBLIC SUB Button6_Click()
DIM s AS String
DIM n AS Integer
DIM T AS String
DIM char AS String
DIM ozaehler AS Integer
DIM izaehler AS Integer
DIM Liste AS String[]
element AS String
txt AS String
text AS String
text = Textarea4.Text
liste = Split(text,Chr(10))
n = liste.length
ozaehler = 0
izaehler = 0
FOR EACH element IN liste
IF element = "0" THEN ozaehler = ozaehler + 1
IF element = "1" THEN izaehler = izaehler + 1
NEXT
textbox2.Text = ozaehler
textbox3.Text = izaehler
textbox4.Text = ozaehler + izaehler
END
PUBLIC SUB Button11_Click()
ME.Close
END
PUBLIC SUB Button7_Click()
UM AS Float
n1 AS Integer
n2 AS Integer
n1 = Val(textbox2.Text)
n2 = Val(textbox3.Text)
UM = 2 * n1 * n2 / ( n1 + n2 ) + 1
textbox5.Text = Str(UM)
END
PUBLIC SUB Button8_Click()
DIM s AS String
DIM n AS Integer
DIM T AS String
DIM char AS String
charnext AS String
DIM runzaehler AS Integer
DIM Liste AS String[]
element AS String
txt AS String
text AS String
text = Textarea4.Text
liste = Split(text,Chr(10))
n = liste.length
FOR EACH element IN liste
s = s & element
NEXT
runzaehler = 1
```

```
T = Trim(s)
FOR n = 1 TO Len(T) - 1
    char = Mid$(T, n, 1)
    charnext = Mid$(T, n + 1, 1)
    IF char <> charnext THEN runzaehler = runzaehler + 1
NEXT
textbox6.Text = runzaehler
END
PUBLIC SUB Button9_Click()
'varianz = 2*n1*n2*(2*n1*n2 - n1 - n2) / (n1 + n2) / (n1 + n2)
varianz AS Float
n1 AS Integer
n2 AS Integer
zwei AS Integer
summe AS Integer
run AS Integer
n1 = Val(textbox2.Text)
n2 = Val(textbox3.Text)
run = Val(textbox6.text)
zwei = 2 * n1 * n2
summe = n1 + n2
PRINT zwei,summe,zwei * (zwei - summe),
varianz = zwei * (zwei - summe) / (summe * summe * (summe - 1))
'varianz=(run * (run - 2))/(4 * (run - 1))
textbox7.Text = Str(varianz)
IF varianz < 0 THEN varianz = -varianz
textbox8.Text = Str(Sqr(varianz))
END
PUBLIC SUB Button10_Click()
z AS Float
run AS Float
UM AS Float
svar AS Float
run = Val(textbox6.text)
UM = Val(textbox5.text)
svar = Val(textbox8.text)
z = (run - UM)/svar
textbox9.Text = Str(z)
END
PUBLIC SUB Button12_Click()
z AS Float
t1 AS String
t2 AS String
t1 = "Die Meßwerte sind mit einem p <0,05 zufällig erhoben worden!"
t2 = "Die Meßwerte sind mit einem p <0,05 NICHT zufällig erhoben worden!"
z = Val(textbox9.Text)
IF z > 1.96 XOR z < -1.96 THEN
textbox10.text = t2
ELSE
textbox10.text = t1
ENDIF
END
PUBLIC SUB Button13_Click()
'noch ein paar Beispielwerte, sicher nicht zufällig
textareal.Text = "1"
textareal.Text = textareal.Text & Chr(10) & "2"
textareal.Text = textareal.Text & Chr(10) & "3"
textareal.Text = textareal.Text & Chr(10) & "4"
textareal.Text = textareal.Text & Chr(10) & "5"
textareal.Text = textareal.Text & Chr(10) & "6"
textareal.Text = textareal.Text & Chr(10) & "7"
textareal.Text = textareal.Text & Chr(10) & "8"
```



```

textareal.Text = textareal.Text & Chr(10) & "9"
textareal.Text = textareal.Text & Chr(10) & "10"
textareal.Text = textareal.Text & Chr(10) & "11"
textareal.Text = textareal.Text & Chr(10) & "12"
textareal.Text = textareal.Text & Chr(10) & "13"
textareal.Text = textareal.Text & Chr(10) & "14"
textareal.Text = textareal.Text & Chr(10) & "15"
END

```

## Vereinfachungen

- Die Beispiele können mit dem Array Befehl viel schneller eingegeben werden.
- Das Sortieren kann mit dem Array befehl Array.sort stark vereinfacht werden.
- Das Ganze sollte mit einem Knopfdruck funktionieren.
- Die Setfocusbefehle sollten in der richtigen Reihenfolge durchlaufen
- Unterprogramme für die einzelnen Abschnitte wären übersichtlicher.

Das ganze funktioniert weniger fehlerträchtig, wenn alles auf einen Knopfdruck hin passiert. Trotzdem sollte man obige Version stehen lassen, das sie Schritt für Schritt vorgeht.

### 4.21.3 Runtest 2

In etwas einfacherer Form wurde der Runtest noch einmal programmiert:

Man braucht dazu

- 9 Commandbuttons
- 9 Textboxen
- 1 Textarea

2 einfache Beispielfolgen wurden integriert.

Listing:

```

PUBLIC SUB Form_Open()
'Der Run-Test ist ein nichtparametrischer Test
'auf Stationarität bzw. Nicht-Korrelation
'IN einer Zeitreihe oder anderen Sequenz.
'Die abzulehnende Nullhypothese ist hier,
'dass aufeinanderfolgende Werte unkorreliert sind.
'Ein run oder "Lauf" ist definiert als eine Folge

```

```

'von gleichen Symbolen IN einer Symbolsequenz
'mit endlichem Alphabet.
'Aus einer Symbolsequenz muss zunächst eine numerische Reihe
'erzeugt werden
'Zunächst wird der Median der Reihe berechnet
'und von allen Werten abgezogen.
'IN der resultierenden Reihe werden negative Zahlen
'durch ein '-' und positive durch ein '+' ersetzt.
'Für Stichprobenumfänge n1,n2 > 10 ist
'(mit n1,n2 gleich der Anzahl der "plus" und der "minus")
'die Wahrscheinlichkeitsverteilung der runs ungefähr normal
'mit dem Mittelwert UM = 2 * n1*n2 / (n1+n2) + 1
'und Varianz
'varianz = 2*n1*n2*(2*n1*n2 - n1 - n2) / (n1 + n2) / (n1 + n2) / (n1 + n2 - 1)
'Die Testgröße z berechnet sich nun nach:
'z = ( U - UM ) / Wurzel aus Varianz
'wobei U die gemessene Anzahl der "runs" von '-'
'und '+' in der Reihe ist.
'z ist ungefähr standardnormalverteilt.
'Beispiel
'13 3 14 14 1 14 3 8 14 17 9 14 13 2 16 1 3 12 13 14
'Median 13      Abzug von der Originalreihe
'0 -10 1 1 -12 1 -10 -5 1 4 -4 1 0 -11 3 -12 -10 -1 0 1
'Umwandlung in + und -
'+ - + + - + - - + + - + - + - - - + +
'1 2 3  4 5 6  7  8 9  101112  13
'U ist die Zahl der Runs
'U = 13 runs bzw U: 13
'n1 = Pluszeichen '+': 11
'n2 = Minuszeichen '-': 9
'UM = (2*11*9)/(11+9) + 1 = 10,9
'varianz '= ... = 4,6 ????'
'σU = 2,1
'z = (13 - 10,9) / 2,1 = 1.0
'Entscheidung: Die Hypothese wird nicht abgelehnt.
'Die Elemente der Stichprobe sind vermutlich zufällig entnommen worden.
'that most of the z-scores for run lengths 1, 2, and 3
'have an absolute value greater than 1.96.
'This IS strong evidence that these data are
'IN fact NOT random.
'Siehe auch: http://www.itl.nist.gov/div898/handbook/eda/section3/eda35d.htm
'http://www.wu-wien.ac.at/usr/stat2/melichar/spsspul/spssk3-9.htm
'Bradley, (1968). Distribution-Free Statistical Tests, Chapter 12.
',
END

PUBLIC SUB Button1_Click()
ME.CLOSE
END

PUBLIC SUB Button2_Click()
DIM s AS String
DIM n AS Integer
DIM T AS String
DIM char AS String
DIM ozaehler AS Integer
DIM izaehler AS Integer
s = textareal.Text
ozaehler = 0
izaehler = 0
T = Trim(s)
FOR n = 1 TO Len(T)

```

```

        char = Mid$(T, n, 1)
        SELECT CASE TRUE
            CASE char = "0"
                ozaehler = ozaehler + 1
            CASE char = "1"
                izaehler = izaehler + 1
        END SELECT
    NEXT
    textbox1.Text = ozaehler
    textbox2.Text = izaehler
    textbox3.Text = ozaehler + izaehler
END

PUBLIC SUB Button3_Click()
'mehrfacher Muenzwurf zur Erzeugung einer Zufallsfolge von 0 und 1
DIM a AS Integer
DIM t AS String
DIM M AS Integer
Randomize()
FOR a = 1 TO 200
    M = Int(2 * Rnd)
    ' Erzeugt einen Wurf zwischen 0 und 1
    t = t & LTrim$(Str$(M))
NEXT
Textareal.Text = t
END

PUBLIC SUB Button4_Click()
UM AS Float
n1 AS Integer
n2 AS Integer
n1 = Val(textbox1.Text)
n2 = Val(textbox2.Text)
UM = 2 * n1 * n2 / (n1 + n2) + 1
'UM = 2 * n1*n2 / (n1+n2) + 1
'Mittelwert der Wahrscheinlichkeitsverteilung der runs
textbox4.Text = Str(UM)
END

PUBLIC SUB Button6_Click()
'varianz = 2*n1*n2*(2*n1*n2 - n1 - n2) / (n1 + n2) / (n1 + n2)
'Varianz der Wahrscheinlichkeitsverteilung der runs
varianz AS Float
n1 AS Integer
n2 AS Integer
zwei AS Integer
summe AS Integer
run AS Integer
n1 = Val(textbox1.Text)
n2 = Val(textbox2.Text)
run = Val(textbox5.text)
zwei = 2 * n1 * n2
summe = n1 + n2
PRINT zwei, summe, zwei * (zwei - summe),
varianz = zwei * (zwei - summe) / (summe * summe * (summe - 1))
'varianz=(run * (run - 2))/(4 * (run - 1))
textbox6.Text = Str(varianz)
IF varianz < 0 THEN varianz = -varianz
textbox7.Text = Str(Sqr(varianz))
END

PUBLIC SUB Button7_Click()
'+ - + + - - - + + - + + - - - + +

```

```
Textareal.Text = "10110100110110100011"
END

PUBLIC SUB Button5_Click()
DIM s AS String
DIM n AS Integer
DIM T AS String
DIM char AS String
charnext AS String
DIM runzaehler AS Integer
s = textareal.Text
runzaehler = 1
  T = Trim(s)
  FOR n = 1 TO Len(T) - 1
    char = Mid$(T, n, 1)
    charnext = Mid$(T, n + 1, 1)
    IF char <> charnext THEN runzaehler = runzaehler + 1
  NEXT
textbox5.Text = runzaehler
END

PUBLIC SUB Button8_Click()
z AS Float
run AS Float
UM AS Float
svar AS Float
run = Val(textbox5.text)
UM = Val(textbox4.text)
svar = Val(textbox7.text)
z = ( run - UM)/svar
'z ist die Prüfvariable
textbox8.Text = Str(z)
END

PUBLIC SUB Button9_Click()
z AS Float
t1 AS String
t2 AS String
t1 = "Die Meßwerte sind mit einem p <0,05 zufällig erhoben worden!"
t2 = "Die Meßwerte sind mit einem p <0,05 NICHT zufällig erhoben worden!"
z = Val(textbox8.Text)
IF z > 1.96 XOR z < -1.96 THEN
textbox9.text = t2
ELSE
textbox9.text = t1
ENDIF
END
```

### **Abbildung des Programmablaufes für die 01 Folge Chaitin A**

Abbildung 47: Bild:Runtest2.png

#### 4.21.4 Runtest 3

Das Programm wurde im Vergleich zu Runtest 2 noch einmal einfacher programmiert. Der Test läuft jetzt auf einen Click. 3 Beispiele wurden integriert.

Man braucht dazu

- 5 Commandbuttons
- 9 Textboxen
- 1 Textarea

3 einfache Beispielfolgen wurden integriert.

**Code:**

```

PUBLIC SUB Form_Open()
ME.Text = "Runttest von Münzwürfen"
END
PUBLIC SUB Button1_Click()
s AS String
'10Folge aus der Textarea
n AS Integer
'Laufvariable
T AS String
'getrimmte 10 Folge aus s
char AS String
'einzelne Position der 10 Folge
oz AS Integer
'Zahl der Nullen
iz AS Integer
'Zahl der Einser
UM AS Float
'Mittelwert der Wahrscheinlichkeitsverteilung der runs
runs AS Integer
'Zahl der runs
charnext AS String
'nächste Position in der Folge
varianz AS Float
'Varianz der Wahrscheinlichkeitsverteilung der runs
zwei AS Integer
summe AS Integer
'Hilfsvariablen zur Berechnung der Varianz
svar AS Float
'Wurzel der Varianz
z AS Float
'Prüfziffer
t1 AS String
t2 AS String
'Bewertungstexte
'-----0 und 1, UM Berechnung-----
s = textareal.Text
oz = 0
iz = 0
T = Trim(s)
  FOR n = 1 TO Len(T)
    char = Mid$(T, n, 1)
    SELECT CASE TRUE
      CASE char = "0"
        oz = oz + 1
      CASE char = "1"
        iz = iz + 1
    END SELECT
  NEXT
textbox1.Text = oz
textbox2.Text = iz
textbox3.Text = oz + iz
UM = 2 * oz * iz / ( oz + iz ) + 1
'Mittelwert der Wahrscheinlichkeitsverteilung der runs
textbox4.Text = Str(UM)
'---Runs-----
runs = 1
  FOR n = 1 TO Len(T) - 1
    char = Mid$(T, n, 1)
    charnext = Mid$(T, n + 1, 1)
    IF char <> charnext THEN runs = runs + 1
  NEXT

```

```

NEXT
textbox5.Text = runs
'-----Varianz-----
'varianz = 2*n1*n2*(2*n1*n2 - n1 - n2) / (n1 + n2) / (n1 + n2)
'Varianz der Wahrscheinlichkeitsverteilung der runs
zwei = 2 * oz * iz
summe = oz + iz
'PRINT zwei,summe,zwei * (zwei - summe),
varianz = zwei * (zwei - summe) / (summe * summe * (summe - 1))
textbox6.Text = Str(varianz)
IF varianz < 0 THEN varianz = -varianz
textbox7.Text = Str(Sqr(varianz))
svar = Sqr(varianz)
z = ( runs - UM)/svar
'z ist die Prüfvariable
textbox8.Text = Str(z)
'-----Bewertung-----
t1 = "Die Meßwerte sind mit einem p <0,05 zufällig erhoben worden!"
t2 = "Die Meßwerte sind mit einem p <0,05 NICHT zufällig erhoben worden!"
IF z > 1.96 XOR z < -1.96 THEN
textbox9.text = t2
ELSE
textbox9.text = t1
ENDIF
END
PUBLIC SUB Button2_Click()
'+ - + + - + - - + + - + - + - - - + +
Textareal.Text = "10110100110110100011"
END
PUBLIC SUB Button3_Click()
'mehrfacher Muenzwurf zur Erzeugung einer Zufallsfolge von 0 und 1
DIM a AS Integer
DIM t AS String
DIM M AS Integer
Randomize()
FOR a = 1 TO 200
M = Int(2 * Rnd)
' Erzeugt einen Wurf zwischen 0 und 1
t = t & LTrim$(Str$(M))
NEXT
Textareal.Text = t
END
PUBLIC SUB Button4_Click()
ME.CLOSE
END
PUBLIC SUB Button5_Click()
Textareal.Text = "10101010101010101010"
END

```

## 4.22 Entropie bei einfachen 01 Folgen

Die Entropie ist ein Begriff der viel gebraucht aber wenig verstanden wird. In der Statistik wird der Begriff noch wenig genutzt, obwohl er eigentlich ein grundlegender Begriff sein könnte.

- Siehe [http://de.wikipedia.org/wiki/Entropie\\_\(Informationstheorie\)](http://de.wikipedia.org/wiki/Entropie_(Informationstheorie))

- Siehe <http://www.madeasy.de/2/entropie.htm>
- Siehe <http://www.madeasy.de/2/zufallgz.htm>

Im folgenden werden einige einfache Gambasprogramme aufgelistet und beschrieben, die die Entropieberechnung einfacher 01 Folgen schrittweise erarbeiten sollen.

### 4.22.1 Vorübung1

Im folgenden Programm werden systematisch alle Möglichkeiten einer zunehmend längeren 01 Folge aufgelistet. Sie brauchen nur eine leere Form, um das Programm in Gang zu bekommen. Die Ausgabe erfolgt im Direktfenster.

```
PUBLIC SUB Form_Open()  
'Dieses Programm zaehlt im Binaersystem von 1 bis 2^8  
'Die vorderen Leerräume werden mit Null aufgefüllt.  
s AS Integer  
z AS Integer  
t AS String  
FOR s = 1 TO 8  
  FOR z = 0 TO (2^s - 1)  
    t = Bin$(z,s)  
    PRINT t  
  NEXT  
NEXT  
END
```

**Ausgabe:**

```
0  
1  
00  
01  
10  
11  
000  
001  
010  
011  
100  
101  
110  
111  
0000  
0001  
0010  
0011  
0100
```



## 4.22.2 Vorübung2

In einer zweiten Vorübung werden jetzt allen Binärzahlen Entropiewerte zugeordnet, die ganz grob vereinfacht ( und damit noch ziemlich falsch) berechnet werden. Eine ganz geordnete 01 Folge hat dabei immer den Entropiewert 0. Eine völlig zufällige 01 Folge hat einen Entropiewert, der ihrer Länge entspricht.

Beispiel:

- 00000000000000000000 Entropie = 0
- 01101100110111100010 Entropie = 20

Vorsicht die berechneten Entropiewerte sind noch ziemlich falsch !!

```
' Gambas class file
t AS String
PUBLIC SUB Form_Open()
s AS Integer
z AS Integer
n AS Integer
e AS Integer
n = 0
FOR s = 1 TO 10
FOR z = 0 TO (2^s - 1)
t = Bin$(z,s)
PRINT "n,t " & n,t;
e = ent(t)
PRINT " E = " & e
n = n + 1
NEXT
NEXT
END
PUBLIC FUNCTION ent(e AS Integer) AS Integer
oz AS Integer
iz AS Integer
gz AS Integer
char AS String
te AS String
n AS Integer
oz = 0
iz = 0
gz = 0
te = Trim(t)
FOR n = 1 TO Len(te)
char = Mid$(te, n, 1)
SELECT CASE TRUE
CASE char = "0"
oz = oz + 1
CASE char = "1"
iz = iz + 1
END SELECT
gz = oz + iz
NEXT
PRINT " gz,oz,iz " & gz,oz,iz;
IF oz > iz THEN e = gz - oz
' IF oz > 0.6*iz THEN e = gz - oz Else runstest
IF oz < iz THEN e = oz
```

```
'IF oz < 0.4*iz THEN e = oz Else runstest
IF oz = iz THEN e = gz
'IF oz = iz THEN runstest
RETURN e
END
```

### 4.22.3 10Statistik

Man kann an das Problem der Unterscheidung zufälliger und nichtzufälliger 01 Folgen auch noch anders herangehen. Eine Möglichkeit zeigt das folgende Programm 01Statistik.

Das Programm kann an an Hand der Beispiele, die in ihm enthalten sind, leicht ausgetestet werden. Man kann jede beliebige 01 Folge in die Textarea kopieren und auswerten lassen. Es werden nur Runs bis maximal einer Länge von 10 berücksichtigt.

Sie brauchen

- viele Befehlsbuttons
- viele Textboxen
- eine Textarea

um das Programm in Gang zu bringen.

Siehe auch: <http://www.madeasy.de/2/prg01st.htm> ( Noch in VB , Layout blieb gleich)

```
' Gambas class file
t AS String
PUBLIC SUB Button9_Click()
  ME.Close
END
PUBLIC SUB Button2_Click()
'diese Funktion soll eine 01 Datei untersuchen
'Sie soll die Zahl der 00 Paare zurueckgeben
'io,oi,oo Zahl der anderen Paare
DIM s AS String
DIM n AS Integer
DIM T AS String
DIM char AS String
DIM oozaehler AS Integer
DIM oizaehler AS Integer
DIM iozaehler AS Integer
DIM iizaehler AS Integer
s = textareal.Text
oozaehler = 0
oizaehler = 0
iozaehler = 0
iizaehler = 0
T = Trim(s)
FOR n = 1 TO Len(T) STEP 2
```

```

char = Mid$(T, n, 2)
SELECT CASE TRUE
CASE char = "00"
oozaehler = oozaehler + 1
CASE char = "01"
oizaehler = oizaehler + 1
CASE char = "10"
iozaehler = iozaehler + 1
CASE char = "11"
iizaehler = iizaehler + 1
END SELECT
NEXT
textbox4.Text = oozaehler
textbox5.Text = oizaehler
textbox6.Text = iozaehler
textbox7.Text = iizaehler
END
PUBLIC SUB Button1_Click()
'Wieviel 1 und 0 finden sich in der Folge ?
DIM s AS String
DIM n AS Integer
DIM T AS String
DIM char AS String
DIM ozaehler AS Integer
DIM izaehler AS Integer
s = textareal.Text
ozaehler = 0
izaehler = 0
T = Trim(s)
FOR n = 1 TO Len(T)
char = Mid$(T, n, 1)
SELECT CASE TRUE
CASE char = "0"
ozaehler = ozaehler + 1
CASE char = "1"
izaehler = izaehler + 1
END SELECT
NEXT
textbox1.Text = ozaehler
textbox2.Text = izaehler
textbox3.Text = ozaehler + izaehler
END
PUBLIC SUB Button6_Click()
'mehrfacher Muenzwurf zur Erzeugung einer Zufallsfolge von 0 und 1
DIM a AS Integer
DIM t AS String
DIM M AS Integer
Randomize()
FOR a = 1 TO 20000
M = Int(2 * Rnd)
' Erzeugt einen Wurf zwischen 0 und 1
t = t & LTrim$(Str$(M))
NEXT
Textareal.Text = t
END
PUBLIC SUB Button8_Click()
M AS String
t AS String
a AS Integer
FOR a = 1 TO 10000
M = "10"
t = t & LTrim$(Str$(M))

```

```
NEXT
Textareal.Text = t
END
PUBLIC SUB Button4_Click()
DIM s AS String
DIM n AS Integer
DIM T AS String
DIM Tempo AS String
DIM char AS String
a AS Integer
DIM o1 AS Integer
DIM o2 AS Integer
DIM o3 AS Integer
DIM o4 AS Integer
DIM o5 AS Integer
DIM o6 AS Integer
DIM o7 AS Integer
DIM o8 AS Integer
DIM o9 AS Integer
DIM oa AS Integer
s = textareal.Text
    T = Trim(s)
' 'Rem diese Funktion soll eine 01 Datei untersuchen
' 'Rem Sie soll die Zahl der 000,0000,00000 etc ( der runs ) zurueckgeben
o1 = 0
o2 = 0
o3 = 0
o4 = 0
o5 = 0
o6 = 0
o7 = 0
o8 = 0
o9 = 0
oa = 0
' 'Rem 00000000001000000000010000000000
' Cls
Tempo = RTrim$(LTrim$(s))
' 'Rem a = 10
a = 10
' 'Rem PRINT Mid$(Tempo, 1, 11)
IF Mid$(Tempo, 1, a + 1) = "00000000001" THEN oa = oa + 1
' 'Rem PRINT Mid$(Tempo, Len(Temp) - 10, 11)
IF Len(Temp) < a THEN GOTO a9
IF Len(Temp) = a AND s = "0000000000" THEN oa = oa + 1
IF Len(Temp) = a THEN GOTO a9
IF Mid$(Tempo, Len(Temp) - 10, 11) = "10000000000" THEN oa = oa + 1
FOR n = 1 TO Len(Temp)
IF Mid$(Tempo, n, a + 2) = "100000000001" THEN oa = oa + 1
'Rem PRINT Mid$(Tempo, n, a)
NEXT
'
a9:
' 'Rem a = 9
a = 9
'Rem 000000000100000000010000000000
' 'Rem PRINT Mid$(Tempo, 1, 10)
IF Mid$(Tempo, 1, 10) = "0000000001" THEN o9 = o9 + 1
'Rem PRINT Mid$(Tempo, Len(Temp) - 9, 10)
IF Len(Temp) < a THEN GOTO a8
IF Len(Temp) = a AND s = "0000000000" THEN o9 = o9 + 1
IF Len(Temp) = a THEN GOTO a8
IF Mid$(Tempo, Len(Temp) - 9, 10) = "1000000000" THEN o9 = o9 + 1
```

```
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "10000000001" THEN o9 = o9 + 1
' 'Rem PRINT Mid$(Tempo, n, a)
NEXT
'
a8:
' 'Rem a = 8
a = 8
' 'Rem 00000000100000000100000000
' 'Rem PRINT Mid$(Tempo, 1, a + 1)
IF Mid$(Tempo, 1, a + 1) = "000000001" THEN o8 = o8 + 1
' 'Rem PRINT Mid$(Tempo, Len(Tempo) - a, a + 1)
IF Len(Tempo) < a THEN GOTO a7
IF Len(Tempo) = a AND s = "00000000" THEN o8 = o8 + 1
IF Len(Tempo) = a THEN GOTO a7
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "1000000001" THEN o8 = o8 + 1
' 'Rem PRINT Mid$(Tempo, n, a)
NEXT
'
a7:
' 'Rem a = 7
a = 7
' 'Rem 00000000100000000100000000
IF Mid$(Tempo, 1, a + 1) = "00000001" THEN o7 = o7 + 1
IF Len(Tempo) < a THEN GOTO a6
IF Len(Tempo) = a AND s = "0000000" THEN o7 = o7 + 1
IF Len(Tempo) = a THEN GOTO a6
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "10000000" THEN o7 = o7 + 1
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "100000001" THEN o7 = o7 + 1
NEXT
'
a6:
' 'Rem a = 6
a = 6
' 'Rem 00000010000001000000
IF Mid$(Tempo, 1, a + 1) = "0000001" THEN o6 = o6 + 1
IF Len(Tempo) < a THEN GOTO a5
IF Len(Tempo) = a AND s = "000000" THEN o6 = o6 + 1
IF Len(Tempo) = a THEN GOTO a5
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "1000000" THEN o6 = o6 + 1
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "10000001" THEN o6 = o6 + 1
NEXT
'
a5:
' 'Rem a = 5
a = 5
' 'Rem 000001000001000000
IF Mid$(Tempo, 1, a + 1) = "000001" THEN o5 = o5 + 1
IF Len(Tempo) < a THEN GOTO a4
IF Len(Tempo) = a AND s = "00000" THEN o5 = o5 + 1
IF Len(Tempo) = a THEN GOTO a4
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "100000" THEN o5 = o5 + 1
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "1000001" THEN o5 = o5 + 1
NEXT
'
a4:
' 'Rem a = 4
a = 4
```

```
' 'Rem 00001000010000
IF Mid$(Tempo, 1, a + 1) = "00001" THEN o4 = o4 + 1
IF Len(Tempo) < a THEN GOTO a3
IF Len(Tempo) = a AND s = "0000" THEN o4 = o4 + 1
IF Len(Tempo) = a THEN GOTO a3
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "10000" THEN o4 = o4 + 1
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "100001" THEN o4 = o4 + 1
NEXT
a3:
a = 3
' 'Rem 00010001000
IF Mid$(Tempo, 1, a + 1) = "0001" THEN o3 = o3 + 1
IF Len(Tempo) < a THEN GOTO a2
IF Len(Tempo) = a AND s = "000" THEN o3 = o3 + 1
IF Len(Tempo) = a THEN GOTO a2
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "1000" THEN o3 = o3 + 1
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "10001" THEN o3 = o3 + 1
NEXT
'
a2:
a = 2
' 'Rem 00100100
IF Mid$(Tempo, 1, a + 1) = "001" THEN o2 = o2 + 1
IF Len(Tempo) < a THEN GOTO aeins
IF Len(Tempo) = a AND s = "00" THEN o2 = o2 + 1
IF Len(Tempo) = a THEN GOTO aeins
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "100" THEN o2 = o2 + 1
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "1001" THEN o2 = o2 + 1
NEXT
aeins:
a = 1
' 'Rem 01010
IF Mid$(Tempo, 1, a + 1) = "01" THEN o1 = o1 + 1
IF Len(Tempo) < a THEN GOTO a0
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "10" THEN o1 = o1 + 1
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "101" THEN o1 = o1 + 1
NEXT
'
a0:
textbox8.Text = o1
textbox9.Text = o2
textbox10.Text = o3
textbox11.Text = o4
textbox12.Text = o5
textbox13.Text = o6
textbox14.Text = o7
textbox15.Text = o8
textbox16.Text = o9
textbox17.Text = oa
END
PUBLIC SUB Button7_Click()
'muss noch erstellt werden
END
PUBLIC SUB Button5_Click()
'diese Funktion soll eine 01 Datei
'untersuchen
'Sie soll die Zahl der 11,111,1111,11111 Runs etc zurueckgeben
DIM s AS String
```

```

DIM n AS Integer
DIM T AS String
DIM Tempo AS String
DIM char AS String
a AS Integer
DIM i1 AS Integer
DIM i2 AS Integer
DIM i3 AS Integer
DIM i4 AS Integer
DIM i5 AS Integer
DIM i6 AS Integer
DIM i7 AS Integer
DIM i8 AS Integer
DIM i9 AS Integer
DIM ia AS Integer
s = textareal.Text
T = Trim(s)
i1 = 0
i2 = 0
i3 = 0
i4 = 0
i5 = 0
i6 = 0
i7 = 0
i8 = 0
i9 = 0
ia = 0
Tempo = RTrim$(LTrim$(s))
' Rem 111111111011111111101111111111
' Cls
a = 10
' Rem PRINT Mid$(Tempo, 1, 11)
IF Mid$(Tempo, 1, a + 1) = "1111111110" THEN ia = ia + 1
' Rem PRINT Mid$(Tempo, Len(Temp) - 10, 11)
IF Len(Temp) < a THEN GOTO b9
IF Len(Temp) = a AND s = "111111111" THEN ia = ia + 1
IF Len(Temp) = a THEN GOTO b9
IF Mid$(Tempo, Len(Temp) - a, a + 1) = "0111111111" THEN ia = ia + 1
FOR n = 1 TO Len(Temp)
IF Mid$(Tempo, n, a + 2) = "0111111110" THEN ia = ia + 1
' Rem PRINT Mid$(Tempo, n, a)
NEXT
b9:
a = 9
IF Mid$(Tempo, 1, 10) = "1111111110" THEN i9 = i9 + 1
IF Len(Temp) < a THEN GOTO b8
IF Len(Temp) = a AND s = "111111111" THEN i9 = i9 + 1
IF Len(Temp) = a THEN GOTO b8
IF Mid$(Tempo, Len(Temp) - 9, 10) = "011111111" THEN i9 = i9 + 1
FOR n = 1 TO Len(Temp)
IF Mid$(Tempo, n, a + 2) = "0111111110" THEN i9 = i9 + 1
NEXT
b8:
a = 8
IF Mid$(Tempo, 1, a + 1) = "111111110" THEN i8 = i8 + 1
IF Len(Temp) < a THEN GOTO b7
IF Len(Temp) = a AND s = "11111111" THEN i8 = i8 + 1
IF Len(Temp) = a THEN GOTO b7
IF Mid$(Tempo, Len(Temp) - a, a + 1) = "011111111" THEN i8 = i8 + 1
FOR n = 1 TO Len(Temp)
IF Mid$(Tempo, n, a + 2) = "0111111110" THEN i8 = i8 + 1
NEXT

```

```
,  
b7:  
a = 7  
IF Mid$(Tempo, 1, a + 1) = "11111110" THEN i7 = i7 + 1  
IF Len(Tempo) < a THEN GOTO b6  
IF Len(Tempo) = a AND s = "1111111" THEN i7 = i7 + 1  
IF Len(Tempo) = a THEN GOTO b6  
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "01111111" THEN i7 = i7 + 1  
FOR n = 1 TO Len(Tempo)  
IF Mid$(Tempo, n, a + 2) = "011111110" THEN i7 = i7 + 1  
NEXT  
,  
b6:  
a = 6  
IF Mid$(Tempo, 1, a + 1) = "1111110" THEN i6 = i6 + 1  
IF Len(Tempo) < a THEN GOTO b5  
IF Len(Tempo) = a AND s = "111111" THEN i6 = i6 + 1  
IF Len(Tempo) = a THEN GOTO b5  
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "0111111" THEN i6 = i6 + 1  
FOR n = 1 TO Len(Tempo)  
IF Mid$(Tempo, n, a + 2) = "01111110" THEN i6 = i6 + 1  
NEXT  
,  
b5:  
a = 5  
,  
IF Mid$(Tempo, 1, a + 1) = "111110" THEN i5 = i5 + 1  
IF Len(Tempo) < a THEN GOTO b4  
IF Len(Tempo) = a AND s = "11111" THEN i5 = i5 + 1  
IF Len(Tempo) = a THEN GOTO b4  
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "011111" THEN i5 = i5 + 1  
FOR n = 1 TO Len(Tempo)  
IF Mid$(Tempo, n, a + 2) = "0111110" THEN i5 = i5 + 1  
NEXT  
b4:  
a = 4  
IF Mid$(Tempo, 1, a + 1) = "11110" THEN i4 = i4 + 1  
IF Len(Tempo) < a THEN GOTO b3  
IF Len(Tempo) = a AND s = "1111" THEN i4 = i4 + 1  
IF Len(Tempo) = a THEN GOTO b3  
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "01111" THEN i4 = i4 + 1  
FOR n = 1 TO Len(Tempo)  
IF Mid$(Tempo, n, a + 2) = "011110" THEN i4 = i4 + 1  
NEXT  
,  
b3:  
a = 3  
IF Mid$(Tempo, 1, a + 1) = "1110" THEN i3 = i3 + 1  
IF Len(Tempo) < a THEN GOTO b2  
IF Len(Tempo) = a AND s = "111" THEN i3 = i3 + 1  
IF Len(Tempo) = a THEN GOTO b2  
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "0111" THEN i3 = i3 + 1  
FOR n = 1 TO Len(Tempo)  
IF Mid$(Tempo, n, a + 2) = "01110" THEN i3 = i3 + 1  
NEXT  
,  
b2:  
a = 2  
IF Mid$(Tempo, 1, a + 1) = "110" THEN i2 = i2 + 1  
IF Len(Tempo) < a THEN GOTO beins  
IF Len(Tempo) = a AND s = "11" THEN i2 = i2 + 1  
IF Len(Tempo) = a THEN GOTO beins
```



```

IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "011" THEN i2 = i2 + 1
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "0110" THEN i2 = i2 + 1
NEXT
beins:
a = 1
IF Mid$(Tempo, 1, a + 1) = "10" THEN i1 = i1 + 1
IF Len(Tempo) < a THEN GOTO b0
IF Mid$(Tempo, Len(Tempo) - a, a + 1) = "01" THEN i1 = i1 + 1
FOR n = 1 TO Len(Tempo)
IF Mid$(Tempo, n, a + 2) = "010" THEN i1 = i1 + 1
NEXT
'
b0:
textbox18.Text = i1
textbox19.Text = i2
textbox20.Text = i3
textbox21.Text = i4
textbox22.Text = i5
textbox23.Text = i6
textbox24.Text = i7
textbox25.Text = i8
textbox26.Text = i9
textbox27.Text = ia
END
PUBLIC SUB Button10_Click()
'Chaitin , ein bekannter Zufallsforscher gab IN einem Aufsatz folgende Sequenz an
'Lit chaitin G.J. 1975 scientific American 232 S 47 -52
t = "01101100110111100010"
textareal.Text = t
END
PUBLIC SUB Button11_Click()
'Chaitin , ein bekannter Zufallsforscher gab IN einem Aufsatz folgende Sequenz an
'01101100110111100010
'Diese Folge wurde veraendert ,
'so dass nach jeder Ziffer ihr Gegenteil folgt,
t = "0110100110100101101001101010100101011001"
textareal.Text = t
'Rem Wie man zb an Hand der ungleichen Paarverteilung gut erkennen kann , wird der Zufall weniger
'Rem Obwohl auf den ersten Blick ohne Kenntnis der Entstehung
'Rem vielleicht sogar mehr Zufall zu erwarten ist
END
PUBLIC SUB Button12_Click()
'Zufallszahl die durch mehrfachen echten Muenzwurf zur Erzeugung einer Zufallsfolge von 0 und 1
'ermittelt wurde.
t = "1011011010101001110010110011100000011110010100001111010100010011011110110000100010101000111011
100101011101111111000001001101000011011101111010101100000100011101100010000010011111000001111010010
0011011110010101000001011010000110001101000110011110111100011011100100110000001111100100000011000010
0000011010101000001100010110000111001111001000011011111110010010101001111100100010010000100100100
001000101001110011110110000010100111111001011110111011000111011010110000011101100111101011001110"
textareal.Text = t
END
PUBLIC SUB Button13_Click()
'BB ,die originalfolge von Chaitin wird einfach 2 * hintereinander geschrieben
t = "0110110011011110001001101100110111100010"
textareal.Text = t
END
PUBLIC SUB Button14_Click()
'Dieser Befehl zaehlt im Binaersystem von 1 bis 2000
'er listet die Zahlen ohne Zwischenraum hintereinander
'012345678... 0.1.10.11.100.101.110 etc
te AS String

```

```
z AS Integer
FOR z = 0 TO 2000
  t = Bin$(z)
  te = te & t
NEXT
textareal.Text = te
END
```

Das Programm ist ein bißchen lang geraten. Es kann mit Sicherheit verbessert werden. Es hat die Limitation, daß es Runs nur bis zu einer Länge von 10 erfasst.

#### 4.22.4 Vorübung3: Programmierung des Problems mit dem Runtest

*Zitat: Wenn du es nicht programmiert hast, dann hast du es nicht verstanden.*

Leider funktioniert der Runtest nicht bei beliebigen 01 Sequenzen , sondern nur wenn die Zahl von 0 und 1 ungefähr gleich ist. Beispielsweise kann man eine Sequenz 00000000000000000010 im Runtest nicht auf ihre Zufälligkeit überprüfen.

Im Unterschied zur Vorübung2 werden jetzt für die Binärzahlen, bei denen die Zahl der Nullen gleich der Zahl der Einsen ist, die Entropien mit dem Runstest berechnet.

- Die Länge der 01 Folge ist variierbar , hier  $s = 8$ . ( Länge ist die Variable  $s$  im Listing)
- Die Zahl der Nullen und Einsen ist zur Vereinfachung gleich groß.
- Das Programm startet von alleine.
- Die Ergebnisausgabe erfolgt im Direktfenster.
- Bewertung:
  - pz Werte um die 0 sind ein Zeichen hoher Entropie. Pz-Werte über 1,5 oder unter -1,5 sind ein Zeichen niedriger Entropie

```
PUBLIC SUB Form_Open()
s AS Integer
'Laenge der Binaerzahl
z AS Integer
'Zaehler von 0 bis 2^s-1
zz AS Integer
'Zaehler von 1 bis alle Varianten mit ozaehler = izaehler
t AS String
'binaerzahl
ozaehler AS Integer
'Zahl der Nullen
izaehler AS Integer
'Zahl der Einsen
```

```

tt AS String
'binaerzahl als String
n AS Integer
'Laenge der Binaerzahl
char AS String
UM AS Float
varianz AS Float
svar AS Float
'Quadratwurzel der Varianz
zwei AS Integer
summe AS Integer
run AS Integer
nn AS Integer
chari AS String
charnext AS String
runzaehler AS Integer
pz AS Float
'Pruefvariable entspricht dem Entropiewert der Folge
'pz Werte um die 0 sind ein Zeichen hoher Entropie.
'pz-Werte > 1,5 oder < -1,5 sind ein Zeichen niedriger Entropie
s = 10
'Laenge der 01 Folge
zz = 0
FOR z = 0 TO (2^s - 1)
  t = Bin$(z,s)
  tt = Str(t)
  'PRINT "tt = " & tt
  ozaehler = 0
  izaehler = 0
  FOR n = 1 TO Len(tt)
    char = Mid$(tt, n, 1)
    SELECT CASE TRUE
      CASE char = "0"
        ozaehler = ozaehler + 1
      CASE char = "1"
        izaehler = izaehler + 1
    END SELECT
  NEXT
  'PRINT izaehler
  'PRINT ozaehler
  IF izaehler = ozaehler THEN
    zz = zz + 1
    t = tt
    PRINT "zz = " & zz & " t = " & t;
    'runtest
    UM = 2 * s/2 * s/2 / ( s/2 + s/2 ) + 1
    'PRINT "UM = " & UM
    zwei = 2 * s/2 * s/2
    summe = s
    varianz = zwei * ( zwei - summe) / (summe * summe * (summe - 1))
    IF varianz < 0 THEN varianz = -varianz
    'PRINT "Varianz = " & varianz
    runzaehler = 1
    FOR nn = 1 TO Len(t) - 1
      chari = Mid$(t, nn, 1)
      charnext = Mid$(t, nn + 1, 1)
      IF chari <> charnext THEN runzaehler = runzaehler + 1
    NEXT
    'PRINT " runzaehler = " & runzaehler;
    svar = Sqr(varianz)
    pz = ( runzaehler - UM)/svar
    PRINT " pz = " & Str(pz)

```

```
'PRINT Str(pz)
ENDIF
NEXT
END
```

**Ergebnisausgabe:**

```
zz = 1 t = 00001111 pz = -2,291287847478
zz = 2 t = 00010111 pz = -0,763762615826
zz = 3 t = 00011011 pz = -0,763762615826
zz = 4 t = 00011101 pz = -0,763762615826
zz = 5 t = 00011110 pz = -1,527525231652
zz = 6 t = 00100111 pz = -0,763762615826
zz = 7 t = 00101011 pz = 0,763762615826
zz = 8 t = 00101101 pz = 0,763762615826
zz = 9 t = 00101110 pz = 0
zz = 10 t = 00110011 pz = -0,763762615826
zz = 11 t = 00110101 pz = 0,763762615826
zz = 12 t = 00110110 pz = 0
zz = 13 t = 00111001 pz = -0,763762615826
zz = 14 t = 00111010 pz = 0
zz = 15 t = 00111100 pz = -1,527525231652
zz = 16 t = 01000111 pz = -0,763762615826
zz = 17 t = 01001011 pz = 0,763762615826
zz = 18 t = 01001101 pz = 0,763762615826
zz = 19 t = 01001110 pz = 0
zz = 20 t = 01010011 pz = 0,763762615826
zz = 21 t = 01010101 pz = 2,291287847478
zz = 22 t = 01010110 pz = 1,527525231652
zz = 23 t = 01011001 pz = 0,763762615826
zz = 24 t = 01011010 pz = 1,527525231652
zz = 25 t = 01011100 pz = 0
zz = 26 t = 01100011 pz = -0,763762615826
zz = 27 t = 01100101 pz = 0,763762615826
zz = 28 t = 01100110 pz = 0
zz = 29 t = 01101001 pz = 0,763762615826
zz = 30 t = 01101010 pz = 1,527525231652
zz = 31 t = 01101100 pz = 0
zz = 32 t = 01110001 pz = -0,763762615826
zz = 33 t = 01110010 pz = 0
zz = 34 t = 01110100 pz = 0
zz = 35 t = 01111000 pz = -1,527525231652
zz = 36 t = 10000111 pz = -1,527525231652
zz = 37 t = 10001011 pz = 0
zz = 38 t = 10001101 pz = 0
zz = 39 t = 10001110 pz = -0,763762615826
zz = 40 t = 10010011 pz = 0
zz = 41 t = 10010101 pz = 1,527525231652
zz = 42 t = 10010110 pz = 0,763762615826
zz = 43 t = 10011001 pz = 0
zz = 44 t = 10011010 pz = 0,763762615826
zz = 45 t = 10011100 pz = -0,763762615826
zz = 46 t = 10100011 pz = 0
zz = 47 t = 10100101 pz = 1,527525231652
zz = 48 t = 10100110 pz = 0,763762615826
zz = 49 t = 10101001 pz = 1,527525231652
zz = 50 t = 10101010 pz = 2,291287847478
zz = 51 t = 10101100 pz = 0,763762615826
zz = 52 t = 10110001 pz = 0
zz = 53 t = 10110010 pz = 0,763762615826
zz = 54 t = 10110100 pz = 0,763762615826
zz = 55 t = 10111000 pz = -0,763762615826
```



## 4.23 Zufallsberechnungen mit der Randomfunktion

Für die Berechnung von Zufallszahlen ( bzw Pseudozufallszahlen) brauchen Sie die Befehle Randomize(), RND und INT

Siehe auch <http://www.madeasy.de/7/prgmini.htm#z>

### 4.23.1 Münzwurf

### 4.23.2 Münze ganz oft, 01 Zufallsfolge

Wenn Sie eine längere Zufallsfolge aus 0 und 1 haben wollen, können Sie dies mit folgendem Programm erreichen. Wie mit einer Münze, die man immer wieder wirft, wird eine Folge Zufallszahlen zwischen 0 und 1 erzeugt. Sie brauchen eine Textarea und einen Befehlsbutton um das Programm in Gang zu bringen. Der Quelltext hinter dem Befehlsfeld lautet:

```
PUBLIC SUB Button1_Click()
'mehrfacher Muenzwurf zur Erzeugung einer Zufallsfolge von 0 und 1
DIM a AS Integer
DIM t AS String
DIM M AS Integer
Randomize()
FOR a = 1 TO 20000
  M = Int(2 * Rnd)
  ' Erzeugt einen Wurf zwischen 0 und 1
  t = t & LTrim$(Str$(M))
NEXT
Textareal1.Text = t
END
```

### 4.23.3 Würfeln bis 6

Diese Programm erzeugt wie ein Würfel Zufallszahlen zwischen 1 und 6. Das Ergebnis wird in einer Textbox ausgegeben. Sie brauchen einen Commandbutton, ein Textlabel und einen Textbox, um das Programm in Gang zu bringen.

Siehe auch: [Bild](#)

Der Quelltext hinter dem Befehlsfeld Würfeln lautet:

```
PUBLIC SUB Button1_Click()
DIM Wuerfel AS Integer
Randomize()
Wuerfel = Int(Rnd(1,7))
' Erzeugt einen Wurf zwischen 1 und 6
TextLabel1.Text = "Sie würfelten eine "
```

```

TextBox1.Text = Wuerfel
END

```

Folgende Befehle wurden verwendet:

### **Randomize: Initialisiert den Zufallszahlengenerator.**

Anmerkungen: Das Zahl-Argument kann ein beliebiger zulässiger numerischer Ausdruck sein. Durch Zahl wird der Zufallszahlengenerator initialisiert, indem ihm ein neuer Startwert zugewiesen wird. Wenn Sie Zahl weglassen, wird als neuer Startwert der von der Timer-Funktion ausgegebene Wert verwendet. Somit ist der Befehl Randomize Timer eigentlich redundant.

Wird Randomize nicht verwendet, dann liefert die Rnd-Funktion bei jeder Ausführung des Programms dieselbe Zufallszahlenfolge. Soll sich die Zufallszahlenfolge bei jeder Ausführung des Programms ändern, dann setzen Sie eine Randomize-Anweisung ohne Argument an den Anfang des Programms.

### **Rnd Liefert eine Zufallszahl.**

```
Rnd ( [ Min [ , Max ] )
```

Berechnet eine Pseudozufallszahl mit dem Lehmer Algorithmus.

Wenn keine Parameter angegeben werden, dann wird eine Zufallszahl im Intervall [ 0 , 1 [ zwischen Null und Eins erzeugt.

Wenn der erste Parameter ( = Min ) angegeben wird, dann wird eine Zufallszahl im Intervall [ 0 , Min [ zwischen Null und dem Parameter erzeugt.

Wenn man die beiden Parameter angibt, dann wird eine Zufallszahl im Intervall [ Min , Max [ erzeugt.

Beispiel:

```
PRINT Rnd
=> 0.019539254718
```

```
PRINT Rnd(2)
=> 0.040205506608
```

```
PRINT Rnd(Pi, Pi(2))
=> 3.204108046818
```

### **Int**

Value = Int ( Zahl )

Schneidet die Stellen nach dem Komma weg und gibt nur die Zahlen vor dem Komma aus. Anders erklärt: Gibt den Zahlenwert vor dem Komma zurück, d.h es wird die ganze Zahl ausgegeben, die kleiner als ursprüngliche Zahl ist .

**Beispiel:**

```
PRINT Int (Pi)
```

=> 3

```
PRINT Int (-Pi)
```

=> -4

#### 4.23.4 6er Würfel überprüfen

Im folgenden wird ein Programm aufgelistet , welches die Randomfunktion an Hand eines Würfels überprüft. Es funktioniert direkt im Terminalmodus von Gambas. Man braucht also keine Form.

```
' Gambas class file
STATIC PUBLIC SUB Main()
DIM x AS Integer
DIM w AS Integer
DIM a AS Integer
DIM b AS Integer
DIM c AS Integer
DIM d AS Integer
DIM e AS Integer
DIM f AS Integer
Randomize ()
FOR x = 1 TO 1000000
'w = Int (Rnd(6) + 1) identisch zur nächsten Zeile
w = Int (Rnd(1,7))
  SELECT CASE w
    CASE 1
      a = a + 1
    CASE 2
      b = b + 1
    CASE 3
      c = c + 1
    CASE 4
      d = d + 1
    CASE 5
      e = e + 1
    CASE 6
      f = f + 1
  END SELECT
NEXT
PRINT a, b, c, d, e, f
END
```



### 4.23.5 Würfeln bis 12

Man braucht eine Textbox , ein Textlabel und einen Commandbutton, um das Programm zu starten. Das Problem ist analog dem 6er Würfel einfach zu lösen :

```
PUBLIC SUB Button1_Click()
Wuerfel AS Integer
Randomize()
Wuerfel = Int(Rnd(1,13)) ' Erzeugt einen Wurf zwischen 1 und 12
TextLabel1.Text = "Sie würfelten eine "
TextBox1.Text = Wuerfel
END
```

Im Terminalmodus ist es kürzer:

```
STATIC PUBLIC SUB Main()
Wuerfel AS Integer
Randomize()
Wuerfel = Int(Rnd(1,13)) ' Erzeugt einen Wurf zwischen 1 und 12
PRINT wuerfel
END
```

Oder noch kürzer:

```
STATIC PUBLIC SUB Main()
Randomize()
PRINT Int(Rnd(1,13))
END
```

### 4.23.6 Lotto

Dieses schon etwas kompliziertere Programm zieht aus den Zahlen 1 bis 49 zufällig sechs Lottozahlen und eine Zusatzzahl. Das Ganze wird geordnet ausgegeben und Doppelgänger werden vermieden.

Bevor man ein Zufallsprogramm akzeptiert, sollte man es 1000 mal laufen lassen und die gleichberechtigte Verteilung und Unberechenbarkeit der Zahlenfolge mit dem Computer überprüfen.

Sie brauchen 2 Befehlsbuttons und eine Textarea um das Programm in Gang zu bringen.

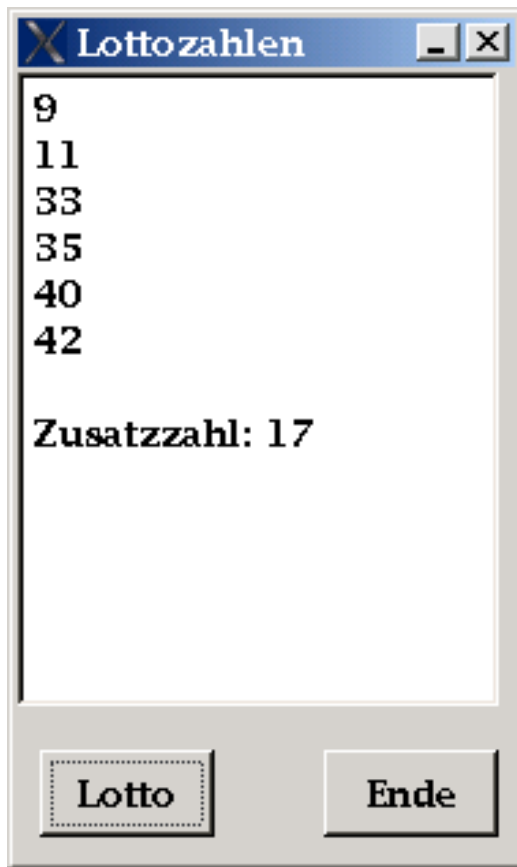


Abbildung 49: Screenshot zum Programm Lottozahlen

Der Quelltext hinter dem Befehlsfeld Lottozahlen lautet:

```
' Gambas class file
ar AS Integer[]
PUBLIC SUB Form_Open()
    ' schon beim Öffnen der Form wird der Array mit lauter Nullen gefüllt.
    x AS Integer
    ar = NEW Integer[]
    ME.Text = "Lottozahlen"
    FOR x = 0 TO 48
        ar.Add(Str(0))
    NEXT
END

PUBLIC SUB Button1_Click()
    x AS Integer
    r AS Integer
    element AS Integer
    txt AS String
    z AS Integer
```

```

txt = ""
'der Array wird komplett auf Null gestellt
FOR x = 0 TO 48
  ar[x] = 0
NEXT
'dann wird der Array mit 6 Zufallszahlen an den Stellen gefüllt, die mit der Zufallszahl
übereinstimmen
Randomize()
FOR x = 0 TO 5
  r = Int(Rnd(1,49))
  IF r = ar[r] THEN x = x - 1
  'Falls ein Doppelgänger auftritt gehe eins zurück
  ar[r] = r
NEXT
'nur die Zahlen ungleich 0 (<> 0 ) werden ausgegeben
FOR EACH element IN ar
  IF element <> 0 THEN txt = txt & element & Chr(10)
NEXT
'Zusatzzahl suchen , die nicht schon gezogen wurde
REPEAT
  z = Int(Rnd(1,49))
UNTIL z <> ar[z]
'Textausgabe wird zusammengestellt
txt = txt & Chr(10) & "Zusatzzahl: " & z
textareal.Text = txt
END

PUBLIC SUB Button2_Click()
  ME.Close
END

```

### 4.23.7 Zufallsbuchstaben

Diese Programm erzeugt Zufallsbuchstaben zwischen A und Z. Das Ergebnis wird in einer Textarea ausgegeben.

Sie brauchen eine Textarea und 2 Commandbuttons, um das Programm in Gang zu bringen. Es schaut dann in etwas so aus:

Der Quelltext hinter dem Befehlsfeld Zufallsbuchstaben lautet:

```

PUBLIC SUB Button1_Click()
DIM Z AS Integer
DIM Buchstabe AS String
Randomize()
Textareal.Text = ""
FOR Z = 1 TO 50
  Buchstabe = Chr$(Int(Rnd(65,91)))
  TextAreal.Text = TextAreal.Text & Buchstabe
NEXT
END

```

Mit dem Befehl CHR\$ werden Zahlen zu Buchstaben umgewandelt.

Character = Chr\$ ( Zahl ) gibt das Zeichen zurück, welches dem ASCII code von Zahl entspricht .

**Beispiel 1:**

```
PRINT Chr$(65)
```

Ergebnis: A

**Beispiel 2:**

```
PUBLIC SUB Button1_Click()  
DIM x AS Integer  
FOR x = 30 TO 133  
    PRINT Chr$(x);  
NEXT  
END
```

### 4.23.8 Zufallspunkte

Mit dem folgenden Programm können Sie ein Zeichenfeld ( DrawingArea ) mit Zufallspunkten füllen.

Sie brauchen für das Programm:

- eine Drawingarea
- zwei Commandbuttons.

Das Ganze schaut dann so aus:

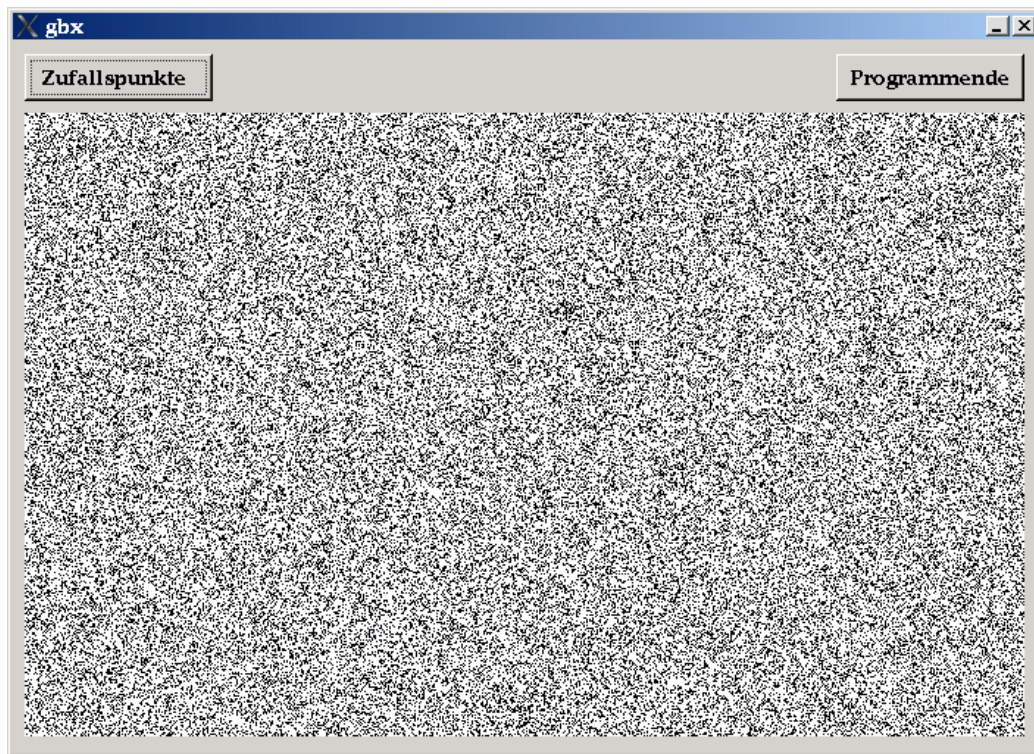


Abbildung 50: Bild:Prgzufallspunkte.png

Der Code dazu:

```
PUBLIC SUB Button1_Click()  
x AS Integer  
y AS Integer  
z AS Integer  
Draw.Begin(DrawingArea1)  
FOR z = 1 TO 100000  
    x = Int(Rnd(0,DrawingArea1.Width))  
    y = Int(Rnd(0,DrawingArea1.Height))  
    Draw.Point(x,y)  
NEXT  
Draw.End  
END  
PUBLIC SUB Button2_Click()  
ME.Close  
END
```

## 4.23.9 Zufallsquadrate

Confetti Programm

Mit dem folgenden Confettiprogramm wird ein Zeichnungsfeld mit vielen kleinen, verschieden farbigen Quadraten gefüllt. In dem Programm taucht 3 mal der Zufall auf: 1. Die Farbe ist per Zufall gewählt. 2. Die x Koordinate der Quadrate ist per Zufall gewählt. 3. Die y Koordinate der Quadrate ist per Zufall gewählt.

Man braucht eine Form, darauf plziert man eine Drawing Area. Dann braucht man noch 2 Commandbuttons.

Das Ganze schaut so aus:

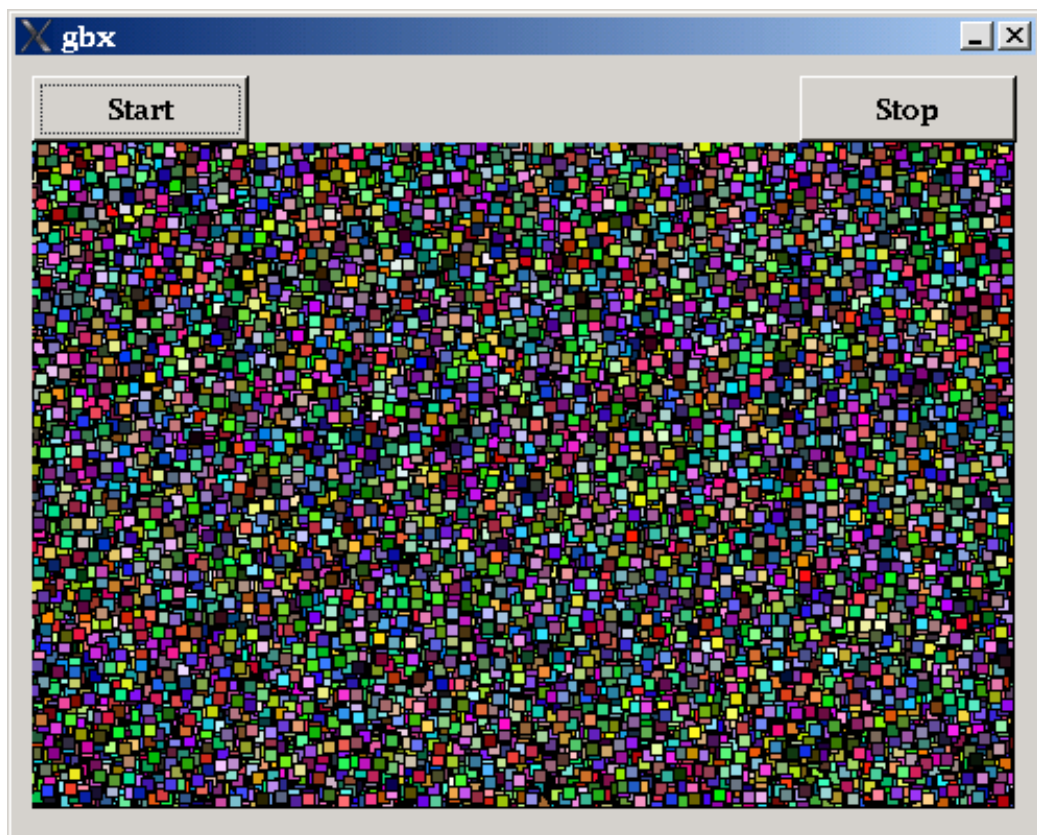


Abbildung 51: Bild:Prgconfetti.png

Der Code dazu:

```
PUBLIC SUB Form_Open()  
  Button1.Text = "Start"  
  Button2.Text = "Stop"  
END  
PUBLIC SUB Button1_Click()  
  i AS Integer  
  X AS Integer  
  Y AS Integer
```

```
Draw.Begin(DrawingArea1)
FOR i = 1 TO 200000
  Draw.FillColor = Int(Rnd(&HFFFFFF&))
  'PRINT Int(Rnd(&HFFFFFF&))
  Draw.FillStyle = Fill.Solid
  X = Int(Rnd(DrawingArea1.Width))
  Y = Int(Rnd(DrawingArea1.Height))
  Draw.Rect(X - 7, Y - 7, 7, 7)
NEXT
Draw.End
END
PUBLIC SUB Button2_Click()
  ME.Close
END
```

### **4.23.10 Münzwurf Statistik**

## **4.24 DrawingArea, Image**

Um unter Gambas Grafiken zu erstellen sollten Sie sich mit dem Steuerelement des Zeichnungsfeldes ( DrawingArea) vertraut machen.

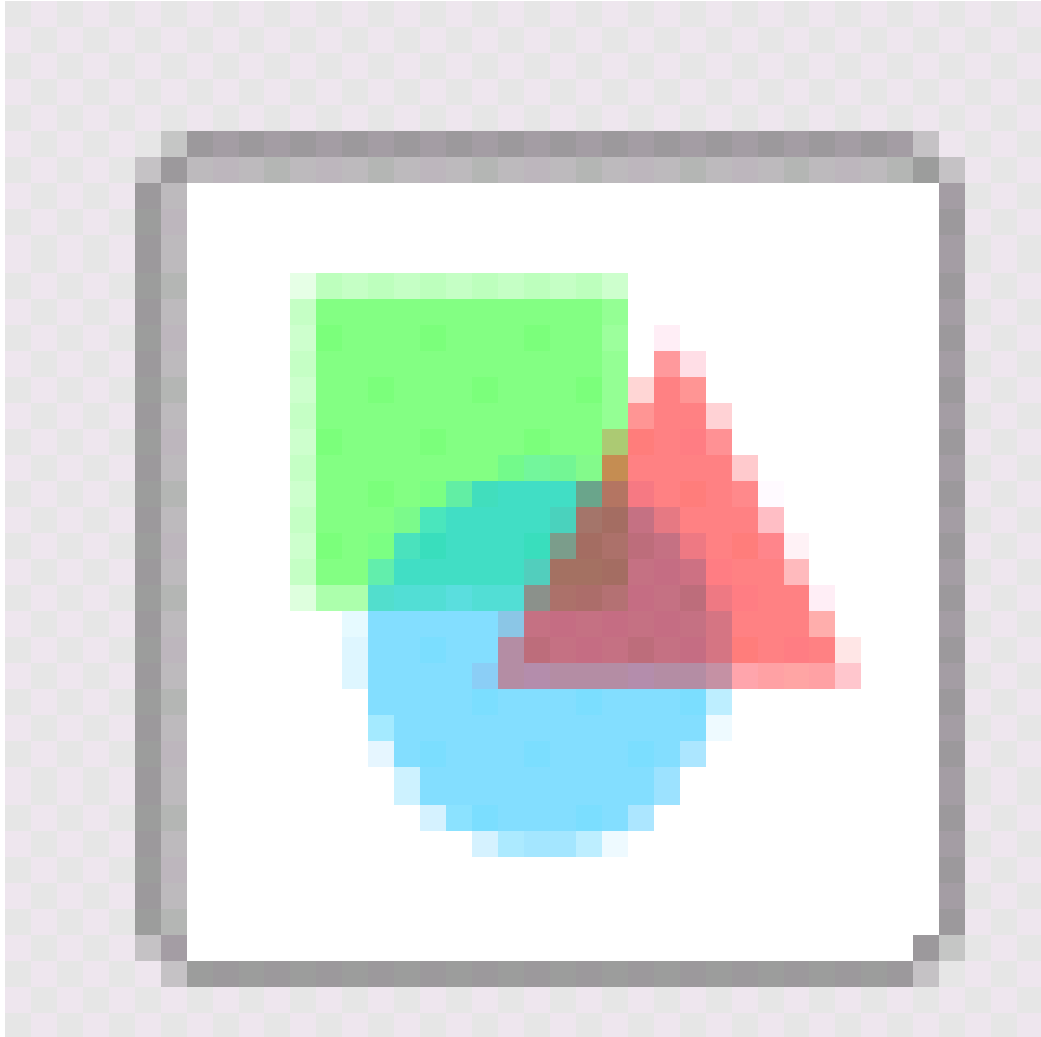


Abbildung 52: Bild:Gambasdrawingareaicon.png

## 4.25 Die Koordinaten der DrawingArea1

Folgendes kleine Programm liefert die Koordinaten der DrawingArea1

Machen Sie ein neues Projekt. Öffnen Sie eine neue Form: Mit der rechten Maus auf Form klicken. Geben Sie der Form irgendeinen Namen oder nehmen Sie den Vorschlag *Form1*. Holen Sie sich eine DrawingArea = Zeichenfeld aus der Werkzeugkiste. Benennen Sie diese nicht um.



Kopieren Sie den Code von unten ins Programm. Starten Sie das Programm mit F5. Die ausgegebenen Zahlen entsprechen den Zahlen bei den Eigenschaften ( = F4 = Properties) der DrawingArea1.

Beachten Sie folgendes : die linke obere Ecke ist (X,Y) , die rechte untere Ecke ist (X + DrawingArea1.Width, Y + DrawingArea1.Height )

Der Code:

```
PUBLIC SUB Form_Open()  
  PRINT DrawingArea1.X  
  PRINT DrawingArea1.Y  
  PRINT DrawingArea1.Width  
  PRINT DrawingArea1.Height  
END
```

Ergebnis

Wenn Sie das Zeichengebiet am Anfang nicht verändert haben werden folgende Werte ausgegeben

```
0  
0  
64  
32
```

## 4.26 Bilder

Siehe [Gambas:\\_Bilder](#)

```
PUBLIC SUB Form_Open()  
  PRINT DrawingArea1.X  
  PRINT DrawingArea1.Y  
  PRINT DrawingArea1.Width  
  PRINT DrawingArea1.Height  
END
```

## 4.27 Farben

Siehe [Gambas:\\_Farben](#)

## 4.28 Mathematische Funktionen

Siehe [Gambas: Rechnen](#)

## 4.29 Punkt, Linie, Kreis

Zeichnen kann man in Gambas nur in einem Grafikfeld (= DrawingArea ). In einem Grafikfeld kann man Punkte, Rechtecke, Ellipsen, Linien und anderes zeichnen. Man kann verschiedene Farben und Stiftbreiten benutzen.

## 4.30 Punkte

Das folgende Beispielprogramm zeichnet ein paar **Punkte**. Einen einzelnen Punkt sieht man schlecht.

Machen Sie dazu ein neues Projekt. Öffnen Sie eine neue Form: Mit der rechten Maus auf Form klicken. Geben Sie ihr den Namen Fmain. Holen Sie sich einen Befehlsknopf = Button aus der Werkzeugkiste . Holen Sie sich eine DrawingArea = Zeichenfeld aus der Werkzeugkiste.

Kopieren Sie den Code ins Programm. Starten Sie das Programm mit F5. Achten Sie auf die Farbgebung bei den Eigenschaften, sonst sehen Sie nichts.

```
PUBLIC SUB Button1_Click()  
Draw.Begin(DrawingArea1)  
Draw.Point(100,100)  
Draw.Point(100,101)  
Draw.Point(100,102)  
Draw.Point(100,103)  
Draw.Point(100,99)  
Draw.Point(100,98)  
Draw.Point(100,97)  
Draw.Point(100,96)  
Draw.End  
END
```

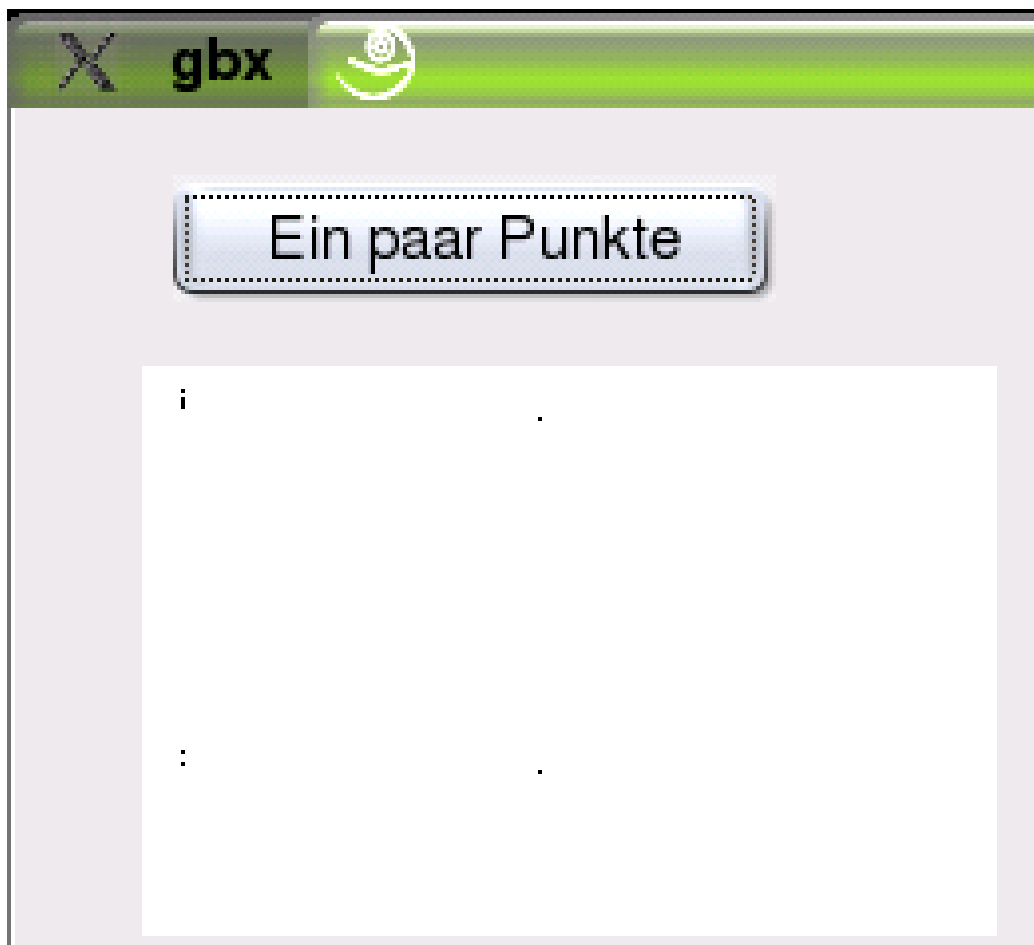


Abbildung 53: Bild:Prgpunkt.png

## 4.31 Linien

Das nächste kleine Programm zeichnet ein paar **Linien**.

Machen Sie ein neues Projekt. Öffnen Sie eine neue Form: Mit der rechten Maus auf Form klicken. Geben Sie ihr den Namen Fmain. Holen Sie sich einen Befehlsknopf = Button aus der Werkzeugkiste . Holen Sie sich eine DrawingArea = Zeichenfeld aus der Werkzeugkiste.

Kopieren Sie den Code ins Programm. Starten Sie das Programm mit F5. Achten Sie auf die Farbgebung bei den Eigenschaften, sonst sehen Sie nichts.

```
PUBLIC SUB Button1_Click()
```

```
DIM B AS Integer ' Variable deklarieren.  
Draw.Begin(DrawingArea1)  
FOR B = 1 TO 200 STEP 10 ' Schleife initialisieren.  
Draw.Line(1, B, 500, B)  
NEXT  
Draw.End  
END
```

### 4.31.1 Linienbreite

Dieses Grafikprogramm zeigt die Programmierung verschieden dicker Linien. Sie brauchen wieder einen Befehlsbutton und eine DrawingArea. Der entscheidende Befehl heißt hier **Draw.LineWidth** Hinter dem Befehl **Linienbreite** steht folgender Code :

```
PUBLIC SUB Button1_Click()  
DIM B AS Integer ' Variable deklarieren.  
Draw.Begin(DrawingArea1)  
Draw.Line(10,100, 20, 100)  
FOR B = 1 TO 100 STEP 10 ' Schleife initialisieren.  
Draw.LineWidth=B ' Stiftbreite einstellen.  
Draw.Line(10+B,100, 20+B, 100)  
NEXT  
Draw.End  
END
```

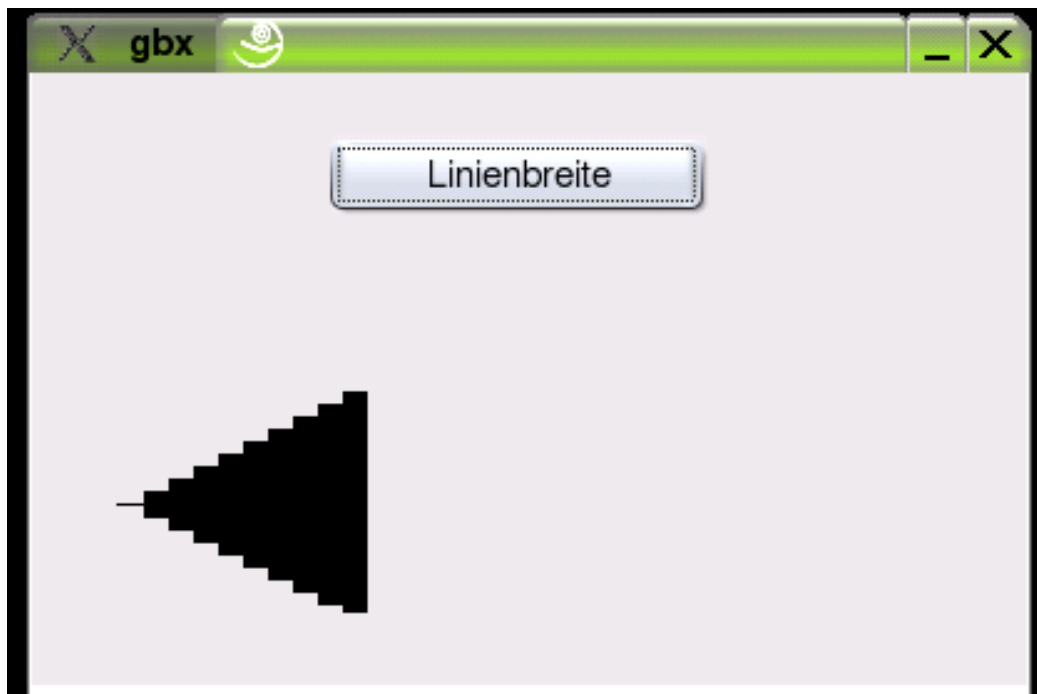


Abbildung 54: Bild:prgbreit.png

## 4.31.2 Noch ein paar Linien

```
PUBLIC SUB Button1_Click()
Draw.Begin(DrawingArea1)
' Zeichnet eine Linie horizontal durch die Mitte der Form
Draw.Line (0, ME.Height / 2, ME.Width, ME.Height / 2)
' Zeichnet eine Linie senkrecht durch die Mitte der Form
Draw.Line (ME.Width / 2, 0,ME.Width / 2, ME.Height)
' Zeichnet eine Linie von der linken oberen zur unteren rechten Ecke
Draw.Line (0, 0,ME.Width, ME.Height)
' Zeichnet eine Linie von der rechten oberen zur unteren linken Ecke
Draw.Line (ME.Width, 0,0, ME.Height)
Draw.End
END
```

Warum werden die Linien nicht sauber zentriert ?

Fügen Sie direkt nach der ersten Zeile folgenden Code ein:

```
DrawingArea1.Resize(ME.Width, ME.Height)
DrawingArea1.Background = &HFFFFFF&
```

Schaut es jetzt besser aus ?

Wenn Sie die Zeile

```
DrawingArea1.Resize(ME.Width, ME.Height)
```

nach dem Befehl `Draw.Begin(DrawingArea1)` einfügen , gibt es eine Fehlermeldung:

```
QPaintDevice: Cannot destroy paint device that is being painted
X Error: BadDrawable (invalid Pixmap or Window parameter) 9
Major opcode: 66
Minor opcode: 0
Resource id: 0x2e00010
X Error: RenderBadPicture (invalid Picture parameter) 180
Major opcode: 153
Minor opcode: 5
Resource id: 0x2e00025
```

## 4.31.3 Farbe der Linie

Wenn Sie die Farbe Ihrer Linie verändern wollen, dann hilft folgendes Programm

```
Draw.Begin(DrawingArea1)
Draw.ForeColor = &HFFFFFF
' weiße Farbe
' Draw.ForeColor = &H0000FF& blaue Linie
' Draw.ForeColor = &HFF00FF& lila
' Draw.ForeColor = &HFF0000& rot
' Draw.ForeColor = &H00FF00& grün
' Draw.ForeColor = &HFFFF00& gelb
' Draw.ForeColor = &H00FFFF& türkis
' &H000000& = Schwarz
Draw.Line(1, 130, 500, 400)
Draw.End
```

## 4.32 Rechteck

Das folgende Programm zeichnet ein **Rechteck**.

Machen Sie ein neues Projekt. Öffnen Sie eine neue Form: Mit der rechten Maus auf Form klicken. Geben Sie ihr den Namen Fmain. Holen Sie sich einen Befehlsknopf = Button aus der Werkzeugkiste . Holen Sie sich eine DrawingArea = Zeichenfeld aus der Werkzeugkiste.

Nennen Sie die Zeichenfläche *da*. Dazu markieren Sie die Zeichenfläche mit der Maus . Drücken Sie F4 und sie erhalten die Eigenschaften = Properties der Zeichenfläche. In der zweiten Zeile bei (Name) geben Sie statt DrawingArea den eigenen Namen da ein.

Kopieren Sie den Code ins Programm. Starten Sie das Programm mit F5. Achten Sie auf die Farbgebung bei den Eigenschaften, sonst sehen Sie nichts.

Beachten Sie folgendes:

Die **Koordinaten der Drawing Area** gehen von der linken Oberen Ecke ( DrawingArea.X = 0, DrawingArea.Y = 0) bis zur unteren rechten Ecke (DrawingArea.Width = beliebig, DrawingArea.Height = beliebig).

Bei den Koordinaten des Rechtecks geben die ersten 2 Ziffern die absoluten Koordinaten der linken oberen Ecke des Rechtecks in der Drawing Area an. Im Beispiel werden 100 und 100 gewählt. Die 2 weiteren Zahlen geben die Koordinaten der rechten unteren Ecke des Rechtecks relativ zur linken oberen Anfangsecke an. Im Beispiel 90 und 90. Die absoluten Koordinaten der rechten unteren Ecke sind im Beispiel dann 190 und 190 .

Beim Draw.Line Befehl ist das anders. Hier wurden absolute Koordinaten gewählt.

Als Übung können Sie versuchen in das Rechteck die 2 Diagonalen mit Drawline einzuzeichnen.

Wie lauten dazu die Koordinaten. Lösung siehe unten.

```
PUBLIC SUB Button1_Click()  
Draw.Begin(da)  
Draw.Rect(100, 100, 90, 90)  
Draw.End  
END
```

### Variante mit Diagonalen

```
PUBLIC SUB Button1_Click()  
Draw.Begin(da)  
Draw.Rect(100, 100, 90, 90)  
Draw.Line(100, 100, 190, 190)
```

```
Draw.Line(100, 190, 190, 100)
Draw.End
END
```

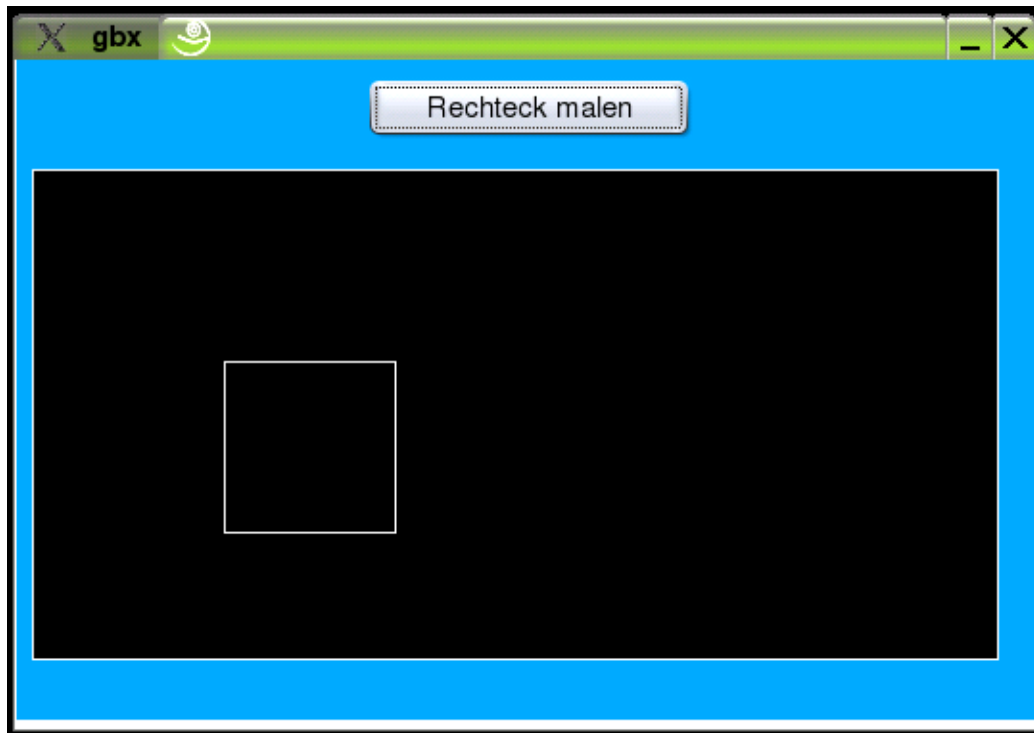


Abbildung 55: Bild:draw.png

### 4.32.1 Rechteck mit Farbe füllen

Das Beispielprogramm zeichnet ein Rechteck und füllt es mit weißer Farbe.

```
PUBLIC SUB Button1_Click()
  Draw.Begin(DrawingArea1) ' Verwende DrawinAreal als "Leinwand"
  Draw.FillColor = Color.white
  Draw.FillStyle = 1
  Draw.ForeColor = Color.white ' der Rand wird auch weiß
  Draw.Rect (100, 100,200,200)
  Draw.End
END
```

## 4.33 Kreis

Dieses Grafikprogramm zeigt die Programmierung einer **Kreisgrafik**:

Hinter dem Befehl *ein paar Kreise* steht der folgende Code:

```
PUBLIC SUB Button1_Click()  
    Draw.Begin(Drawingarea1)  
    Draw.Ellipse(10, 50, 90, 90)  
    Draw.Ellipse(10, 50, 50, 50)  
    Draw.Ellipse(10, 50, 20, 20)  
    Draw.End  
END
```

Sie brauchen eine Zeichenfläche (Klasse Drawingarea) und einen Knopf der Klasse Commandbutton auf Ihrem Formular, um das Programm in Gang zu bringen. Spielen Sie mit den einzelnen Größen etwas herum und beobachten Sie wie sich die Grafiken ändern.

Vereinfachen Sie das Programm so, daß nur noch ein Kreis gezeichnet wird.

Beachten Sie folgendes:

- Die erste Zahl hinter dem Befehl Draw.Ellipse gibt die x Koordinate der linken oberen Ecke eines Rechtecks an, in die die Ellipse , bzw. der Kreis tangential eingebettet ist.

- Man kann diesen Wert xloe ( x Wert linke obere Ecke) nennen

Die zweite Zahl gibt die y Koordinate der linken oberen Ecke an.

- Man kann diesen Wert yloe ( y Wert linke obere Ecke) nennen

Die dritte Zahl gibt die x Koordinate der rechten unteren Ecke an. Der Wert ist kein Absolutwert sondern relativ zur linken oberen Ecke.

- Man kann diesen Wert rxrue ( relativen x Wert rechte untere Ecke) nennen

Die vierte Zahl gibt die y Koordinate der rechten unteren Ecke an.

- Man kann diesen Wert ryrue ( relativen y Wert rechte untere Ecke) nennen

Will man einen Kreis programmieren, dann müssen der dritte und der vierte Parameter gleich sein, denn nur so wird das berührende Rechteck ein Quadrat. Nur ein Quadrat kann mit allen 4 Seiten einen Kreis berühren.

```
Draw.Ellipse(xloe, yloe, rxrue, ryrue)
```

Für den Kreis gilt rxrue = ryrue

Überprüfen Sie, ob im obigen Kreisprogramm diese Vorgaben eingehalten werden. Verändern Sie den Parameter 3 und 4 und schauen Sie sich an , was passiert.

Um das Ganze zu verdeutlichen programmiert man am besten einen Kreis und das umgebende Quadrat.



```
PUBLIC SUB Button1_Click()  
Draw.Begin(Drawingarea1)  
Draw.Ellipse(10, 50, 90, 90)  
Draw.Rect(10, 50, 90, 90)  
Draw.End  
END
```

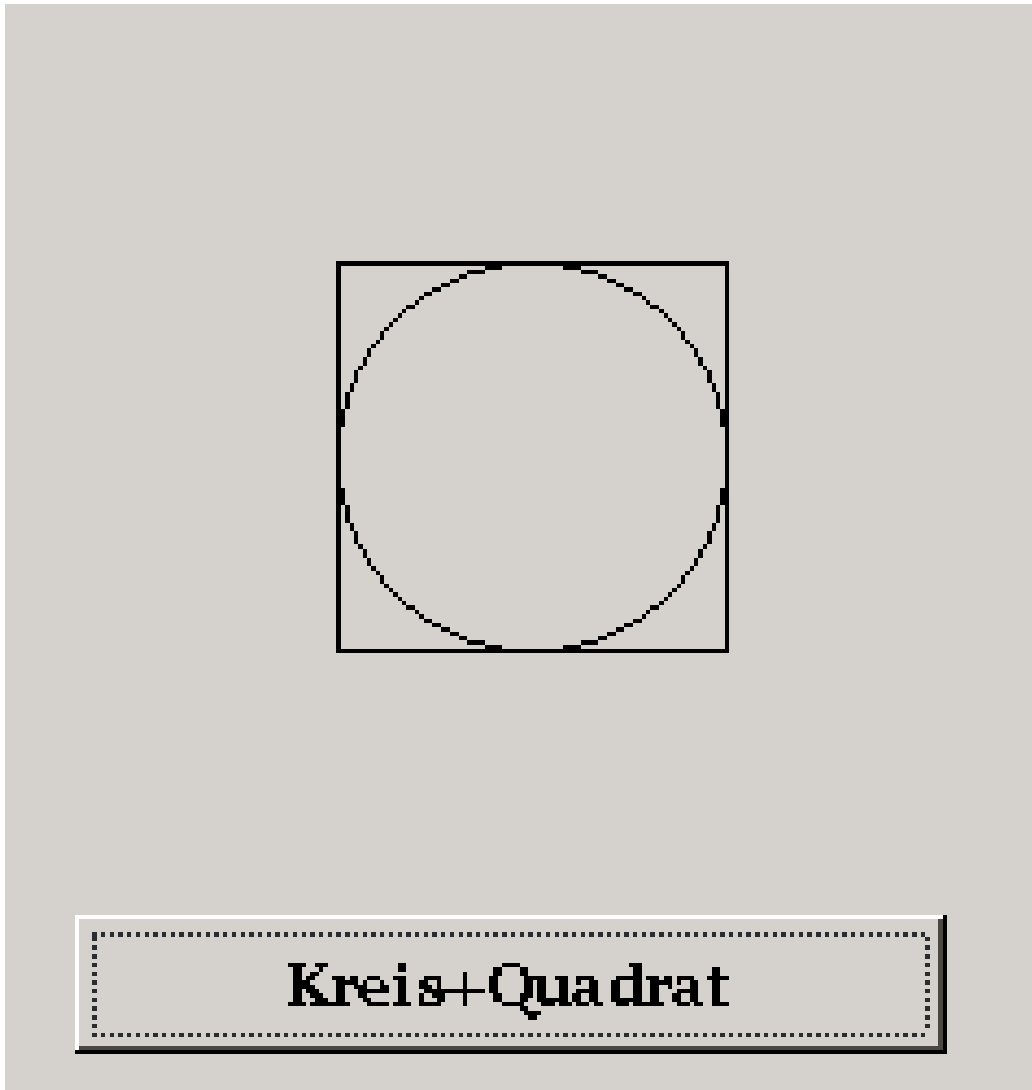


Abbildung 56: Bild:Gambaskreisplusquadrat.png

Aus diesen Vorgaben kann man den Radius des Kreises berechnen:

Radius = (dritte oder vierter Parameter von Draw.Ellipse) / 2)

Aus diesen Vorgaben kann man auch die Koordinaten des Kreismittelpunktes berechnen.

```
mx = xloe + rxrue / 2  
my = yloe + rxrue / 2
```

Wie kann man dann die 2 Brennpunkte einer Ellipse berechnen, wenn also rxrue und ryrue verschieden sind ??

### 4.33.1 Die Röhre

Ein weiteres Beispiel mit vielen verschiedenen Kreisen. Das Programm wurde mit Menusteuerung programmiert. Siehe [Gambas: Menü](#) Außerdem wird eine DrawingArea auf der Form gebraucht.

So schaut das Ganze aus:

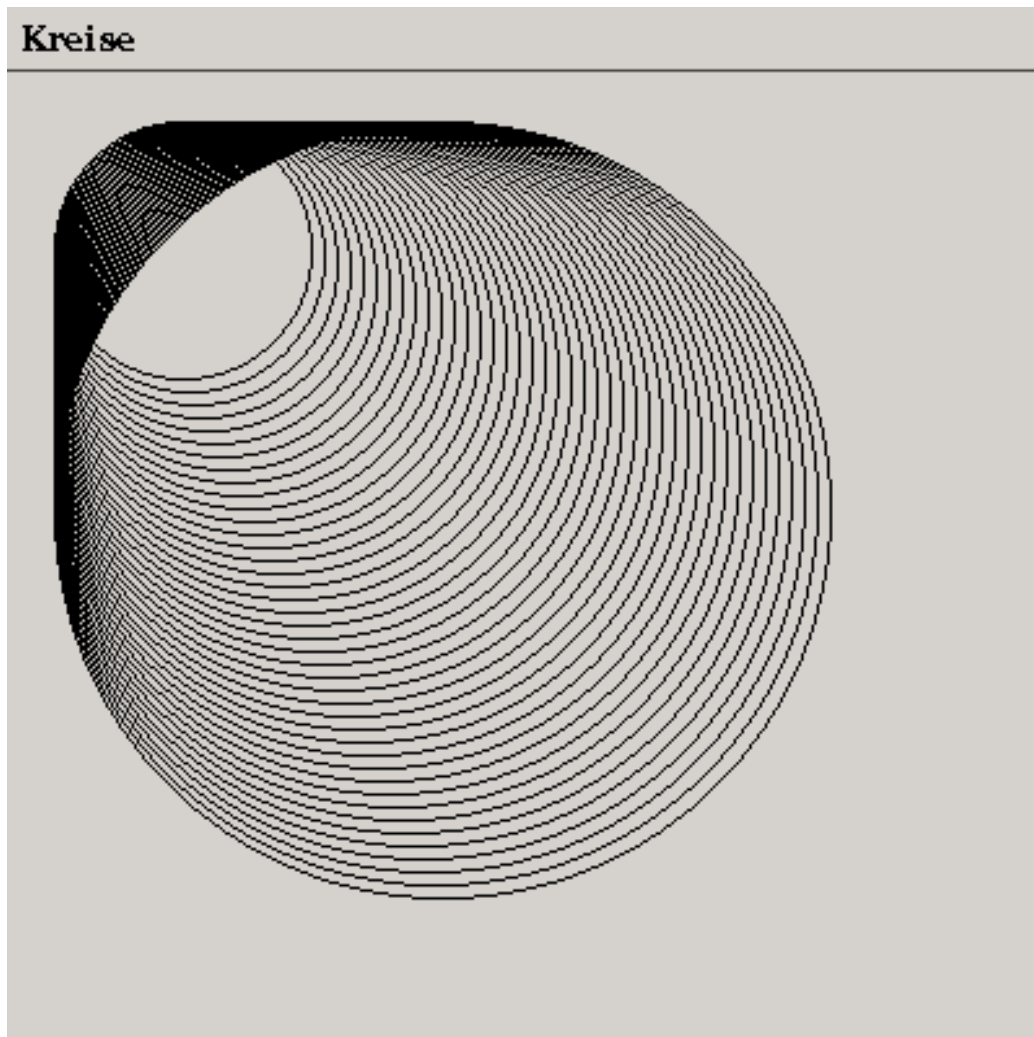


Abbildung 57: Bild:Prgkreis4.png

Der Code dazu:

```
PUBLIC SUB Form_Open()  
    DrawingArea1.Resize(ME.Width, ME.Height)  
END  
PUBLIC SUB Menu2_Click()  
    x AS Integer  
    Draw.Begin(DrawingArea1)  
    FOR x = 0 TO 200 STEP 5  
        Draw.Ellipse(10, 10, x + 100, x + 100,5)  
    NEXT  
    Draw.End  
END  
PUBLIC SUB Menu3_Click()  
    ME.Close
```

END

### 4.33.2 Konzentrische Kreise

Im folgenden Beispiel werden mehrere konzentrische Kreise in eine DrawingArea gezeichnet. Sie brauchen

- eine Drawingarea ( Grafikfeld)
- einen Befehlsbutton

um das Programm in Gang zu bringen.

```
PUBLIC SUB Form_Open()  
  Drawingareal.cached = TRUE  
  IF Drawingareal.width > Drawingareal.height THEN  
    Drawingareal.width = Drawingareal.height  
  ELSE  
    Drawingareal.height = Drawingareal.width  
  ENDIF  
  Button1.Text = "Kreise"  
  ME.Text = "Konzentrische Kreise"  
  END  
  PUBLIC SUB Button1_Click()  
    mx AS Integer  
    'x Koordinate des Mittelpunktes  
    my AS Integer  
    'y Koordinate des Mittelpunktes  
    'Radius = mx - a  
    a AS Integer  
    'linke obere Ecke x Koordinate  
    b AS Integer  
    'linke obere Ecke y Koordinate  
    c AS Integer  
    'a + c ergibt die x Koordinate der rechten unteren Ecke  
    d AS Integer  
    'b + d ergibt die y Koordinate der rechten unteren Ecke  
    mx = Drawingareal.width / 2  
    Draw.Begin(Drawingareal)  
    a = 0  
    FOR a = 0 TO mx STEP 10  
      b = a  
      c = 2 * (mx - a)  
      d = c  
      draw.Ellipse (a,b,c,d)  
    NEXT  
    draw.End  
  END
```

Die folgende Abbildung zeigt einen Screenshot des Programmes. Eine Kleinigkeit ist allerdings falsch. Fällt es Ihnen auf.

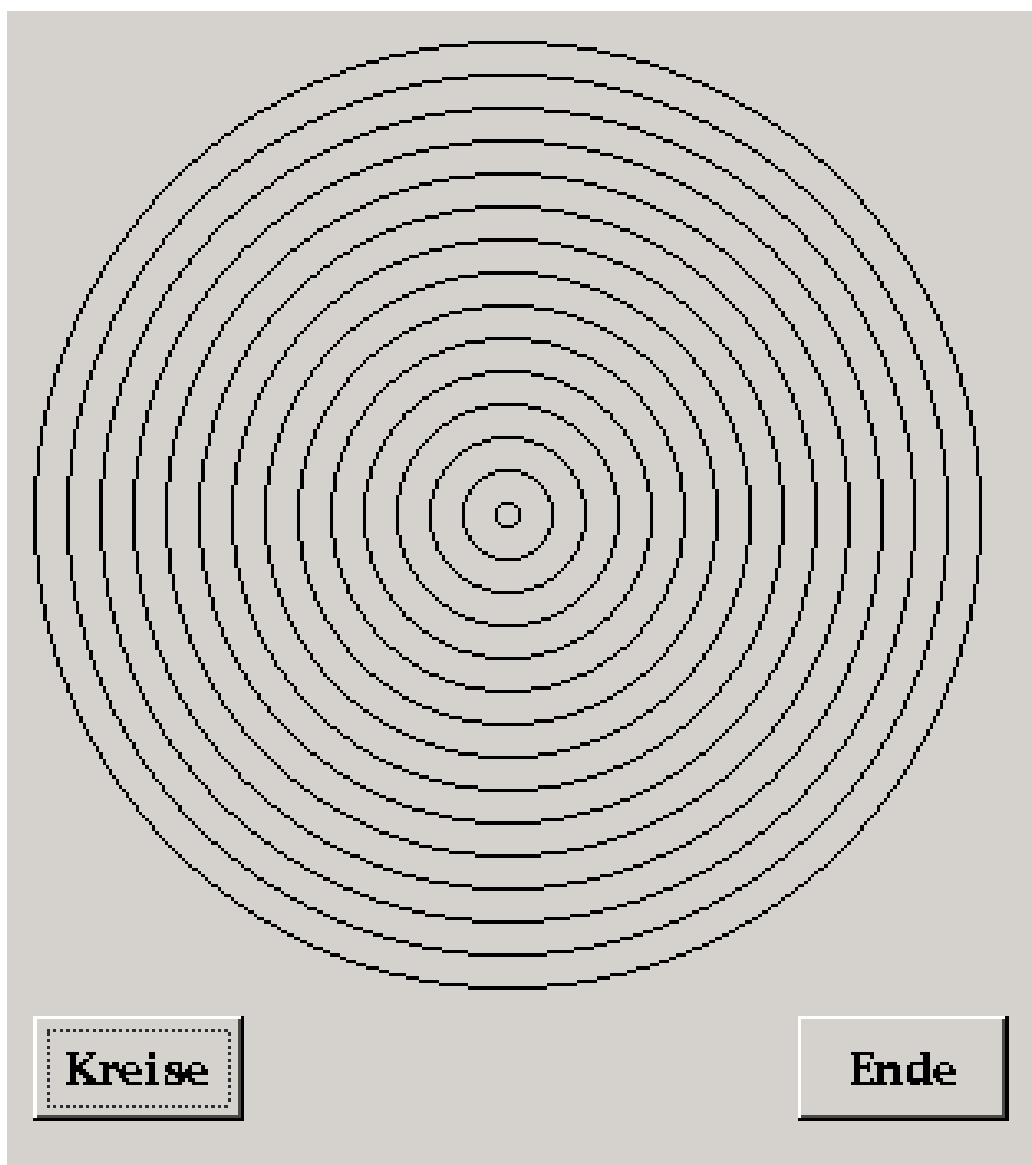


Abbildung 58: Bild:Gambaskreiskonzentrisch.png

Der zweite Befehlsbutton und der Code dazu fehlt im obigen Programm.

```
PUBLIC SUB Button2_Click()  
    ME.Close  
END
```

## 4.34 Tortengrafik

Fügt man dem Befehl *Draw(Ellipse)* außer den 4 notwendigen Parametern noch 2 weitere Parameter hinzu erhält man Kreissegmente, die man gut für eine Tortengrafik nutzen kann.

- Parameter 5, Gradzahl bei der das Segment beginnen soll
- Parameter 6, Gradzahl, bei der das Segment endet soll.

Beispiel:

```
PUBLIC SUB Button1_Click()  
Draw.Begin(Drawingareal)  
Draw.Ellipse(10, 50, 90, 90, 10, 90)  
Draw.End  
END
```

Man erhält einen Viertelkreis , der bei zehn Grad im Westem beginnt und entgegen des Uhrzeigersinnes läuft.

Siehe auch: <http://www.binara.com/gambas-wiki/bin/view/Gambas/DrawFigures>

## 4.35 Ellipse

Um die wichtigsten Parameter des Befehls *Draw.Ellipse* zu verstehen, können Sie folgendes Programm ausprobieren:

### 4.35.1 Layout

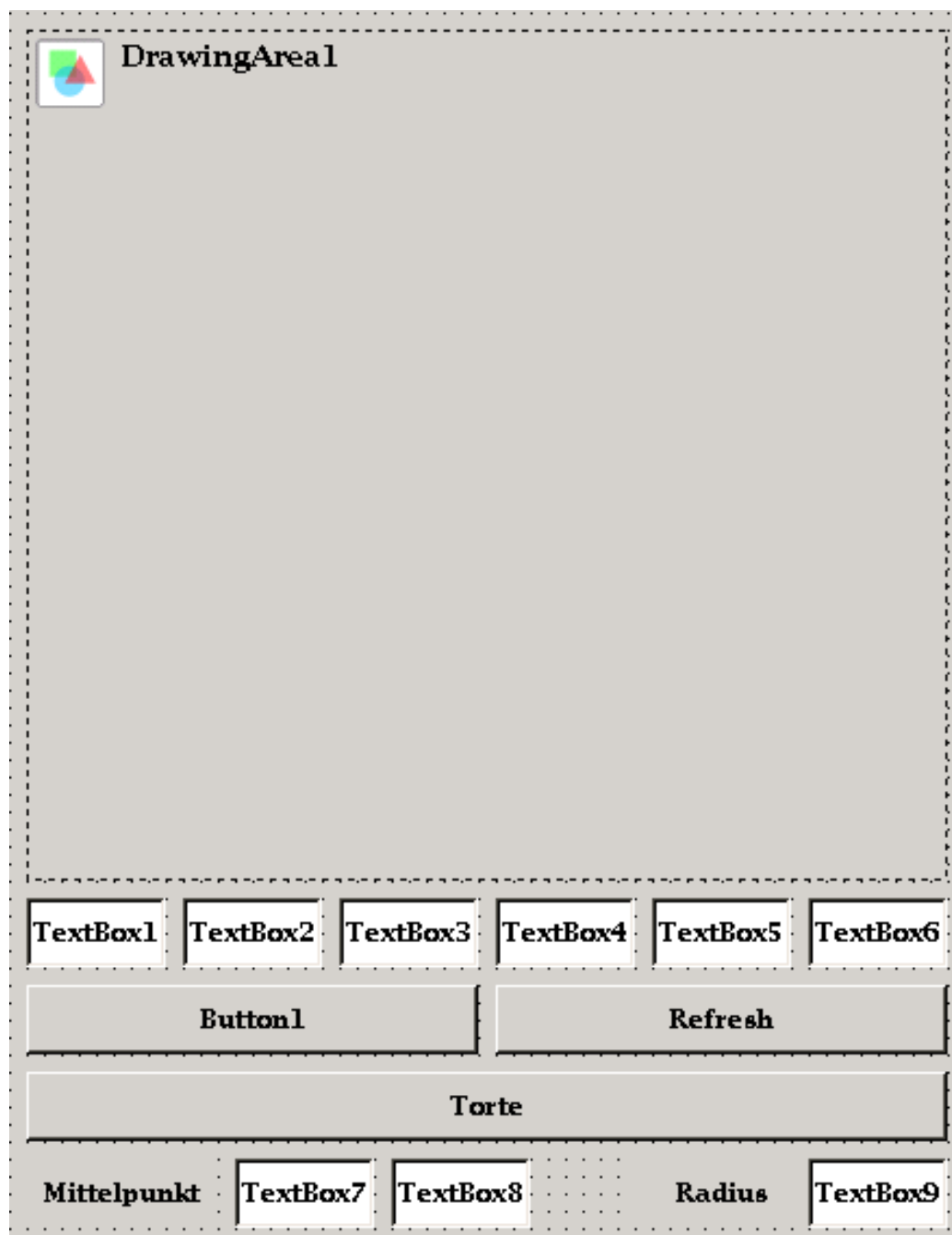


Abbildung 59: Bild:Drawellipseparameter0.png

Sie brauchen:

- 9 Textboxen auf Ihrer Form
- 1 Drawingarea
- 3 Befehlsbuttons
- 2 Labelfelder

auf Ihrer Form um das Programm in Gang zu bringen.

### 4.35.2 Der Code

```
PUBLIC SUB Form_Open()
textbox1.Text = "0"
textbox2.Text = "0"
textbox3.Text = Str(Drawingareal.Width)
textbox4.Text = Str(Drawingareal.Height)
textbox5.Text = "10"
textbox6.Text = "10"
textbox7.Text = Str(0 + Drawingareal.Width/2)
textbox8.Text = Str(0 + Drawingareal.Height/2)
textbox9.Text = Str(Drawingareal.Width/2)
Button1.Text = "Kreis"
END
PUBLIC SUB Button1_Click()
a AS Integer
b AS Integer
c AS Integer
d AS Integer
a = Val(Textbox1.text)
b = Val(Textbox2.text)
c = Val(Textbox3.text)
d = Val(Textbox4.text)
Draw.Begin(Drawingareal)
Draw.ellipse(a,b,c,d)
Draw.End
textbox7.Text = Str(a + c/2)
textbox8.Text = Str(b + d/2)
textbox9.Text = Str(c/2)
CATCH
message.Info("Bitte geben Sie in jedes Feld eine Zahl ein !")
END
PUBLIC SUB Button2_Click()
Draw.Begin(Drawingareal)
Drawingareal.Refresh
Draw.End
END
PUBLIC SUB Button3_Click()
a AS Integer
b AS Integer
c AS Integer
d AS Integer
e AS Float
f AS Float
a = Val(Textbox1.text)
```



```
b = Val(Textbox2.text)
c = Val(Textbox3.text)
d = Val(Textbox4.text)
e = Val(Textbox5.text)
f = Val(Textbox6.text)
Draw.Begin(Drawingareal)
Draw.ellipse(a,b,c,d,e,f)
Draw.End
CATCH
message.Info("Bitte geben Sie in jedes Feld eine Zahl ein !")
END
```

Und so schaut das Programm zur Laufzeit aus :

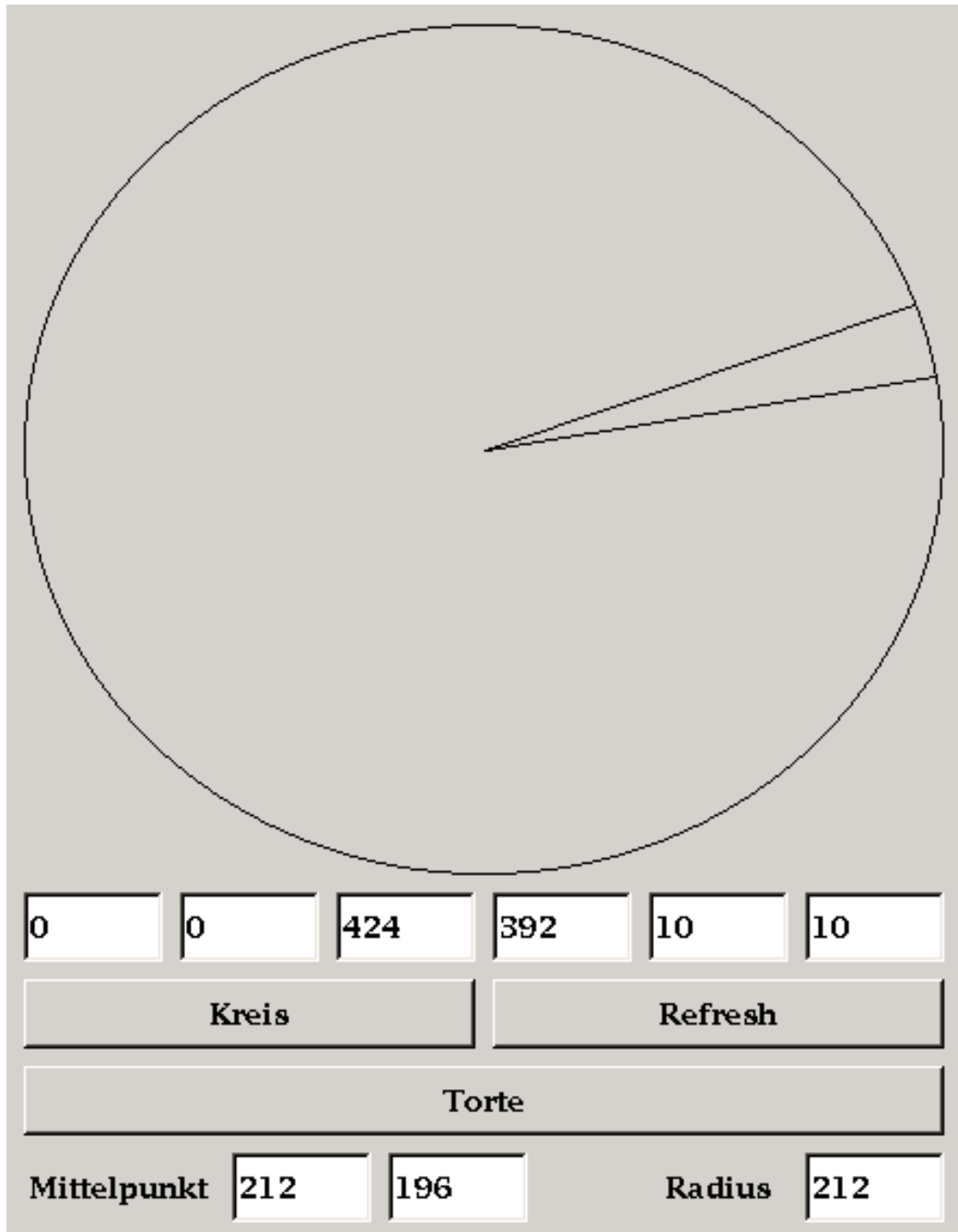


Abbildung 60: Bild:Drawellipseparameter.png

## 4.36 Der Seestern

**”Der kleine G. hatte gar nicht gewußt, dass man unterwasser auch so schöne Sterne finden konnte.”**

Das Seesternprogramm zeigt den Übergang von einer geordneten Struktur ins Chaos. Normalerweise haben Seesterne eigentlich 5 Beine. Bei unserem Programm ist eines verloren gegangen.

Um das Programm in Gang zu bringen, braucht man eine DrawingArea und einen Befehlsbutton.

## 4.37 Layout

Seestern

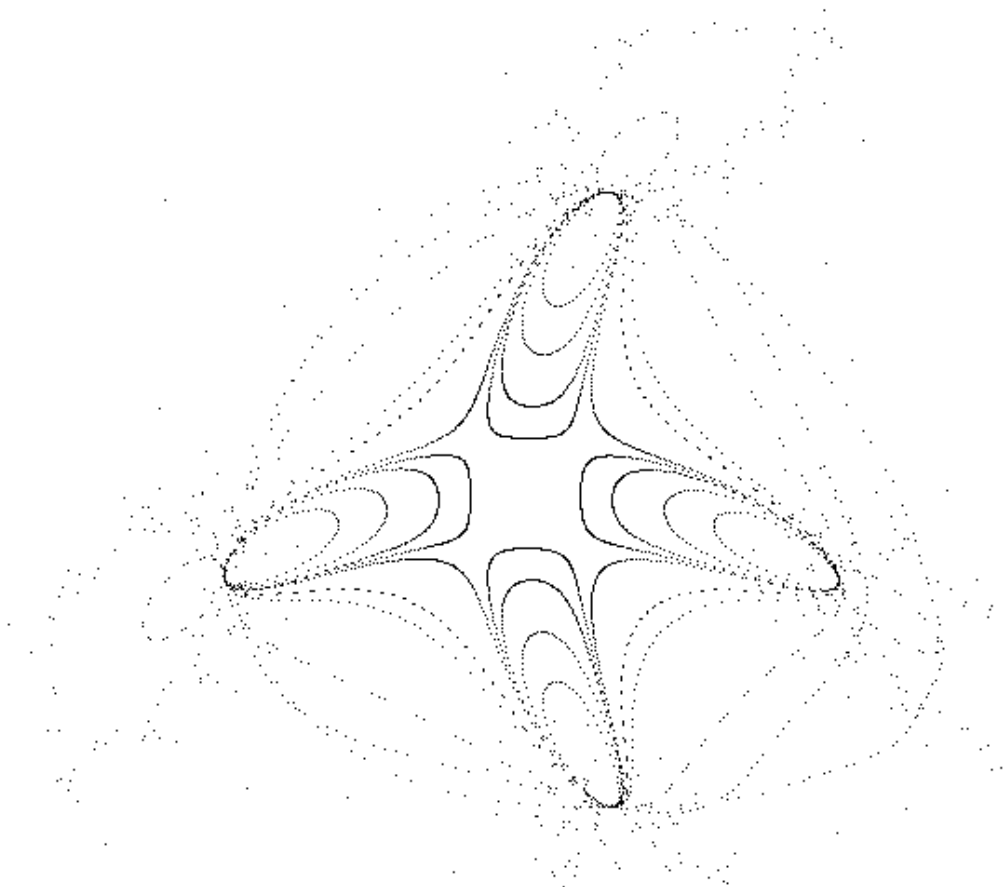


Abbildung 61: Bild:Gambasseestern.png

## 4.38 Der Code

```
PUBLIC SUB Form_Open()  
    DrawingAreal.Width = 1000  
    DrawingAreal.H = 1000  
    DrawingAreal.X = 0  
    DrawingAreal.Y = 0  
    DrawingAreal.Background = &HFFFFFF&  
    Button1.Text = "Seestern"  
END  
PUBLIC SUB Button1_Click()
```

```
E AS Integer
AW AS Float
H AS Integer
X AS Float
Y AS Float
I AS Integer
A AS Float
B AS Float
C AS Float
Draw.Begin(DrawingArea1)
FOR E = 1 TO 13
IF E = 1 THEN
AW = 55
H = 100
ENDIF
IF E = 2 THEN
AW = 80
H = 120
ENDIF
IF E = 3 THEN
AW = 88
H = 120
ENDIF
IF E = 4 THEN
AW = 90
H = 150
ENDIF
IF E = 5 THEN
AW = 90.1
H = 300
ENDIF
IF E = 6 THEN
AW = 90.2
H = 490
ENDIF
IF E = 7 THEN
AW = 90.21
H = 1000
ENDIF
IF E = 8 THEN
AW = 90.22
H = 600
ENDIF
IF E = 9 THEN
AW = 90.25
H = 250
ENDIF
IF E = 10 THEN
AW = 90.3
H = 180
ENDIF
IF E = 11 THEN
AW = 91
H = 100
ENDIF
IF E = 12 THEN
AW = 92
H = 100
ENDIF
IF E = 13 THEN
AW = 92.2
H = 410
```

```
ENDIF
X = 0.1513
Y = -0.5388
'Pi = 3.141593:
A = AW * Pi / 180'
'Rem umrechnung d.360 Grad Winkels ins Bogenmass
FOR I = 0 TO H
B = X * Cos(A) - (Y - X * X) * Sin(A)
C = X * Sin(A) + (Y - X * X) * Cos(A)
X = B
Y = C
'Draw.Point (400 + E ,400 + E)
Draw.Point(B * 300 + 300, C * 300 + 300)
'PRINT B, C
NEXT
'FOR z = 1 TO 1000000: NEXT z: Rem Zeitverzoeigerung kann man variieren
NEXT
Draw.End
END
```

## 4.39 Picture,Image,Drawingarea,Picturebox etc

### 4.39.1 Was ist der Unterschied zwischen einem Image und einem Picture ?

Unter Linux werden Bilder und Grafiken mit X11 dargestellt. Das X System hat dabei eine Netzkomponente. Ein Bild, welches von einem Programm bearbeitet wird, kann auf einem ganz anderen Rechner im Netz gezeigt werden.

Ein **Image in Gambas** ist ein Bild, das auf ein Programm beschränkt bleibt. So kann man schneller darauf zugreifen und mehr mit dem Bild anstellen. ( Strecken, drehen etc)

Ein **Picture in Gambas** ist ein Bild das unter X dargestellt wird, ob das nun auf derselben Maschine oder sonst wo im Netz passiert , ist dabei offen. Jede Veränderung an dem Bild geht also den Umweg über einen Netzaufruf oder eine lokale pipe. Deswegen funktioniert das deutlich langsamer.

In Programmen mit intensiver Bildernutzung, wird man je nach Bedarf von einem in das andere Format wechseln und konvertieren, wenn es nützlich erscheint.

Das Image Bildformat ist also gut für Bildmanipulationen, das Picture Bildformat ist besser geeignet für reine Darstellungszwecke.

## **4.40 Grafikfeld ( DrawingArea)**

siehe [Gambas:\\_Grafik](#)

## **4.41 Bildfeld (Picturebox)**

## **4.42 Scrollviewfeld**

Um mit Bilder oder Grafiken zu arbeiten ist das Steuerelement ScrollView ganz praktisch. Man findet es in der Werkzeugkiste (F6) neben der DrawingArea.

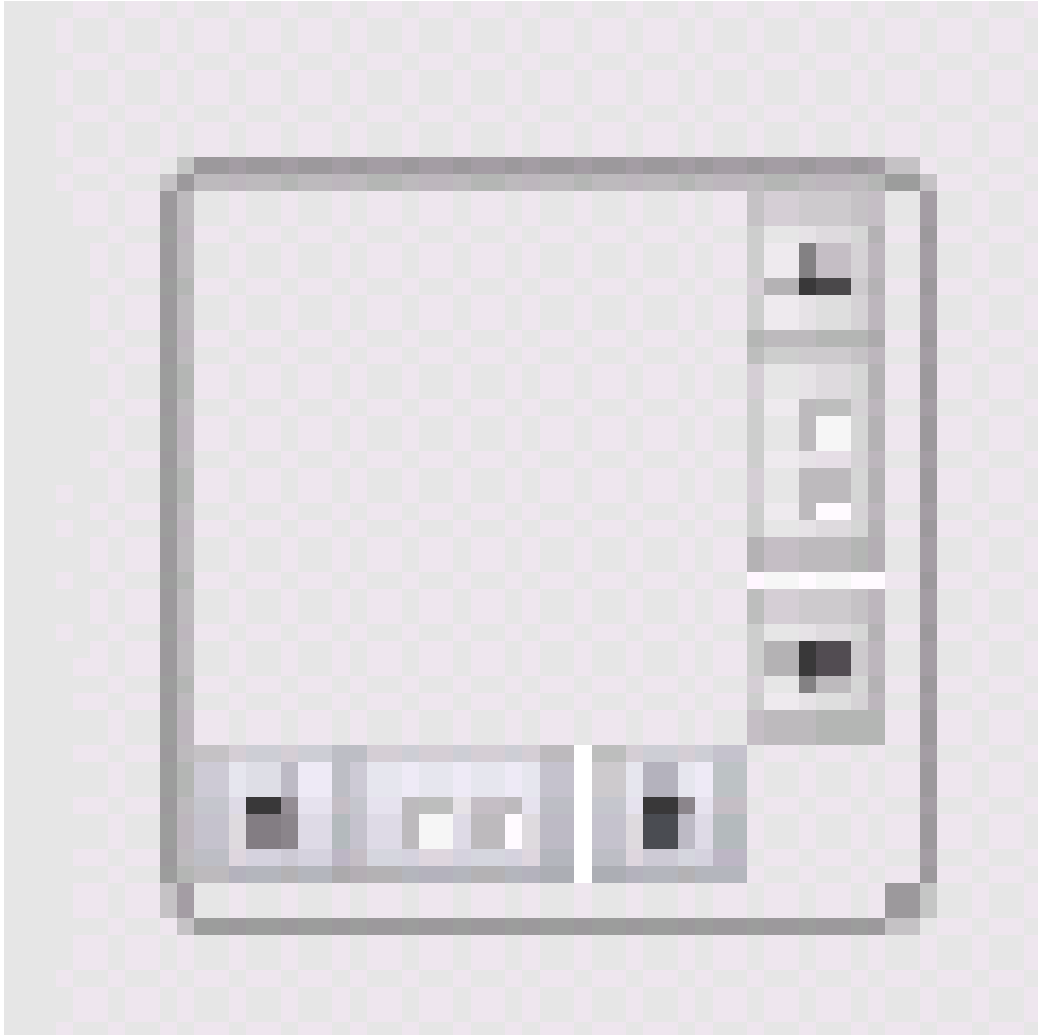


Abbildung 62: Bild:GambasScrollview.png

Mit dem Scrollviewfeld kann man auch größere Bilder anschauen, denn es stellt eine vertikale und horizontale Schiebeleiste bereit.

Wenn man das Scrollviewfeld nutzen will, dann plaziert man es als erstes auf der Form. Man macht es etwas größer. Dazu zieht man es an der rechten unteren Ecke größer auf. Dann läßt man es markiert. Man erkennt dies an den kleinen weißen Rechtecken am Rand des Steuerelementes. Jetzt plaziert man in das Scrollviewfeld hinein eine DrawingArea oder eine PictureBox. Auch diese zieht man etwas größer.



Mittels Code muß man dann beide Steuerelemente aneinander anpassen. Wie das geht zeigt das Programmbeispiel in *Laden Sie ein Bild in eine PictureBox*. weiter unten.

## 4.43 Laden Sie ein Bild in eine PictureBox

Mit diesem kleinen Programm wird ein Screenshot in eine PictureBox geladen.

Wenn Sie nicht wissen, was ein Screenshot ist, dann schauen Sie hier nach:

<http://de.wikipedia.org/wiki/Screenshot>

Um das Programm in Gang zu bringen, brauchen Sie eine Form, einen Befehlsbutton, ein Scrollviewfeld. In dieses Scrollviewfeld platzieren Sie eine PictureBox.



Abbildung 63: Bild:GambaspictureBoxicon.png

Code:

```
PUBLIC SUB Button1_Click()
```

```
    bild AS NEW Picture
    bild = Desktop.Grab()
    'Screenshot wird gemacht
    PictureBox1.Picture = bild
END
```

Sie können diesen Code verbessern, wenn Sie die PictureBox an das Scrollviewfeld anpassen:

```
PUBLIC SUB Button1_Click()
    bild AS NEW Picture
    'Form1.Hide
    'Form wird ausgeblendet
    bild = Desktop.Grab()
    'Screenshot wird gemacht
    PictureBox1.Resize(bild.Width, bild.Height)
    'PictureBox so groß machen wie das Bild
    PictureBox1.X = 0
    PictureBox1.Y = 0
    'PictureBox an Scrollview anpassen
    PictureBox1.Picture = bild
    'Form1.Show
END
```

## 4.44 Umwandeln eines Pictures in ein Image

Im folgenden Beispiel wird eine Screenshot erzeugt. Dieser wird in ein Image umgewandelt und in einer DrawingArea angezeigt.

Um das Programm in Gang zu bringen brauchen Sie:

- Eine Form
  - Einen Commandbutton = Befehlsknopf
  - Ein ScrollViewfeld = Feld mit vertikaler und horizontaler Verschiebbarkeit
    - \* Im Scrollviewfeld eine Drawingarea

Der Code:

```
PUBLIC SUB Button1_Click()
    bild AS NEW Picture
    i AS NEW Image
    bild = Desktop.Grab()
    'hier wird ein Screenshot gemacht
    i = bild.image
    'hier wird das bild in ein image umgewandelt
    DrawingArea1.Cached = TRUE
    DrawingArea1.Resize(i.Width,i.Height)
    DrawingArea1.Clear()
    Draw.Begin(DrawingArea1)
    Draw.Image(i, 0, 0)
    Draw.End
```

```
DrawingArea1.Visible = TRUE
DrawingArea1.Refresh
END
```

## 4.45 Laden Sie ein Bild mit Dialog in eine PictureBox

Mit dem folgenden Programm können Sie ein Bild mit folgendem Format: jpg; jpeg; png; bmp mit einem Dialogfenster in eine PictureBox laden.

Um das Programm in Gang zu bringen brauchen Sie

- eine PictureBox aus der Werkzeugkiste
- einen Befehlsbutton in der PictureBox

Code:

```
PUBLIC SUB Button1_Click()
    p AS picture
    Dialog.Path = "/usr/share/wallpapers"
    IF NOT Exist(Dialog.Path) THEN
        Dialog.Path = System.Home
    ENDIF
    Dialog.Filter = [ "Picture files (*.jpg; *.jpeg; *.png; *.bmp)", "All files (*.*)" ]
    IF Dialog.OpenFile() THEN RETURN
    p = NEW picture
    p.Load(Dialog.Path)
    PictureBox1.Resize(p.Width, p.Height)
    PictureBox1.Picture = p
    PictureBox1.Move(0, 0, ME.ClientW, ME.ClientH)
    PictureBox1.Visible = TRUE
CATCH
    Message.Warning(Error.Text & " !")
END
```

Bei diesem Programm stört noch, dass das Bild beim Laden 2 mal angezeigt wird. Wie kann man diese vermeiden ?

## 4.46 Laden Sie ein Bild ohne Dialog in die PictureBox

Vorsicht. Dieses Beispiel ist etwas kompliziert.

Mit dem folgenden Programm kann man ein Bild in die PictureBox laden. Das Programm beruht auf dem Beispiel Programm blights, welches mit Gambas mitgeliefert wird.

Starten sie ein neues Projekt . Holen Sie sich eine neue Form mit der Rechten Maustaste auf Form im Projekt. Nennen Sie die Form wie sie wollen. Platzieren sie auf der Form 2 Befehlstasten und eine PictureBox . Speichern Sie in das Programmverzeichnis 2 Bilddateien **blon.xpm** und **bloff.xpm**, die sie unter blights in den Beispielen von Gambas finden.

Wenn Sie die Bilder nicht finden, nehmen sie andere Bilddateien oder schauen Sie einmal in folgendem Verzeichnis nach: /opt/gambas/share/gambas/examples/Basic

Mit den Befehlstasten An und Aus laden Sie nun verschiedene Bilder in die PictureBox1.

Der Programmcode schaut so aus :

```
' Gambas class file
PUBLIC SUB Button1_Click()
PictureBox1.Picture = Picture["bloff.xpm"]
END
'-----
PUBLIC SUB Button2_Click()
PictureBox1.Picture = Picture["blon.xpm"]
END
```

## **4.47 Bild speichern ohne Dialog**

## **4.48 Bild speichern mit Dialog**

## **4.49 Bild aus dem Clipboard holen**

## **4.50 Bild in den Zwischenspeicher laden**

## **4.51 Resize**

Mehrere grafische Elemente von Gambas kennen die Resizemethode. Wie der Name schon sagt kann man damit die Größe eines grafischen Objektes auf dem Bildschirm verändern.

Wie das geht zeigt folgendes Beispiel:

```
PUBLIC SUB Form_Open()
  TextBox2.Text = "200"
  TextBox3.Text = "200"
  Button1.Text = "Resize Button1"
  Button1.Width = 200
  Button1.Height = 50
END
PUBLIC SUB Button1_Click()
  Button1.Resize(Val(TextBox1.Text),Val(TextBox2.Text))
END
```

Um das Programm in Gang zu bringen brauchen Sie 2 Textboxen und einen Befehlsbutton auf Ihrer Form. Drückt man auf den Befehlsbutton nach Start des Programmes, dann wird seine Größe an Hand der Werte in den 2 Textboxen verändert. Bereits beim Start des Programmes wird das Unterprogramm Form\_Open abgearbeitet und macht ein paar sinnvolle Voreinstellungen. Im Unterprogramm Button1.Click wird dann geresized d.h die Größe verändert. Als Alternative kann man natürlich auch eingeben:

```
Button1.Width = Val(TextBox1.Text)
Button1.Height = Val(TextBox2.Text)
```

Man kann also auch ohne Resizemethode resizen.

Beispiel 2: Automatisches Resize einer Textarea an eine Form:

```
PUBLIC SUB Form_Resize()
  TextArea1.Move(0, 0, ME.ClientWidth, ME.ClientHeight)
END
```

Sie brauchen eine Form und eine Textarea um das Programm in Gang zu bringen. Ganz egal wo die Textarea steht, sie wird an die Form angepasst. Auch wenn man die Form verschiebt bleibt, die Textarea an der Form kleben.

## 4.52 Cached

## 4.53 Refresh

## 4.54 Screenshot

Mit diesem kleinen Programm wird ein Screenshot in eine PictureBox geladen.

Wenn Sie nicht wissen, was ein Screenshot ist, dann schauen Sie hier nach:

<http://de.wikipedia.org/wiki/Screenshot>

Um das Programm in Gang zu bringen, brauchen Sie eine Form, einen Befehlsbutton, ein Scrollviewfeld. In dieses Scrollviewfeld platzieren Sie eine PictureBox.

Code:

```
PUBLIC SUB Button1_Click()  
    bild AS NEW Picture  
    bild = Desktop.Grab()  
    'Screenshot wird gemacht  
    PictureBox1.Picture = bild  
END
```

## 4.55 Farben

Man kann die Farbe verschiedener Objekt ganz einfach über das Eigenschaftsfenster des Objektes verändern ( Vordergrund , Hintergrund etc). Das Eigenschaftsfenster öffnet man im Entwurfsmodus wie üblich mit F4.

Man kann die Farbeinstellungen aber auch während der Laufzeit ändern. Dies soll im folgenden Programmbeispiel gezeigt werden:

Dieses Miniprogramm zeigt die RGB Farben. Sie können in den Textboxen Werte von 0 - 255 eingeben und sich die Farbe anschauen. Aus den 3 Dezimalzahlen wird ein Hexadezimalwert als String gebastelt:

```
sHEX = Hex$(r,2) & Hex$(g,2) & Hex$(b,2)  
sHEX = "&H" & sHEX & "&"
```

Wollen Sie das Programm zum laufen bringen brauchen Sie auf Ihrer Form folgende Felder, die Sie sich aus dem Werkzeugkasten holen müssen:

- 3 Textlabelfelder
- 3 Textfelder ( TextBox1-3 ) für die RGB Wert eingabe
- 1 Befehlsfeld ( = Command Button )
- 1 Zeichenfeld ( = DrawingArea ) für die Farbdarstellung

Das Ganze schaut dann so aus:

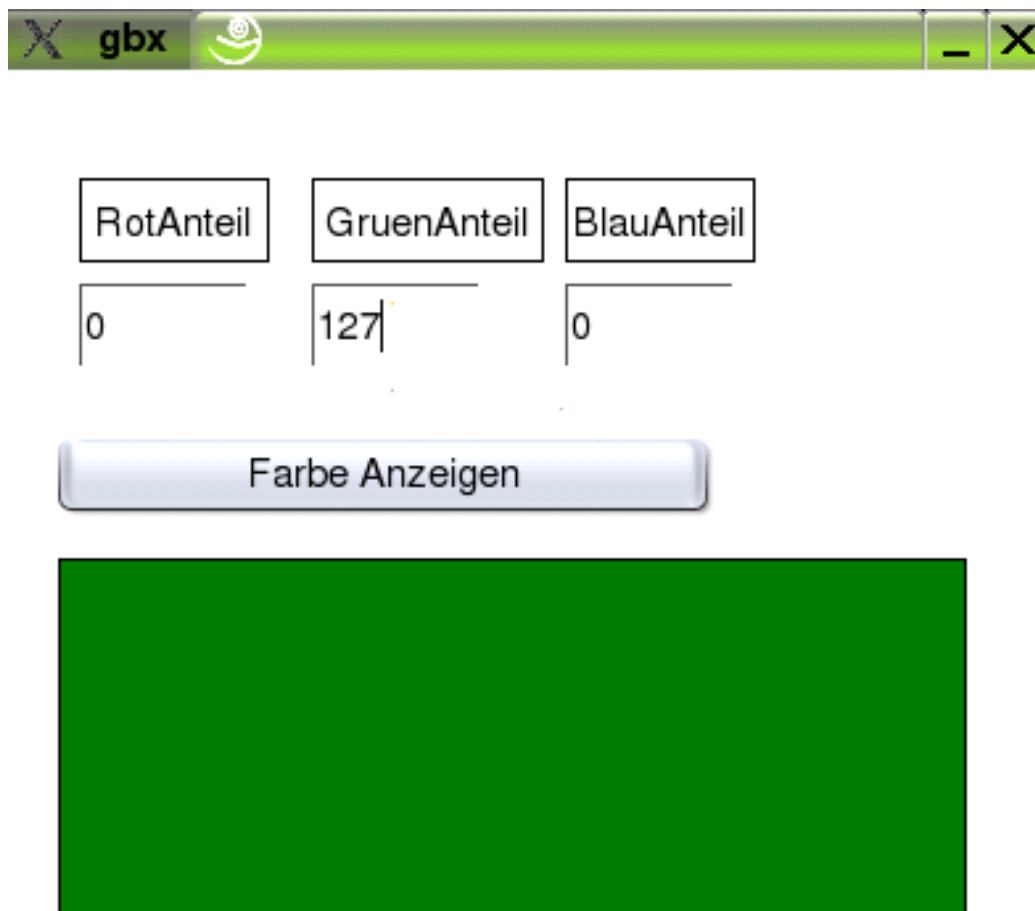


Abbildung 64: Bild:Grün.png

### Programmcode

```
PUBLIC SUB Button1_Click()  
DIM sHex AS String  
DIM r AS Integer  
DIM g AS Integer  
DIM b AS Integer  
IF Textbox1.text = "" THEN Textbox1.Text = 0  
IF Textbox2.text = "" THEN Textbox2.Text = 0  
IF Textbox3.text = "" THEN Textbox3.Text = 0  
r = Val(Textbox1.Text)  
g = Val(Textbox2.Text)  
b = Val(Textbox3.Text)  
sHEX = Hex$(r,2) & Hex$(g,2) & Hex$(b,2)  
sHEX = "&H" & sHEX & "&"  
DrawingArea1.BackColor = Val(sHEX)  
END
```

## 4.56 Farbspektrum

Das natürliche Farbspektrum des Lichtes reicht von dunkelrot über orange, gelb, grün nach blau und violett.

Farbe	Wellenlaenge	RGB
Rot	~ 625-740 nm	#DF0000
Orange	~ 590-625 nm	#FF8000
Gelb	~ 565-590 nm	#FFFF00
Gruen	~ 520-565 nm	#00FF00
Cyan	~ 500-520 nm	#00FFFF
Blau	~ 450-500 nm	#0000FF
Indigo	~ 430-450 nm	#0000AF
Violett	~ 380-430 nm	#50007F

Will man sich dieses Farbspektrum anschauen, hilft einem folgendes Programm. Man kann es noch verbessern, wenn man einen Schieberegler zur Farbwahl nutzt.

Man braucht für das Programm eine Form, eine DrawingArea und einen Commandbutton. Wenn man den Commandbutton mehrfach betätigt wird die Farbe immer weiter verändert. Sie beginnt bei rot und endet bei violett.

### Programmcode

```

sp AS Integer
PUBLIC SUB Form_Open()
  sp = 1
  drawingarea1.BackColor = &HC2020C&
END
PUBLIC SUB Button1_Click()
  DIM ss AS Integer
  ss = 1
  ss = sp
  drawingarea1.BackColor = Val(spektrum(ss))
  sp = sp + 20
  ' Hier kann man die Farbsprünge wählen , addieren sie zu sp 5,10,20,50 oder 100
  IF sp > 1120 THEN sp = 1
  'bei violett springt es wieder nach rot zurück
END
PUBLIC FUNCTION spektrum(ss AS Integer) AS String
  x AS Integer
  SHEX AS String
  DIM s AS Integer
  DIM r AS Integer
  DIM g AS Integer
  DIM b AS Integer
  s = 0
  FOR x = 1 TO 255
    'rot nach gelb
    r = 255

```



```
b = 0
sHEX = Hex$(r,2) & Hex$(x,2) & Hex$(b,2)
sHEX = "&H" & sHEX & "&"
s = s + 1
IF s = ss THEN RETURN sHEX
NEXT
's = 255
FOR x = 255 TO 1 STEP -1
'gelb nach grün
g = 255
b = 0
sHEX = Hex$(x,2) & Hex$(g,2) & Hex$(b,2)
sHEX = "&H" & sHEX & "&"
s = s + 1
'PRINT s
IF s = ss THEN RETURN sHEX
NEXT
's = 510
FOR x = 1 TO 255
'von grün nach cyan
r = 0
g = 255
sHEX = Hex$(r,2) & Hex$(g,2) & Hex$(x,2)
sHEX = "&H" & sHEX & "&"
s = s + 1
'PRINT s
IF s = ss THEN RETURN sHEX
NEXT
's = 765
FOR x = 255 TO 1 STEP - 1
'von cyan nach blau
r = 0
b = 255
sHEX = Hex$(r,2) & Hex$(x,2) & Hex$(b,2)
sHEX = "&H" & sHEX & "&"
s = s + 1
'PRINT s
IF s = ss THEN RETURN sHEX
NEXT
's = 1020
b = 255
FOR x = 1 TO 100
'von blau nach lila
g = 0
sHEX = Hex$(x,2) & Hex$(g,2) & Hex$(b,2)
sHEX = "&H" & sHEX & "&"
b = b - 2
s = s + 1
'PRINT s
'PRINT shex
IF s = ss THEN RETURN sHEX
NEXT
's = 1120
END
```

## **4.57 Wie man MouseDown, MouseMove und MouseUp benutzt**

Wenn die linke Maustaste gedrückt wird, dann wird ein mousemove-Ereignis in regelmäßigen Abständen gestartet .

Das Programm

Holen Sie sich eine neue Form und benennen Sie diese als Fmaus. Deklarieren Sie diese als Startform.

Holen Sie sich aus der Werkzeugkiste 3 Malfelder ( Drawingarea ) und benennen Sie diese in den Eigenschaften ( = Properties) in da1, da2 und da3 um. Kopieren Sie sich den unten angegebenen Code und Compilieren Sie das Programm. Läuft es ? Falls ja experimentieren Sie etwas mit der Maus und den Farben herum.

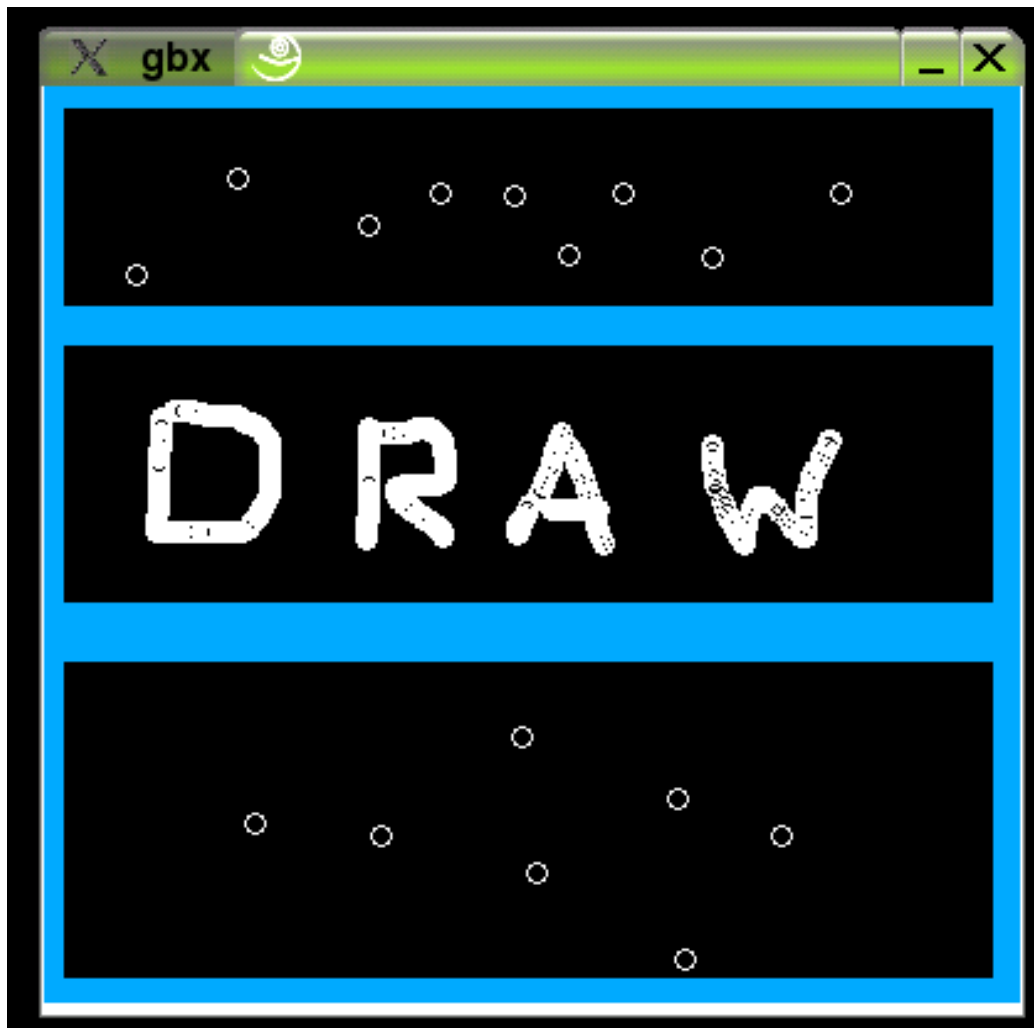


Abbildung 65: Bild:prgmaus.png

Klicken Sie in die schwarzen Felder ! Im zweiten Feld muß man die Mausetaste gedrückt halten. Im dritten Feld muss man die Mausloslassen.

Der Code:

```
STATIC PUBLIC SUB Main()  
  hForm AS Fmaus  
  hForm = NEW Fmaus  
  hForm.show  
END  
PUBLIC SUB dal_MouseDown()  
  Draw.Begin(dal)  
  Draw.Ellipse(Mouse.X,Mouse.Y, 9, 9)  
  Draw.End  
END
```

```
PUBLIC SUB da2_MouseMove()  
  Draw.Begin(da2)  
    Draw.Ellipse(Mouse.X,Mouse.Y, 9, 9)  
  Draw.End  
END  
PUBLIC SUB da3_MouseUp()  
  Draw.Begin(da3)  
    Draw.Ellipse(Mouse.X,Mouse.Y, 9, 9)  
  Draw.End  
END
```

## 4.58 Die Koordinaten der Maus

Folgendes Programm funktioniert nur in einem Grafikfeld (= DrawingArea).

Das Programm liefert fortlaufend die Koordinaten der Maus, wenn man sie auf dem Grafikfeld bewegt und die linke Maustaste gedrückt hält.

Machen Sie ein neues Projekt. Öffnen Sie eine neue Form: Mit der rechten Maus auf Form klicken. Geben Sie ihr irgendeinen Namen. Holen Sie sich eine DrawingArea = Zeichenfeld aus der Werkzeugkiste.

Holen Sie sich 2 Textboxen aus der Werkzeugkiste.

Die Form schaut dann so aus:

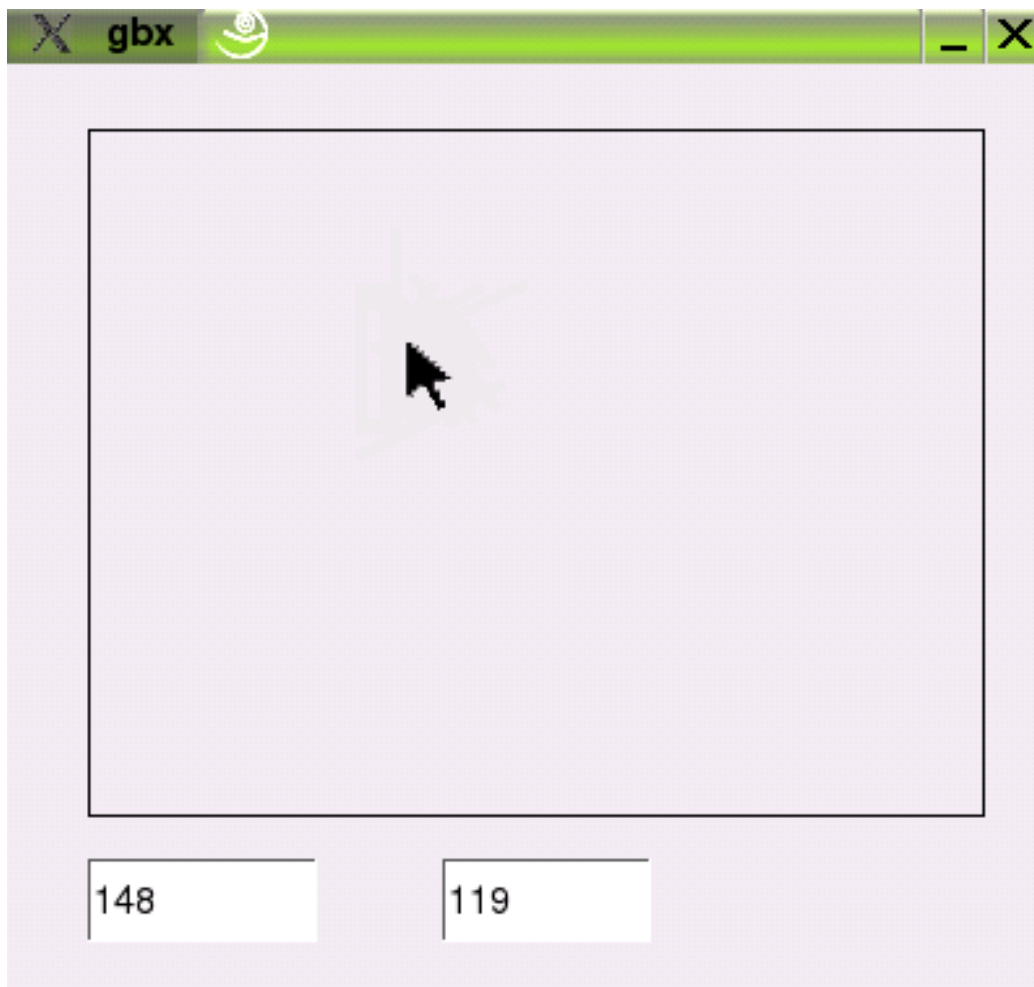


Abbildung 66: Bild:prgmausxy1.png

Kopieren Sie den Code von unten ins Programm. Starten Sie das Programm mit F5. Achten Sie auf die Farbgebung bei den Eigenschaften, sonst sehen Sie nichts. Achten Sie auch auf die Farbgebung der Form. Die Farbeigenschaften der Form werden auf die Eigenschaften der Formkomponenten übertragen.

Fahren Sie mit der Maus im Zeichenfeld umher und drücken Sie die linke Maustaste. Dann werden die Koordinaten der Maus in den Textboxen angezeigt.

Beachten Sie folgendes : die linke obere Ecke ist (0,0) , die rechte untere Ecke ist (DrawingArea1.Width, DrawingArea1.Height )

Der neue Code:

```
PUBLIC SUB DrawingArea1_MouseMove()
```

```
Textbox1.text = Mouse.X  
Textbox2.text = Mouse.Y  
END
```

Wie erhält man Absolute Koordinaten ?

Der alte Code: Gambas älter als 0.65

```
PUBLIC SUB DrawingArea1_MouseMove(X AS Integer, Y AS Integer, Button AS Integer)  
Textbox1.text = X  
Textbox2.text = Y  
END
```

## **4.59 Drag and Drop , Nehmen und Ziehen mit der Maus.**

Das Programm zeigt die Drag und Drop eigenschaft. Man zieht ein Bild mit der Maus auf der Form hin- und her solange man die linke Maustaste betätigt.

Das Programm funktioniert mit einem Bildfeld ( = Image ). Man kann aber auch andere Objekte auf diesselbe Art verschieben.

Machen Sie ein neues Projekt. Öffnen Sie eine neue Form: Mit der rechten Maus auf Form klicken. Geben Sie ihr irgendeinen Namen. Holen Sie sich mit F6 ein Image = Bild aus der Werkzeugkiste und belassen Sie den Namen als Image1 .

Die Form schaut dann so aus:

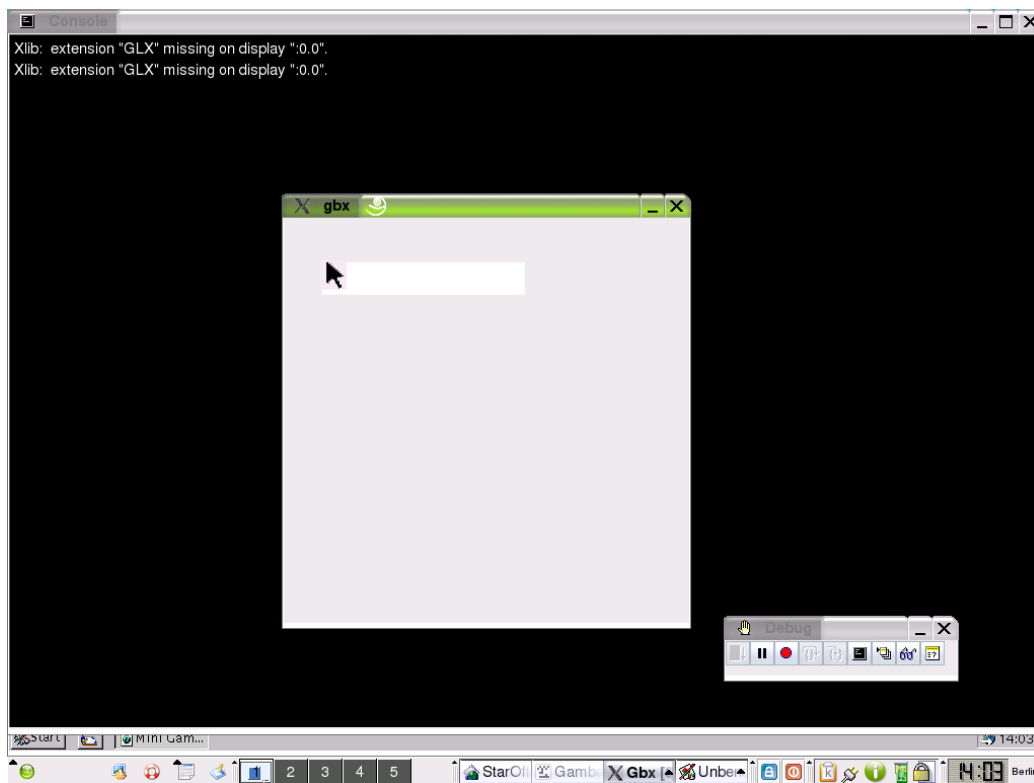


Abbildung 67: Bild:prgdragdrop.png

Kopieren Sie den Code von unten ins Programm. Starten Sie das Programm mit F5. Achten Sie auf die Farbgebung bei den Eigenschaften, sonst sehen Sie nichts. Achten Sie auch auf die Farbgebung der Form. Die Farbeigenschaften der Form werden auf die Eigenschaften der Formkomponenten übertragen.

Fahren Sie mit der Maus im Zeichenfeld umher und drücken Sie die linke Maustaste. Dann wird das Bildfeld entsprechend den Koordinaten der Maus verschoben.

Der Code:

```
PUBLIC SUB Form_MouseMove ()
Image1.X = Mouse.X
Image1.Y = Mouse.Y
END
```

Wenn Sie wollen, dass das Bild bereits beim ersten Mouseclick zur Maus springt, dann fügen Sie noch folgenden Code ein:

```
PUBLIC SUB Form_MouseDown ()
Image1.X = Mouse.X
Image1.Y = Mouse.Y
END
```

### alter Code

```
PUBLIC SUB Form_MouseMove(X AS Integer, Y AS Integer, Button AS Integer)
    Image1.X = X
    Image1.Y = Y
END
```

## 4.60 Maustasten abfragen

Im folgenden Programm wird die linke Maustaste abgefragt. Wird sie gedrückt, dann malt die Maus eine runde Form. Wird die rechte Taste gedrückt, wird eine rechteckige Form zum Malen benutzt.

Sie brauchen ein Zeichenfeld ( Drawingarea) auf Ihrer Form, um das Programm in Gang zu bringen.

```
PUBLIC SUB Form_Open()
    DrawingArea1.Cached = FALSE
    DrawingArea1.BackColor = &HFFFFFF&
    'weisser Hintergrund
END
PUBLIC SUB DrawingArea1_MouseMove()
    IF Mouse.Button = 1 THEN
        Draw.Begin(DrawingArea1)
        Draw.Ellipse(Mouse.X, Mouse.Y, 9, 9)
        Draw.End
    ELSE
        Draw.Begin(DrawingArea1)
        Draw.Rect(Mouse.X, Mouse.Y, 9, 9)
        Draw.End
    ENDIF
END
```

## 4.61 Eine andere Cursorform über die Eigenschaften festlegen

Man kann die Form des Mauszeigers ( Cursor) bei den Eigenschaften einiger Steuerelemente festlegen. Nimmt man eine leere Form und schaut sich deren Eigenschaften mit der F4 Taste an, dann findet sich dort in der Mitte der Eigenschaftstabelle eine Zeile mit der Bezeichnung Mouse. Clickt man diese Zeile im rechten Teil an, dann erhält man eine Auswahl von verschiedenen Mouseformen:

- Arrow
- Blank
- Cross



- Custom
- Default
- Horizontal
- Pointing
- SizeAll
- SizeE
- SizeH
- SizeN
- SizeNE
- SizeNESW
- SizeNW
- SizeNWSE
- SizeS
- SizeSE
- SizeSW
- SizeV
- SizeW
- SplitH
- SplitV
- Text
- Vertical
- Wait

Probieren Sie verschiedene Mouseformen aus und starten Sie Ihr Programm einfach mit F5, dann sehen Sie die gewählte Mauszeigerform.

Über einigen Steuerelementen läßt sich die Zeigerform wieder neu wählen oder sie ist bereits voreingestellt.

Sie können den Mauszeiger auch in ihrem Programmablauf ändern. Wie das geht zeigt folgendes Programm: Sie brauchen eine Form und 2 Befehlsknöpfe, um es in Gang zu bringen.

```
PUBLIC SUB Button1_Click()  
  Form1.Mouse = Mouse.Wait  
END  
PUBLIC SUB Button2_Click()  
  Form1.Mouse = Mouse.Pointing  
END
```

## 4.62 Die Zufallsmaus

Sie können die ganzen Mauszeigerformen anschauen , wenn Sie einmal folgendes Programm ausprobieren.

Um es in Gang zu bringen brauchen Sie nur einen Timer ( Taktgeber) auf Ihrer Form.

```
PUBLIC SUB Form_Open()  
  timer1.Enabled = TRUE  
  timer1.Delay = 300  
END  
PUBLIC SUB Timer1_Timer()  
  x AS Integer  
  Mouse AS Integer  
  Randomize  
  x = Int(Rnd(1,15))  
  form1.Mouse = x - 2  
END
```

Bereits beim starten des Programmes wird der Timer angeschaltet und seine Zeitverzögerung ( Delay ) auf einen mittleren Wert von 300 eingestellt. Im Unterprogramm des Timers werden Zufallszahlen zwischen 1 und 15 erzeugt. Diese Zufallszahlen werden der Eigenschaft des Mauszeigers zugeordnet. Deswegen variiert die Mausform alle möglichen Formen per Zufall durch. Die Eigenschaften der Maus finden Sie unter der Gambas Komponente Gb.QT .

Stört Sie der Zufall im obigen Programm. Dann probieren Sie einmal folgende Variante:

```
PUBLIC SUB Button1_Click()  
  x AS Integer  
  Mouse AS Integer  
  FOR x = -1 TO 13  
    form1.Mouse = x  
    WAIT 0.5  
    Button1.Text = "Maus" & Str(x)  
  NEXT  
END
```

In diesem Programm werden alle Mausformen mit einer Verzögerung von einer halben Sekunde ( Wait 0.5) durchprobiert. Um das Programm in Gang zu bringen brauchen Sie eine Form und einen Befehlsbutton.

Die Zeigerform der Maus kann über folgende konstanten Werte festgelegt werden, wenn man die Mauseigenschaft mit

```
Mouse AS Integer
```

definiert.

- Default = -1
- Arrow = 0
- Cross = 2
- Wait = 3
- Text = 4
- SizeS = 5
- SizeE = 6
- SizeNESW = 7
- SizeNWSE = 8
- SizeAll = 9
- Blank = 10
- SplitV = 11
- SplitH = 12
- Pointing = 13

## 4.63 Ein anderes Cursorbild mit dem Cursorbefehl

Man kann mit dem Befehl *Cursor* dem Cursor ein neues Symbol zuordnen. Wie das geht soll das folgende Beispielprogramm zeigen.

```
PUBLIC SUB Form_Open()  
hPict AS Picture  
hCursor AS Cursor  
hPict = Picture["neuerpfeil.png"]  
hCursor = NEW Cursor(hPict)  
ME.Cursor=hCursor  
END
```

Die Bilddatei *neuerpfeil.png* müssen sie mit einem Malprogramm neu erstellen und im Verzeichnis abspeichern, in dem auch ihr Gambasprogramm steht. Fehlt

die Bilddatei, dann ergibt das eine Fehlermeldung *Null objekt*, da das zu ladenden Bildobjekt für die Klasse *Cursor* fehlt.

### 4.63.1 Theorie

Hinter dem Befehl *Cursor* steht eine Klasse, in die ein beliebiges Bild als Mauszeiger geladen werden kann. Die Klasse ist zur Laufzeit erzeugbar. Dies wird auch im obigen Beispielprogramm so gehandhabt. Mit dem Befehl wird ein neuer *Cursor* aus einem *Picture* Objekt erzeugt. Die Mausspitze kann man genauer definieren, im allgemeinen wird die linke obere Ecke des Mausbildes herangezogen.

Die Syntax für den *Cursor* lautet dann:

```
hCursor = NEW Cursor ( Picture AS Picture [ , X AS Integer, Y AS Integer ] )
```

X und Y sind die Koordinaten der Mausspitze ( Hot spot)

## 4.64 Frage

Wie kann man den Mauszeiger zur Uhrform ( busy or wait = Warten ) und zurück verändern ?

Lösung:

```
ME.Mouse = Mouse.Wait  
...  
ME.Mouse = Mouse.Normal
```

## 4.65 Textausgabe

Die Textausgabe kann mit dem *Print* Befehl im Direktfenster oder in einer *Textbox* oder *Textarea* erfolgen.

Das kürzeste Gambas Programm mit einer Textausgabe lautet

```
PUBLIC SUB Form_Open()  
PRINT 1  
END
```

Man braucht eine leere Form, um es zu starten.

Starten Sie ein neues Projekt. Nennen Sie es wie sie wollen. Holen Sie sich eine neue Form mit der rechten Maustaste. Nennen Sie diese wie sie wollen. Es geht auch einfach F .

Dann kopieren Sie den Programmcode in den Gambas Programmcodebereich Form.class.

Starten Sie das Programm mit F5

Diese Programm nutzt den **Print** Befehl. Alles was das Programm macht: Es gibt die Zahl eins im Direktfenster aus.

Will man das Programm abändern, um die Textausgabe in einer Textbox zu erreichen, dann schaut der Code so aus:

```
PUBLIC SUB Form_Open()
  Textbox1.Text = "1"
END
```

Versucht man das Programm mit einer leeren Form zu starten, dann erhält man eine Fehlermeldung.

```
Unknown identifier: textbox1 Zeile 5 in Form1.class
```

Man braucht also eine Textbox auf der Form. Diese holt man sich mit F6 und Doppelklick auf die Textbox aus der Werkzeugkiste.

## 4.65.1 Print

Der gute alte Printbefehl ist etwas aus der Mode gekommen. Trotzdem kann man ihn noch sehr gut gebrauchen. Mit dem Print Befehl leitet man die Programmausgabe in die Standardausgabe ( meist die Textkonsole) um.

Dies ist unterschiedlich zu Visual Basic. Dort gab der Printbefehl sein Ergebnis in der Form aus. Das machte er ohne Rücksicht auf die Dinge, die sonst noch in der Form zu finden waren. Die Ausgabe war deswegen oft schlecht lesbar. Er hatte auch den Nachteil, keinen Zeilenumbruch zu können.

Man kann den Printbefehl in Gambas sehr gut zur Fehlersuche benutzen, wenn man mit ihm den Wert von Variablen in die Textkonsole umleitet und ausgibt. Viele Hilfetexte für Gambas benutzen außerdem den Printbefehl.

Versuchen Sie jetzt folgendes Beispiel:

```
PUBLIC SUB Button1_Click()
  DIM k AS Integer
  FOR k = 1 TO 10
    PRINT k;
  NEXT
  PRINT
  PRINT 1,2,3,4,5,6,7,8,9,10
```

```
FOR k = 1 TO 10
  PRINT k
NEXT
END
```

Holen Sie sich eine neue Form. Darauf plazieren Sie einen Befehlsknopf (= CommandButton) . Diesen können Sie sich mit F6 aus der Werkzeugkiste holen. Klicken Sie im Entwurfsmodus auf den Befehlsknopf und weisen Sie ihm den Programmcode weiter unten zu. Wenn das hier gezeigte Programm für sie noch zu kompliziert ist , dann geben Sie einfach einmal Print 1 oder Print "Hallo Welt! " ein.

Starten Sie das Programm mit F5 und drücken Sie auf den Befehlsknopf, dann sollten Sie folgende Ausgabe erhalten:

```
12345678910
1 2 3 4 5 6 7 8 9 10
1
2
3
4
5
6
7
8
9
10
```

Man beendet das Programm mit einem Klick auf die Raute rechts oben .

Der Printbefehl gibt Ausdrücke auf dem Standard Ausgabe gerät aus. Die Ausdrücke werden vorher mit der String Funktion Str() in eine Textausgabe umgewandelt.

Text muß für den Printbefehl in Anführungszeichen stehen:

Print "Hallo Welt" ist korrekt Print Hallo Welt funktioniert nicht.

Bei ganzen Zahlen können die Anführungszeichen weggelassen werden. Print "1" ist identisch mit Print 1.

Falls nach dem letzten Zeichen kein Komma oder Strichpunkt folgt, wird ein Neue Zeile Befehl automatisch angefügt.

Falls im Print Befehl ein Komma auftaucht wird eine Tabsprung (ASCII code 9) eingefügt. Meist entspricht der Tabsprung 5 Leerzeichen.

Der Doppelpunkt nach dem Printbefehl funktioniert in Gambas nicht.

```
Print 1: Print 2
```

**gibt eine Fehlermeldung.**

```
Syntax error
```

## 4.65.2 String\$

Mit dem Befehl String\$ kann man ein Zeichen oder eine Zeichenfolge mehrfach ausgeben

Syntax:

```
String = String$ ( Häufigkeit , Zeichenfolge )
```

Beispiel :

```
PRINT String$(12, "**")
```

Ausgabe: \*\*\*\*\*

```
PRINT String$(2, "Gambas")
```

Ausgabe: GambasGambas

## 4.66 Texteingabe

Die Texteingabe erfolgt über eine Textbox oder eine Textarea.

Man kann die Texteingabe auf die Eingabe bestimmter Zeichen begrenzen. Wie das geht, zeigt folgendes Beispiel. Es nutzt den **STOP EVENT** Befehl:

Sie brauchen eine Textbox auf Ihrer Form um das Programm in gang zu bringen:

```
'Meine Textbox verarbeitet nur Ziffern.
PUBLIC SUB TextBox1_KeyPress()
  IF Instr("0123456789", Key.Text) = 0 THEN
    STOP EVENT
  ENDIF
END SUB
```

Ähnliches können Sie mit dem folgenden Code erreichen. Interessant ist hier, die anfangs unverständliche Nutzung des If Then Befehles:

```
PUBLIC SUB Form_Open()
  ME.Text = "Nur Zahlen erlaubt !"
END
PUBLIC SUB TextBox1_KeyPress()
  IF key.Code >= 48 AND key.Code <= 57 THEN
  ELSE IF key.Code = key.BackSpace THEN
  ELSE IF key.Code = key.Delete THEN
  ELSE
    STOP EVENT
  ENDIF
END
```

## 4.67 Text Löschen

Mit dem folgenden kleinen Programm löscht man eine Textbox leer. Es ist ziemlich simpel. Holen Sie sich eine Textbox aus der Werkzeugkiste . Dann noch einen BefehlsButton. Siehe <http://www.madeasy.de/7/prgtxtdel.htm>

```
PUBLIC SUB Button1_Click()  
  TextBox1.Text = ""  
END
```

## 4.68 Text zusammenkleben mit &

Text kleben kann man in Gambas mit dem Befehl **&**. Das Zeichen finden Sie oberhalb der Ziffer 6. Man kann den Text nicht einfach mit dem Plus Zeichen zusammenkleben, denn im Gegensatz zu VB ist + in Gambas nur für die mathematische Addition reserviert.

Das Beispielprogramm klebt 2 Textteile aneinander und gibt sie in einer TextArea aus. Es ist ziemlich einfach. Holen Sie sich drei TextAreas aus der Werkzeugkiste . Dann noch zwei Buttons. Benennen Sie die Buttons bei den Properties (= Eigenschaften mit F4 Taste aufrufen) um.

Siehe <http://www.madeasy.de/7/prgtxtplus.htm>

Der Programmcode schaut so aus:

```
PUBLIC SUB Button1_Click()  
  TextArea1.Text = "1.Teil "  
  TextArea2.Text = "2.Teil "  
END  
  
PUBLIC SUB Button2_Click()  
  TextArea3.Text = "1.Teil " & " und 2.Teil "  
END
```

Auch folgender Code ist korrekt:

```
TextArea3.Text = TextArea1.Text & TextArea2.Text
```

*TextArea3.Text = TextArea1.Text ' +TextArea2.Text* gibt eine Meldung True oder T.

## 4.69 Neue Zeile mit & Chr\$(10) & Chr\$(13)



Die Zeichen Chr\$(10), Chr\$(13) haben eine besondere Bedeutung. Sie symbolisieren keinen Buchstaben wie zb Chr\$(60) sondern führen einen Zeilenwechsel durch. Die alten Bezeichnungen dafür waren LF und CR, d.h Line feed und Carriage return. Sie stammen noch aus der Schreibmaschinenarea.

Beachten Sie das in Gambas das Pluszeichen + nur als mathematisches Zeichen verwendet werden darf. Zum kleben von Strings benutzt man **&**.

Das Beispiel Programm klebt 2 Textteile aneinander und fügt eine neue Zeile dazwischen. Das Ergebnis wird in einer dritten TextArea ausgegeben.

Holen Sie sich drei TextAreas aus der Werkzeugkiste . Dann noch zwei Buttons. Benennen Sie die Buttons bei den Properties ( = Eigenschaften mit F4 Taste aufrufen) um.

<http://www.madeasy.de/7/prglinefeed.htm>

Der Programmcode schaut so aus:

```
PUBLIC SUB Button1_Click()  
  TextArea1.Text = "1.Teil "  
  TextArea2.Text = "2.Teil "  
END  
  
PUBLIC SUB Button2_Click()  
  TextArea3.Text = "1.Teil " & Chr$(10) & Chr$(13) & " und 2.Teil "  
END
```

## 4.70 Textlabel Fett, Kursiv und farbig

Die Textausgabe in einem Textlabel kann man kursiv,fett oder farbig gestalten. Wie das geht zeigt folgendes Programm.

Holen Sie sich einen TextLabel aus der Werkzeugkiste . Dann noch einen Button. Benennen Sie den Button bei den Properties ( = Eigenschaften mit F4 Taste aufrufen) um.

<http://www.madeasy.de/7/prgtextlabel.htm>

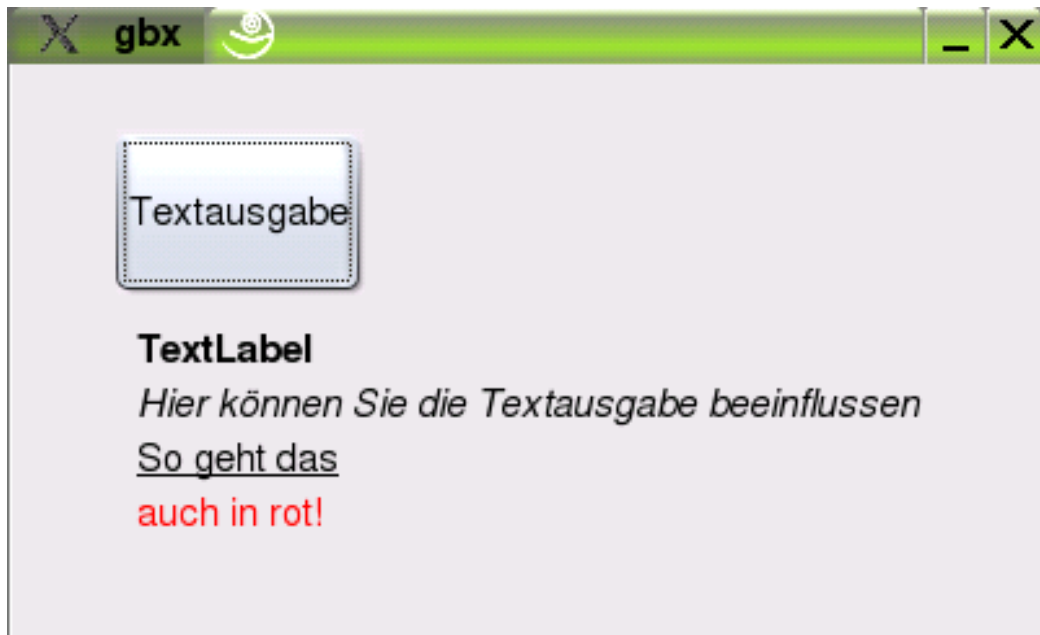


Abbildung 68: Screenshot des Textlabel Programmes

Clicken Sie auf den Button im Entwurfsmodus und geben sie den Programmcode von unten ein. Leider wird unter der Wikieingabe der Quelltext falsch dargestellt. Drücken Sie deswegen in diesem Wikiabsatz rechts oben auf *Bearbeiten* und übernehmen Sie den Wikiquelltext des Programmes direkt aus der Bearbeitenversion. Dann funktioniert es richtig.

```
PUBLIC SUB Button1_Click()  
    "TextLabel1.Text= "TextLabel<br>"&  
    "Hier können Sie die Textausgabe beeinflussen <br>"&  
    "So geht das<br>"&  
    "auch in rot!"  
END
```

## 4.71 Der Split Befehl

Der Befehl Split unterteilt einen Textstring in mehrere Teile. Die Textausgabe wird durch ein Separatorzeichen getrennt. Das Separatorzeichen ist auf ein Komma voreingestellt. Will man ein anderes Separatorzeichen muss man dies angeben. In dem angegebenen Beispiel wird als Separatorzeichen ein Leerzeichen verwendet.

### 4.71.1 Einen Satz in seine Worte zerlegen

Mit dem Splitbefehl kann man sehr einfach eine Satz in seine Worte zerlegen:

Beispielprogramm im Terminalmodus

```

STATIC PUBLIC SUB Main()
DIM liste AS String[]
DIM element AS String
DIM trenn AS String
DIM text1 AS String
trenn = " "
text1 = "Dies ist ein Satz mit mehreren Worten."
liste = Split(text1, trenn)
FOR EACH element IN liste
PRINT element
NEXT
END

```

Ausgabe im Direktfenster:

```

Dies
ist
ein
Satz
mit
mehreren
Worten.

```

### 4.71.2 Beispiel2: Eine Zahlreihe summieren

Siehe Gambas Rechnen Summierprogramm Addy

### 4.71.3 Beispiel 3 des Splitbefehls

Das Beispielprogramm zeigt Ihnen den Gebrauch des Splitbefehls .

Das hier angebene Beispiel ist etwas kompliziert , da in der Ausgabe die Reihenfolge der Wort noch von hinten nach vorne vertauscht wird.

Holen Sie sich eine neue Form. Platzieren Sie darauf ein Textfeld ( = Textbox). (Holen Sie sich das Element mit F6 und Doppelclick aus der Werkzeugkiste.)

Platzieren Sie darauf ein Befehlsfeld und benennen Sie es neu, wie auf dem Bild gezeigt :

Holen Sie aus der Werkzeugkiste zwei Textlabelfelder.

Programmcode:

```
STATIC PUBLIC SUB Main()
hForm AS F
hForm = NEW F
hForm.show
END
PUBLIC SUB Button1_Click()
myAr AS String[]
myStr AS String
outstr AS String
x AS Integer
outstr = ""
myAr = Split(TextBox1.Text, " ")
FOR x = 1 TO myAr.Length
    ostr = ostr & " " & myAr[myAr.Length - x]
NEXT
TextLabel2.Text = ostr
END
```

## Split Syntax

```
Array = Split ( String [ , Separators , Escape ] )
```

Splits a string into substrings delimited by `Separators`. Escape characters can be specified: any separator characters enclosed between two escape characters are ignored in the splitting process.

Note that `Split` takes only three arguments: if you want to use several separators, you should pass them as the second parameter, concatenated in a single string.

By default, the comma character is the separator, and there are no escape characters.

This function returns a string array filled with each detected substring.

## Beispiel

```
DIM Elt AS String[]
DIM Sb AS String
Elt = Split("Gambas Almost Means BASIC ! 'agree ?", " ", "'")
FOR EACH Sb IN Elt
    PRINT Sb
NEXT
```

## Ausgabe:

```
Gambas
Almost
Means
BASIC
!
agree ?
```

## 4.72 Der Replace Befehl - Zeichenersetzen

Mit dem Replace\$ Befehl kann man Buchstaben oder Silben in einem Text austauschen.

Der Befehl hat folgende Syntax:

```
Ergebnis = Replace$ ( Text , Suchmuster , Ersatzmuster )
```

Der Befehl tauscht an jeder Stelle im Text die Silbe Suchmuster durch das neue Ersatzmuster aus. Das ganze wird in Ergebnis gespeichert. Beachten Sie Ergebnis, Text, Suchmuster, Ersatzmuster sind willkürlich gewählte Namen Falls Text ein leerer Text ist, dann ist *Ergebnis* auch Null. Falls Suchmuster leer ist (null), dann ist Ergebnis = Text.

### 4.72.1 Kleine Beispiele aus der Gambas Hilfe

```
PRINT Replace$("Gambas is basic", "bas", "BAS")
```

» GambAS is BASic

```
PRINT Replace$("Gambas is basic", "a", "")
```

»Gmbs is bsic

```
PRINT Replace$("Gambas is basic", " ", "--")
```

»Gambas–is–basic

### 4.72.2 Beispielprogramm

Im folgenden Beispiel kann man die Austauschaktion beobachten und nach belieben verändern.

### 4.72.3 Layout

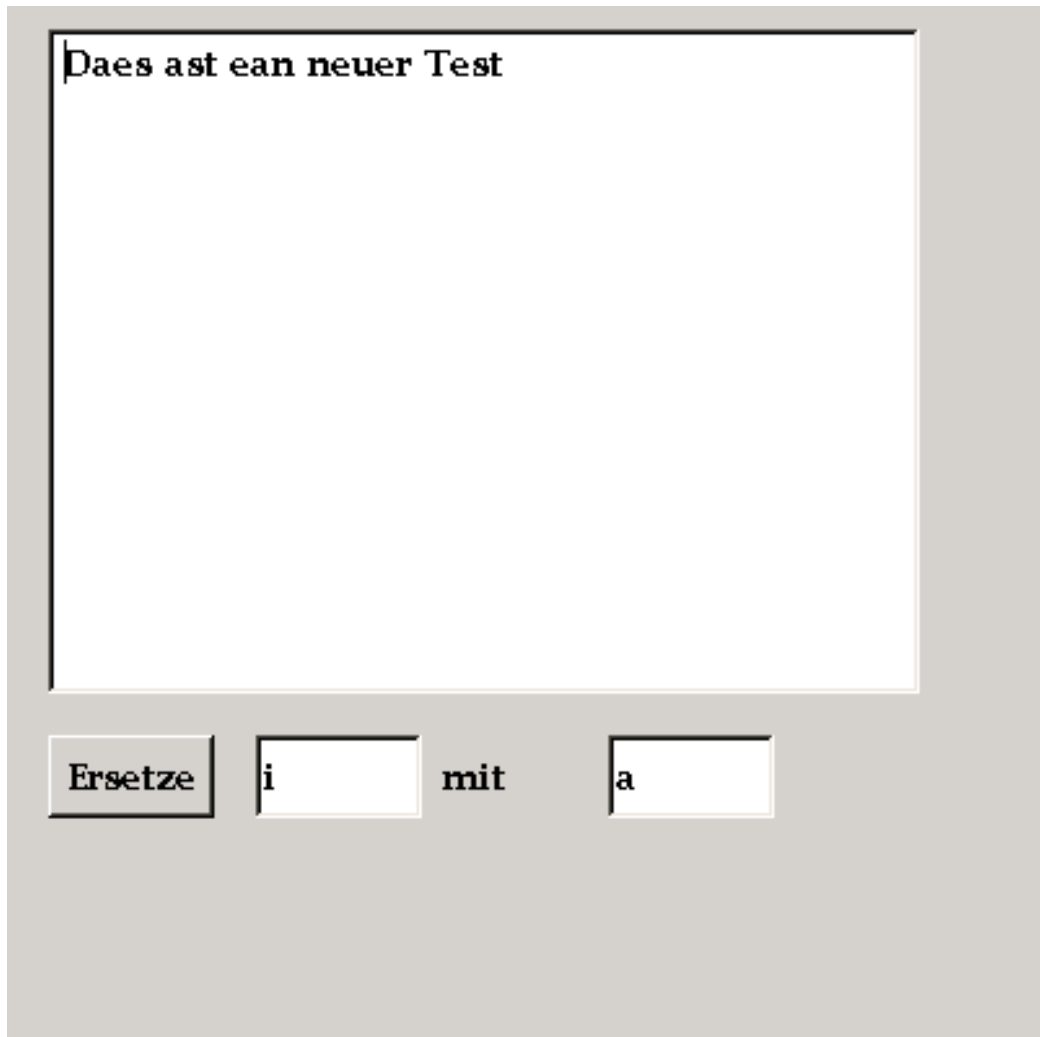


Abbildung 69: Bild:Gambasreplace.png

Man braucht:

- 1 Textarea
- 1 Befehlsbutton
- 2 Textboxen
- 1 Label

## 4.72.4 Der Code

```

PUBLIC SUB Form_Open()
    ME.Text = "Replace-Befehl"
    Textareal.Text = "Dies ist ein neuer Test"
    Textbox1.Text = "i"
    Textbox2.Text = "a"
    Labell.Text = "mit"
    Button1.Text = "Ersetze"
END
PUBLIC SUB Button1_Click()
    a AS String
    a = Replace$(Textareal.Text, Textbox1.text, Textbox2.text)
    textareal.Text = a
END

```

## 4.73 Instr An welcher Position steht der Teil ?

Der Befehl *InStr* sucht eine kurze Zeichenfolge (String) in einer längeren Zeichenfolge.

```
print Instr("bearbeitet", "ea")
```

Ergibt in der Ausgabe:

```
2
```

Die Textfolge *ea* wird ab der Position 2 in *bearbeitet* gefunden: *bearbeitet*.

Das kurze Programm

```

STATIC PUBLIC SUB Main()
    PRINT Instr("Dieser Text wird jetzt gleich bearbeitet", "ei")
    PRINT Instr("Dieser Text wird jetzt gleich bearbeitet", "ei", 28)
    PRINT Instr("Diesen String wird jetzt gleich bearbeitet ", "ein")
END

```

ergibt als Ergebnis in der Direktausgabe:

```
26
36
0
```

- 26 ist die erste Fundstelle.
- 36 ist die Fundstelle nach der Position 28
  - Wenn man nach der zu suchenden Textsequenz nach einem Komma noch eine ganze Zahl N eingibt, dann fängt der Befehl erst ab der N.ten Stelle im Text zu suchen an.

*ein* gibt es gar nicht im Text , deswegen wird 0 ausgegeben.

## 4.74 RInStr

Der Befehl RInStr funktioniert wie sein Namensvetter InStr, nur sucht er vom rechten Ende her. (*von Rechts* deswegen **RInStr**)

Beispiel mit allen Möglichkeiten von RInStr

```
STATIC PUBLIC SUB Main()  
PRINT RInStr("Dieser Text wird jetzt gleich bearbeitet", "ei")  
PRINT RInStr("Dieser Text wird jetzt gleich bearbeitet", "ei", 28)  
PRINT RInStr("Dieser Text wird jetzt gleich bearbeitet", "ein")  
END
```

Das gleiche Beispiel im Grafikmodus:

```
PUBLIC SUB Button1_Click()  
PRINT RInStr("Dieser Text wird jetzt gleich bearbeitet", "ei")  
PRINT RInStr("Dieser Text wird jetzt gleich bearbeitet", "ei", 28)  
PRINT RInStr("Dieser Text wird jetzt gleich bearbeitet", "ein")  
END
```

Um das Beispiel in Gang zu bringen braucht man einen Befehlsbutton auf der Form.

Ausgabe:

```
36  
26  
0
```

## 4.75 Subst Platzhalter im Text

Mit Subst kann man einen variablen Platzhalter in einen Text einfügen. Der zu ersetzende Teil beginnt immer mit & und einer Zahl danach. (&1, &2, &3 ....usw.). Das Einfügen erfolgt der Reihe nach.

```
print Subst("Diesen Text in einer &1 bearbeiten", "Textbox")
```

&1 wird durch *Textbox* ersetzt.

```
print Subst("&1 Texte in der &2 ersetzen. ", "Alle", "Textbox")
```

Ausgabe:

```
Alle Texte in der Textbox ersetzen.
```



## 4.76 Länge eines Strings ( einer Textfolge) feststellen Len

Mit dem Befehl **Len** kann man die Länge eines Strings messen.

```
print Len("Die Länge wird gleich gemessen.")
```

Ausgabe:

```
32
```

Beispiel:

```
PUBLIC SUB Form_Open()
  Button1.Text = "Länge"
  TextBox1.Text = "Dies ist ein Test"
  TextBox2.Text = ""
END
PUBLIC SUB Button1_Click()
  textbox2.Text = Str(Len(textbox1.text))
END
```

Sie brauchen einen Befehlsbutton und 2 Textboxen auf ihrer Form, um das Beispiel in Gang zu bringen. Wenn Sie nicht wissen was der Befehl *str* bedeutet, dann schauen Sie hier nach: [Gambas: Umwandlung von Variablen](#) Str wandelt jede andere Variable in einen String um. Probieren Sie einfach einmal das Beispiel ohne Str aus. Auch das funktioniert.

## 4.77 CHR() ASCII-Zahlencode in Zeichen umwandeln

Der CHR Befehl wandelt Zahlen zu Buchstaben und Zeichen um

Syntax:

```
Character = Chr$ ( Zahl )
```

Er gibt das Zeichen zurück, welches dem ASCII code von Zahl entspricht . Auf das früher übliche Stringzeichen \$ können Sie verzichten.

```
Chr(Zahl) ist gleich Chr$(Zahl)
```

Das Gegenteil des CHR Befehls ist der Befehl ASC.

Beispiel 1:

```
PRINT Chr$(65)
```

Ausgabe: A

### Beispielprogramm zum CHR Befehl im Terminalmodus

```
STATIC PUBLIC SUB Main()
DIM x AS Integer
FOR x = 30 TO 133
PRINT x,Chr$(x)
NEXT
PRINT
PRINT "Zeilenumbruch CHR(10)", Chr$(10)
PRINT "Return CHR(13)", Chr$(13)
PRINT "Leerzeichen Chr$(32)", "a";Chr$(32);"a"
PRINT "0 CHR48", Chr$(48), Str(0)
PRINT "9 CHR57", Chr$(57), Str(9)
PRINT "@ CHR64", Chr$(64), Str("@")
PRINT "A CHR65", Chr$(65), Str("A")
PRINT "Z CHR90", Chr$(90), Str("Z")
PRINT "a CHR97", Chr$(97), Str("a")
PRINT "z CHR122", Chr$(122), Str("z")
END
```

## 4.78 ASC() Wandelt alphanumerische Zeichen in ASCII-Zahlencode um

Der Befehl Asc gibt den ASCII code eines Buchstabens oder Zeichens zurück. Wenn die Position nicht angegeben wird, dann wird der ASCII code des ersten Zeichens zurückgegeben.

### Syntax

```
Asc ( String [ , Position ] )
```

### Beispiel:

```
PRINT Asc("Gambas")
```

### Ausgabe: 71

```
PRINT Asc("Gambas", 3)
```

### Ausgabe: 109

### Beispiel im Terminalmodus:

```
STATIC PUBLIC SUB Main()
DIM test AS String
DIM x AS Integer
test = "1234567 Dies ist ein Test"
FOR x = 1 TO Len(test)
PRINT Asc(test,x)
NEXT
END
```

## 4.79 Str() wandelt andere Variablen in einen Textstring um

Haben Sie eine Zahl oder eine andere Variable, die Sie in Text umwandeln wollen, dann sollten Sie den Befehl Str() verwenden. Einfache Umwandlungen macht Gambas automatisch ohne den Str Befehl, verlassen Sie sich aber nicht auf diese Automatismen. Im Zweifelsfall fügen Sie ein Str ein.

Beispiel: Sie brauchen eine Textbox und einen Befehlsknopf, um es in Gang zu bringen.

```
PUBLIC SUB Form_Open()
  ME.Text = "Str wandelt andere Variablen in Text um"
  Button1.Text = "Start"
  textbox1.Text = ""
END
PUBLIC SUB Button1_Click()
  textbox1.Text = Str(8 * 37)
  'textbox1.Text = 8 * 37
  'Probieren Sie es einmal ohne Str
END
```

Siehe auch [Gambas: Umwandlung von Variablen#str\\$](#)

## 4.80 Zeichensatz

Folgendes kleine Programm gibt den aktuellen Zeichensatz Ihres Systems aus. Sie brauchen dazu nur eine leere Form.

```
PUBLIC SUB Form_Open()
  PRINT System.Charset
END
```

Ausgabe zb im Direktfenster:

```
ISO-8859-15
```

Siehe auch CONV\$

## 4.81 Schriftart (Font) über einen Standarddialog wählen

Im folgenden Beispielprogramm kann man die Schriftart einer Textarea über einen Standarddialog wählen.

Sie brauchen dazu eine Textarea und einen Commandbutton.

```
PUBLIC SUB Button1_Click()  
  IF Dialog.SelectFont() THEN RETURN  
  Textarea1.Font = Dialog.Font  
END
```

Sie können den Font natürlich auch über die Eigenschaften der Textarea festlegen oder im Programm direkt codieren.

Ein Beispielprogramm zeigt die direkte Codierung während der Laufzeit. Sie brauchen eine Form und einen Textarea, um das Programm in Gang zu bringen.

```
PUBLIC SUB Form_Open()  
  Form1.Font.Name = "Utopia"  
  Form1.Font.Bold = TRUE  
  Form1.Font.Italic = TRUE  
  Form1.Font.Size = "24"  
  Form1.Font.StrikeOut = FALSE  
  Form1.Font.Underline = TRUE  
END
```

## 4.82 Alle verfügbaren Schriftarten aufzählen

Mit dem folgenden Beispielprogramm können Sie sich alle verfügbaren Fonts aufzählen lassen. Sie brauchen einen Befehlsbutton um das Programm starten zu können. Die Ausgabe der Fonts erfolgt im Direktfenster.

```
PUBLIC SUB Button1_Click()  
  DIM c AS String  
  FOR EACH c IN Fonts  
    PRINT c  
  NEXT  
END
```

## 4.83 Text in die Zwischenablage

Die Zwischenablage ist eine der praktischsten Erfindungen seit es den PC gibt. Auch unter Gambas kann man sie nutzen.

Das folgende kleine Programm kopiert mit dem Befehl 1 einen Text in den Zwischenspeicher und holt ihn mit dem Befehl 2 wieder heraus.

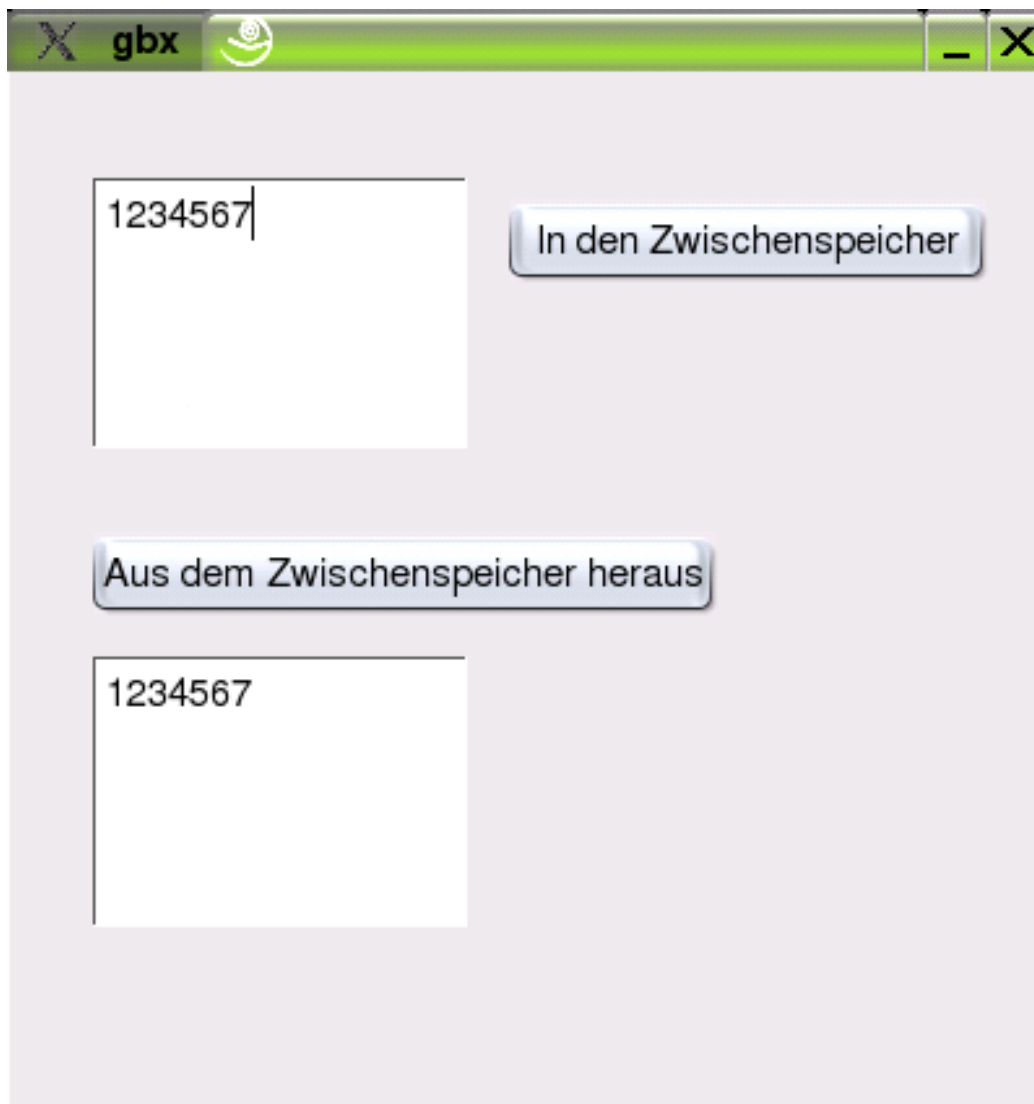


Abbildung 70: Bild:prgclipboard.png

Hinter den Befehlsfeldern *In den Zwischenspeicher* und *aus dem Zwischenspeicher heraus* steht folgender Programmcode:

```
PUBLIC SUB Button1_Click()  
DIM t AS String  
Clipboard.Clear  
' Zwischenablage löschen.  
t = textareal.Text  
Clipboard.Copy (t)  
'Text in Zwischenablage kopieren.  
textareal.Text = ""  
END
```

```
PUBLIC SUB Button2_Click()  
DIM a$ AS String  
'Text aus Zwischenablage holen.  
textarea2.Text = Clipboard.Paste ()  
END
```

#### Kürze Variante :

```
' Gambas class file  
PUBLIC SUB Button1_Click()  
Clipboard.Copy (Textareal.Text)  
'Text in Zwischenablage kopieren.  
END  
  
PUBLIC SUB Button2_Click()  
textarea2.Text = Clipboard.Paste ()  
'Wieder heraus holen  
END
```

## 4.84 Theorie der Zwischenablage

Die Clipboardklasse ist static. Was heißt das ??

Das Clipboard hat folgende Eigenschaften:

- Format
- Type

Es hat folgende Methoden

- Clear
- Copy
- Paste

Es hat folgende Konstanten

- Image
- None
- Text

You cannot instantiate this class. Referenced by: Graphical User Interface Component - gb.qt

## 4.85 Wie kann man Grafik in die Zwischenablage kopieren und wieder heraus holen ?

In Linux kann man einen Screenshot mit **Strg und der Drucktaste** in den Zwischenspeicher kopieren. Wie funktioniert das unter Gambas ??

## 4.86 Übersicht der Befehle

Date	Gibt das aktuelle Systemdatum zurück
Format	ermöglicht verschiedene Ausgaben für Zeit und Datum
IsDate	Gibt einen booleschen Wert zurück, der angibt, ob ein Ausdruck in ein Datum umgewandelt werden kann
Minute	Gibt eine ganze Zahl im Bereich von 0 bis 59 zurück, die die Minute in der Stunde darstellt
Month	Gibt eine ganze Zahl im Bereich von 0 bis 12 zurück, die den Monat im Jahr darstellt
Now	Gibt das aktuelle Datum und die aktuelle Zeit auf Ihrem Computer zurück
Time, Time\$	Gibt einen Wert vom Typ Date zurück, der die aktuelle Systemzeit angibt
Timer	Gibt die Anzahl der Sekunden zurück, die seit Programmbeginn vergangen sind
Weekday	Gibt den Wochentag als ganze Zahl zurück (1 = Sonntag, 2 = Montag, usw.)
Year	Gibt das Jahr als ganze Zahl zurück
Wait	Wartet eine vorgegebene Zeitspanne

## 4.87 Wait-Befehl (Pause)

Der Wait-Befehl hat folgende Syntax.

WAIT [ Pause ]

Er ruft die Event Schleife auf. Falls für *Pause* ein Wert eingegeben wurde, dann bleibt er in der Warteschleife bis die Pausenzeit vergangen ist. Pause ist eine Fließkommazahl. Wenn man eine warteschleife von 100 ms programmieren will, muss man

```
WAIT 0.1
```

eingeben.

Während der Waitschleife sind Tastatur- oder Mauseingaben nicht möglich. Nur Zeichnungs, Zeitgeber und Dateibeschreibungsereignisse wie Process\_Write laufen noch ab.

Beispiel:

Das folgende Beispiel zeigt den Gebrauch des Wait-Befehls. Man braucht dazu eine Form mit einem Befehlsbutton und einer Textbox.

```
PUBLIC SUB Button1_Click()  
    WAIT 5  
    'wartet 5 Sekunden  
    WAIT 0.1  
    'wartet 100 Millisekunden  
    textbox1.Text = ""  
END
```

## 4.88 Zeit

Wichtige Befehle für die Zeit sind **time** und **now**. Der Timer ist eine Stopuhr die immer wieder ein bestimmtes Zeitintervall durchläuft und dann ein Ereignis startet. Vorsicht: die Timerfunktion ist etwas anderes als der Timer. Die Timerfunktion misst die Zeit in Sekunden die seit dem Programmstart vergangen ist.

Einfache Umrechnungen sollten Sekunden in Minuten und Stunden und Tage umrechnen können. Das ganze auch wieder rückwärts.

### 4.88.1 Time Befehl

Ein gutes Beispiel für eine einfache Funktion ist der Befehl **Time** (= Zeit). Diese Funktion liefert die aktuelle Systemzeit. Sie können diese Funktion mit folgendem Befehl testen:



```
Print Time
```

Das folgende kleine Programm ohne Grafik gibt die Zeit im Direktfenster aus:

```
STATIC PUBLIC SUB Main()
  PRINT Time
END
```

Wenn Sie die Zeit immer wieder aktualisiert haben wollen, müssen Sie eine Schleife einbauen. Das Programm schaut dann so aus:

```
STATIC PUBLIC SUB Main()
  DIM i AS Integer
  FOR i = 1 TO 100
    PRINT Time
    WAIT 0.1
  NEXT
END
```

Das Programm gibt 10 Sekunden lang alle 100 Millisekunden die Zeit aus.

Ein ähnliches Beispiel für die Grafikausgabe aktualisiert die Zeitausgabe immer wieder. Dazu nutzt man den Timer, das **Zeitgeber-Steurelement**. Dieser hat zunächst mit der aktuellen Zeit nichts zu tun, sondern löst ein Ereignis nach einer bestimmten Zeit aus.

Das folgende kleine Beispielprogramm kombiniert den Timer und die Timefunktion zu einer ständig aktualisierten Zeitangabe, wie sie diese auch rechts unten an ihrem PC finden.

Starten Sie ein neues Projekt. Nennen Sie es Zeitausgabe. Platzieren Sie anschließend eine Textbox auf Ihrer Form. Außerdem brauchen Sie das Zeitgebersteuerelement mit dem Uhrensymbol. Holen Sie sich beides mit F6 und einem Doppelclick auf das Symbol aus der Werkzeugkiste. Achten Sie darauf, dass der Zeitgeber mit der **Enabled Eigenschaft** erst angeschaltet werden muss. Dies können Sie bei den Eigenschaften des Timers zur Entwurfszeit einstellen oder Sie können es auch im Programmcode mit der Zeile

```
Timer1.Enabled = TRUE
```

erreichen.

```
PUBLIC SUB Timer1_Timer()
  Timer1.Enabled = TRUE
  TextBox1.Text = Time
END
```

Das Programm zeigt in der Textbox die aktuelle Zeit an. Diese wird jede Sekunde (= 1000 Millisekunden) aktualisiert.

## 4.88.2 Timer-Funktion

Die Timer-Funktion hat nichts mit dem gleichnamigen Zeitgebersteuerelement zu tun.

Die Timer-Funktion gibt die Zeit in Sekunden aus, die seit Programmbeginn verstrichen sind.

In Visual Basic ist das anders. Hier gibt die Timer-Funktion die Anzahl der Sekunden zurück, die seit 0 Uhr verstrichen sind, mit einer Genauigkeit von 1/100 Sekunde.

Hinter dem Befehl *Zeit seit Programmstart* steht folgender Code:

```
PUBLIC SUB Button1_Click()  
    TextBox1.Text = Timer  
END
```

**Beispiel:**

```
PRINT Timer
```

**Ergebnis variabel**

```
0.291657006775
```

Ferner kann die Funktion dazu dienen, die Laufzeit eines Programms oder eines Programmteils zu ermitteln.

**Beispiel:**

```
Beginn = Timer  
'Schleife laufen lassen  
Ende = Timer  
Zeit = Ende - Beginn  
'Zeit wird in Sekunden gemessen
```

Sie können die Timer-Funktion zusammen mit der **Randomize-Anweisung** verwenden, um einen Startwert für die Rnd-Funktion zu erzeugen.

### 4.88.3 Format Befehl für die Zeit und Datumsausgabe

Der Format\$ Befehl ist sehr umfassend in seinen Möglichkeiten. Im folgenden soll nur die Zeit und Datumsfunktion betrachtet werden. Allgemein funktioniert er so:

```
String = Format$ ( Expression [ , Format ] )
```

Der Befehl Format wandelt einen Ausdruck (Expression) in einen Textstring um, je nach dem wie man es selber haben will.

Die Funktion nutzt ihre länderspezifischen Einstellungen des Rechners und muss deswegen immer wieder überprüft werden, insbesondere wenn man von einer Sprache in die andere wechselt.

Benutzerdefinierte Datumsformate kann man folgendermassen ausgeben:

yy	gibt die Jahreszahl in 2 Ziffern aus
----	--------------------------------------

yyyy	gibt die Jahreszahl in 4 Ziffern aus
m	gibt den Monat aus
mm	gibt den Monat in 2 Ziffern aus
mmm	gibt den Monat als abgekürzte Textform aus z.B. Aug = August
mmmm	gibt den Monat in voller Bezeichnung aus
d	gibt den Tag aus
dd	gibt den Tag in 2 Ziffern aus
ddd	gibt den Wochentag in abgekürzter Form aus
dddd	gibt den Wochentag in voller Länge aus
/	gibt den Datumstrenner (=separator) aus
h	gibt die Stunde aus
hh	gibt die Stunde in 2 Ziffern aus
n	gibt die Minuten aus
nn	gibt die Minuten in 2 Ziffern aus
s	gibt die Sekunden aus
ss	gibt die Sekunden in 2 Ziffern aus
:	gibt den Zeittrenner (= time separator) aus

Das zusammengefasste Beispielprogramm im Terminalmodus nutzt neben dem Formatbefehl noch die Funktion **Now** für den momentanen Zeitpunkt. Now = Jetzt.

```

STATIC PUBLIC SUB Main()
  PRINT Format$( Now, "mm/dd/yyyy hh:mm:ss" )
  PRINT Format$( Now, "m/d/yy h:m:s" )
  PRINT Format$( Now, "ddd dd mmm yyyy" )
  PRINT Format$( Now, "dddd dd mmmm yyyy" )
END

```

Die Ausgabe im Direktfenster mit deutscher Ländereinstellung schaut so aus:

```

11.12.2004 20:11:10
11.12.04 20:11:10
Fr 12 Nov 2004
Freitag 12 November 2004

```

Benutzerdefinierte Einzelbeispiele:

Befehl	Ausgabe
--------	---------

PRINT Format\$( Now, "mm/dd/yyyy hh:mm:ss" )	04/15/2002 09:05:36
PRINT Format\$( Now, "m/d/yy h:m:s" )	4/15/02 9:5:36
PRINT Format\$( Now, "ddd dd mmm yyyy" )	Mon Apr 15 2002
PRINT Format\$( Now, "dddd dd mmmm yyyy" )	Monday April 15 2002

#### 4.88.4 Problem mit Null Uhr Zeit rückwärts

Siehe dazu die [Anmerkung von InselAX](#).

Originaltext Mitteilung:

Ich habe auch noch ein anderes Problem gelöst. Es hat mit dem Datentyp "DATE" zu tun. Date ist nicht gleich Date! Das Problem trat auf als ich die aktuelle Uhrzeit, die in mein Eingabeformular übernommen wird und mit Hilfe eines Timers dort weiterläuft, (an Hand einer Konfigurationsdatei) auf GMT (Greenwich-Zeit; hier in Deutschland -1 Stunde bei Winterzeit; -2 Stunden bei Sommerzeit) umstellen wollte. Da hat mein Timer beim Wechsel von 23:59 auf 00:00 Uhr Ortszeit angefangen rückwärts zu laufen... warum auch immer...!?!?.

Die Lösung besteht darin, etwa folgendes zu schreiben:

```
DIM gmt AS Date
DIM ortszeit AS Date
DIM korrektur AS Date
```

```
ortszeit = Time(Now)
gmt = ortszeit - korrektur
```

(wobei korrektur Val("01:00:00") ist und 01:00:00 aus einer Datei kommt)

Wenn ich jetzt z.B. schreiben würde:

```
TextBoxZeit.Text = gmt
```

dann würde der Timer beim Wechsel rückwärts laufen und gegen 00:00 zählen, wobei er bei er bei 01:00:00 anfängt.

Problemlösung ist erst mal:

```
zeit = gmt + Val("00:00:00")
```

und dann

```
TextBoxZeit.Text = gmt
```

#### 4.88.5 Problem mit Greenwich Time

Siehe dazu die [Anmerkung von InselAX](#).

In einer GMT-TextBox läuft die Greenwich Zeit (hier in Deutschland Lokalzeit - 1 St.) Das funktionierte auch 23 Stunden lang. Für 1 Stunde von 00:00 Uhr bis 01:00 wurde zwar die korrekte Zeit in GMT angezeigt (23:00 bis 00:00), aber das Auslesen der Textbox brachte nur "00/00/00". Habe Benoit das Problem geschildert und er bat mich ihm meine Software zu schicken. Er schickte mir gestern einen Patch zurück....Ich hänge Dir ihn auch an. Es gab einen Bug in der Konvertierung von Float nach Date. Mit dem Patch funktioniert es. Du musst ihn in `./src/exec` (da wo Dein Gambas steckt....) kopieren und dann Gambas neu kompilieren.

Siehe: [gbx\\_value.c](#)

#### 4.88.6 Gambas-Buch > Zeit und Datum > "Problem mit Null Uhr Zeit rückwärts" und "Problem mit Greenwich Time"

Das Problem liegt an der Art, wie in Date das Datum und die Uhrzeit gespeichert werden.

Date ist ein Clone des Datentyps Double mit einem eingegrenzten Wertebereich von -657434 bis 2958465,99998843

Dabei wird der ganzzahlige Teil als Datum interpretiert und ermöglicht ein Datum von "01.01.100"- "31.12.9999". Der Nachkommateil wird als Zeit interpretiert, wobei folgende Formel verwendet wird:

$$(\text{Sekunden seit Mitternacht}) / (24*60*60)$$

Folglich gilt:

$$0 \leq \text{Zeit} < 1$$

Da die Zeit somit niemals negativ sein kann, wird ein negatives Vorzeichen ignoriert.

Wenn du jetzt Dein Programm im Gambas-Buch betrachtest, wird in der Festlegung der Ortszeit durch die Time-Funktion der Vorkommateil gelöscht. nach 0 Uhr erhält gmt einen negativen Wert (-1 Stunde) durch die Ignorierung des Vorzeichens wird dadurch 1 Uhr. In der folgenden Stunde (bis 0:59:59 Uhr) wird gmt mit Werten von -1 Stunde bis -1 Sekunde versorgt, die in "1:00:00" - "0:00:01" umgewandelt werden. Die Zeit läuft also nur scheinbar rückwärts.

```
DIM gmt AS Date
DIM ortszeit AS Date
DIM korrektur AS Date

ortszeit = Time(Now)
gmt = ortszeit - korrektur 'Korrektur="1:00:00"
```

Hier ist also ein Fehler im Programm! Du mußt dafür sorgen, daß gmt niemals negativ wird. Der Fehler liegt in der Position der Time-Funktion. Vergleiche dazu folgendes Programm:

```
DIM gmt AS Date
DIM heute AS Date
DIM korrektur AS Date

heute = Now
gmt = Time(heute - korrektur)
```

Hier wird der Vorkommteiler(Datum) erst nach Abzug der Korrektur gelöscht. Folglich wird gmt nicht negativ und die Zeitberchnung funktioniert richtig. Das Programm läst sich noch weiter vereinfachen:

```
DIM gmt AS Date
DIM korrektur AS Date

gmt = Time(Now - korrektur)
```

Übrigens kannst Du auf die gleiche Art das korrigierte Datum abfragen:

```
datum = Int(Now - korrektur)
```

Es liegt somit kein Bug vor, es ist nur eine Frage der richtigen Programmierung! (und der genauen Kenntnis des Datentyps Date)– [InselAX](#) 16:28, 15. Jun. 2008 (CEST)

## 4.89 Datum

Wichtige Befehle für das Datum sind **date**, **weekday**, **now**, **format** und **format\$**

Interessant sind folgende Fragen:

- Welcher Wochentag hat ein bestimmtes Datum? Siehe **weekday** Funktion
- Wieviel Tage liegen zwischen 2 verschieden Datumsangaben?

### 4.89.1 Heute

Das heutige Datum und die aktuelle Zeit erhält man mit dem Befehl **Now**.

**Beispiel:**

```
PRINT Now
```

**Ausgabe:**

```
05/16/2002 22:31:30
'so lang ist es schon her seit Benoit dies programmiert hat
```

Das folgende Miniprogramm gibt das heutige Datum in einer Textbox aus :

Siehe: [prgheute.png](#)

Hinter dem Befehl Heute steht folgender Code:

```
PUBLIC SUB Button1_Click()
    TextBox1.Text = "Heute ist der " & Str$(Now)
END
```

Anmerkung für VB6-Programme: In VB6 muß Str\$ durch CStr ersetzt werden – [InselAX 17:58](#), 17. Jun. 2008 (CEST)

**Beispiel 2:**

Der Befehl **Date** verkürzt die Zeit- und Datumsausgabe auf die reine Datumsausgabe. Probieren Sie dazu folgendes Programm. Sie brauchen nur eine leere Form, um es zu starten.

```
PUBLIC SUB Form_Open()
    PRINT Now
    PRINT Date(Now)
END
```

Da das Datum im Datentyp 'Date' als ganzzahliger Teil gespeichert wird kann man Date(Now) auch durch Int(Now) ersetzen.– [InselAX 17:59](#), 17. Jun. 2008 (CEST)

Anmerkung für VB6-Programme: In VB6 muß Date durch DateValue ersetzt werden – [InselAX 17:58](#), 17. Jun. 2008 (CEST)

## 4.89.2 Wochentag

**Beispielprogramm:**

Es gibt den Wochentag des heutigen Tages aus. Das Programm funktioniert im Terminalmodus:

```
STATIC PUBLIC SUB Main()
    PRINT WeekDay(Now)
    PRINT "Heute ist ";
    IF WeekDay(Now) = 1 THEN PRINT "Montag"
    IF WeekDay(Now) = 2 THEN PRINT "Dienstag"
    IF WeekDay(Now) = 3 THEN PRINT "Mittwoch"
    IF WeekDay(Now) = 4 THEN PRINT "Donnerstag"
    IF WeekDay(Now) = 5 THEN PRINT "Freitag"
```

```
IF WeekDay(Now) = 6 THEN PRINT "Samstag"
IF WeekDay(Now) = 7 THEN PRINT "Sonntag"
END
```

Das Programm nutzt den Befehl `Weekday` und den Befehl `Now` von Gambas.

Das Programm geht davon aus, daß der Montag der erste Tag der Woche ist. In VB6 hat `Weekday` einen weiteren (optionalen) Parameter, der den ersten Tag der Woche angibt (1=So 2=Mo 3=Di 4=Mi 5=Do 6=Fr 7=Sa), wobei Sonntag die Voreinstellung ist. Damit das Programm in VB6 korrekt funktioniert, muß jedes `'WeekDay(Now)'` durch `WeekDay(Now,2)` ersetzt werden.

Alternativ kann man den Wochentag auch mit der Format-Anweisung anzeigen. `Format(Now,"ddd")` für den auf 2 Buchstaben abgekürzten Wochentag und `Format(Now,"dddd")` für den ausgeschriebenen Wochentag.– [InselAX](#) 18:23, 17. Jun. 2008 (CEST)

### 4.89.3 Schaltjahr

Das folgende Beispielprogramm berechnet für eine Jahreseingabe, ob das Jahr ein Schaltjahr ist. In einem Schaltjahr gibt es einen 29. Februar.

Hinter dem Befehl **Ist das Jahr ein Schaltjahr?** steht folgender Code:

```
PUBLIC SUB Button1_Click()
    DIM leapyear AS String

    'keine Eingabe
    IF TextBox1.Text = "" THEN
        TextBox2.Text = "Es wurde keine Jahreszahl angegeben!"
        RETURN
    ENDIF

    'Eingabe keine Zahl oder Zahl mit Punkt (z.B. 1234.5)
    IF NOT IsInteger(Val(TextBox1.Text)) OR TextBox1.Text LIKE "*.*" THEN
        TextBox2.Text = "\" & TextBox1.Text & "\" ist keine Jahreszahl!"
        RETURN
    ENDIF

    IF Val(TextBox1.Text) MOD 400 = 0 THEN
        leapyear = " ist ein Schaltjahr."
    ELSE IF Val(TextBox1.Text) MOD 100 = 0 THEN
        leapyear = " ist eine Jahrhundertwende, aber kein Schaltjahr."
    ELSE IF Val(TextBox1.Text) MOD 4 = 0 THEN
        leapyear = " ist ein Schaltjahr."
    ELSE
        leapyear = " ist kein Schaltjahr."
    ENDIF

    TextBox2.Text = "Das Jahr " & Val(TextBox1.Text) & leapyear
END
```



---

Wikipedia bietet weiterführende Informationen zum Thema [Schaltjahr](#).

#### 4.89.4 IsDate

Die **IsDate**-Funktion liefert einen Wert TRUE oder FALSE, der anzeigt, ob eine Variable vom Datumstyp ist als ein gültiges Datum interpretiert werden kann. Damit ist beispielsweise auch eine Überprüfung auf Schaltjahr möglich.

```
IsDate("29.2.2008")=True
```

```
IsDate("29.2.2009")=False
```

– [InselAX](#) 18:46, 17. Jun. 2008 (CEST)

Syntax: IsDate(Variant)

#### 4.89.5 CDate

Die **CDate**-Funktion wandelt einen Ausdruck in einen Datums-Datentyp um.

Syntax: CDate(Ausdruck)

Das Argument Ausdruck muss ein Zeichenfolgenausdruck oder ein numerischer Ausdruck sein, der als Datum interpretiert werden kann. Erlaubte Werte für Dateninformationen sind die Werte vom 1. Januar 100 (-657434) bis zum 31. Dezember 9999 (2958465). Der Tag 0 ist dabei der 30.12.1899. Dieser Tag ist mit CDate nicht zu erreichen, da CDate 0 als Uhrzeit ohne Datum interpretiert.– [InselAX](#) 19:11, 17. Jun. 2008 (CEST)

#### 4.89.6 Year

Die **Year**-Funktion liefert eine Ganzzahl (Integer) aus dem Bereich von 100 bis 9999, die das Jahr in einem als Argument eingegebenen Datum darstellt.

Syntax: Year(Datum)

Anmerkungen: Das Datum-Argument ist ein beliebiger numerischer oder Zeichenfolgen-Ausdruck, der ein Datum und/oder eine Uhrzeit vom 1. Januar 100 bis zum 31. Dezember 9999 darstellen kann, wobei der 1. Januar 1900 gleich 2 ist. Zahlen links vom Dezimalzeichen in Datum stehen für das Datum, Zahlen rechts davon stehen für die Uhrzeit. Negative Zahlen stehen für Datumsangaben vor dem 30. Dezember 1899.

Weist Datum den Wert Null auf, dann liefert die Year-Funktion auch den Wert Null.

## 4.90 Fragen

Gibt es in Gambas eine Befehl mit dem man das Datum verändern kann? (In Freebasic gibt es dazu den Befehl **SETDATE**: SETDATE month\$ + "-" + day\$ + "-" + year\$)

Gibt es in Gambas eine Befehl mit dem man die Uhrzeit verändern kann? (In Freebasic gibt es dazu den Befehl **SETTIME**)

Unter Linux gehören Datum und Uhrzeit zu den „hoheitlichen“ Aufgaben, nur Benutzer bzw. Programme mit root-Rechten können sie ändern. Entsprechende Gambas-Befehle wären also unter Linux nutzlos.

Gibt es einen Befehl zur Zeitmessung?

Siehe oben: [Timer-Funktion](#)

Zur Soundprogrammierung braucht man die Datei gb.sdl und natürlich eine funktionierende Soundkarte, die in Linux eingebunden ist. Siehe zB <http://www.binara.com/gambas-wiki/bin/view/Gambas/GbSdl> Siehe auch: <http://www.binara.com/gambas-wiki/bin/view/Gambas/FutureSDLComponent>

Zum Thema Sound in Gambas gibt es ein ganz gutes Beispiel : In den Beispielen die mit Gambas geliefert werden ist ein MusicPlayer im Quelltext dabei. zb

```
' Gambas class file

STATIC PRIVATE $bDoNotMove AS Boolean

PUBLIC SUB btnOpen_Click()

    IF Dialog.OpenFile() THEN RETURN

    lblTitle.Text = File.Name(Dialog.Path)
    ' Approximation...
    sldPos.MaxValue = 600

    Music.Load(Dialog.Path)
    btnPlay_Click

END

PUBLIC SUB btnPlay_Click()

    timMusic.Enabled = TRUE
    Music.Play
```

```
END

PUBLIC SUB btnPause_Click()

    Music.Pause

END

PUBLIC SUB btnStop_Click()

    Music.Stop
    timMusic.Enabled = FALSE
    lblPos.Text = ""
    sldPos.Value = 0

END

PUBLIC SUB timMusic_Timer()

    DIM iPos AS Integer
    DIM iVal AS Integer
    DIM sPos AS String
    DIM iInd AS Integer

    iPos = Music.Pos

    IF NOT $bDoNotMove THEN
        Object.Lock(sldPos)
        IF iPos > sldPos.MaxValue THEN
            sldPos.MaxValue = sldPos.MaxValue * 2
        ENDIF
        sldPos.Value = iPos
        Object.Unlock(sldPos)
    ENDIF

    FOR iInd = 0 TO 2

        iVal = iPos MOD 60
        iPos = iPos \ 60
        IF iInd THEN
            sPos = Format(iVal, "00") & ":" & sPos
        ELSE
            sPos = Format(iVal, "00")
        ENDIF

    NEXT

    lblPos.Text = sPos

END

PUBLIC SUB sldPos_Change()

    Music.Pos = sldPos.Value

END

PUBLIC SUB sldPos_MouseDown()

    $bDoNotMove = TRUE
```

```
END

PUBLIC SUB sldPos_MouseUp()

    $bDoNotMove = FALSE

END

PUBLIC SUB sldVolume_Change()

    Music.Volume = 1 - sldVolume.Value / sldVolume.MaxValue

END
```

Mit Gambas 2 können Sie Bilder von Webcameras oder TV Karten speichern und weiterbearbeiten. Voraussetzung ist, dass es sich um v4l (video for linux) Geräte handelt.

Die "VideoDevice" Klasse erlaubt die Steuerung des Gerätes, die Abspeicherung eines Bildes in einer Datei und sie unterstützt die **Gambas stream**-Schnittstelle mit ihren üblichen Befehlen (READ, SEEK, FLUSH usw.).

Siehe <http://gambas.gnulinex.org/v4l/>

## 4.91 Datei speichern

Im folgenden Beispielprogramm wird der Text einer Textarea in einer Datei auf der Festplatte gespeichert. Das Ganze erfolgt über den Standarddialog. Sie brauchen eine Textarea und einen Commandbutton um das Programm in Gang zu bringen.

**Beispiel:**

```
PUBLIC SUB Form_Open()
    Textareal.Text = "Dies ist ein Test"
END
PUBLIC SUB Button1_Click()
    Dialog.SaveFile()
    File.Save(Dialog.Path, Textareal.Text)
    CATCH
    IF ERROR THEN RETURN
END
```

## 4.92 Datei speichern ohne Dialog

Sie können eine Datei auch ohne Standarddialog speichern. Dann müssen Sie allerdings den Pfad und den Dateinamen im Programm angeben. Wie das geht, zeigt

folgendes Beispiel. Sie brauchen eine Textarea und einen Commandbutton um es in Gang zu bringen.

Programm:

```
PUBLIC SUB Form_Open()
Textareal.Text = "Dies ist ein Test"
END
PUBLIC SUB Button1_Click()
File.Save(system.Home & "/test.txt",TextAreal.Text)
CATCH
IF ERROR THEN RETURN
END
```

Mit dem Befehl *system.home* wird das aktuelle Benutzerverzeichnis gewählt zb */home/ihrname*.

Überprüfen Sie - am besten mit dem Konqueror oder mit der Konsole- , ob die Datei auch wirklich gespeichert wurde. Probieren Sie aus, was passiert wenn Sie die Fehleroutine mit Catch weglassen. ( Das macht solange nichts, solange die Pfadangabe korrekt ist. Ist die Pfadangabe falsch, dann gibt es eine Fehlermeldung: *File or directory does not exist.*) Probieren Sie aus, was passiert wenn die Datei, in die gespeichert werden soll, schon existiert. ( Die existierende Datei wird ohne Rücksicht auf Verluste überschrieben.) Probieren Sie aus, was passiert, wenn die Pfadangabe falsch ist. ( Es kommt zu keiner Fehlermeldung, solange die Fehleroutine mit Catch eingebaut ist !!)

Programmvariante: Nur ein Commandbutton ist nötig.

```
PUBLIC SUB Button1_Click()
File.Save(system.Home & "/test.txt","Dies ist ihr Text")
CATCH
IF ERROR THEN RETURN
END
```

## 4.93 Datei laden 1

Auf den meisten Linuxsystemen findet sich die Datei *passwd*. Diese wollen wir in eine Textarea laden

Dazu muß man wissen, wo die Datei steht: Sie steht im Verzeichnis: */etc*

Für das Programm brauchen Sie:

- eine Form
- eine Textarea

Code:

```
PUBLIC SUB Form_Open()  
s AS String  
s = File.Load("/etc/passwd")  
'Die Datei muss existieren,  
'sonst kann sie nicht geladen werden.  
textareal.Text = s  
END
```

Wie Sie sehen stehen in der Datei die Passwörter aller angemeldeten Benutzer Ihres Linuxsystems. Diese Datei ist lesbar aber schreibgeschützt.

## 4.94 Datei laden 2

Wenn Sie im Abschnitt Datei speichern die Datei test.txt in ihrem Homeverzeichnis gespeichert haben, können Sie sich diese wieder problemlos laden:

```
PUBLIC SUB Button1_Click()  
DIM sLine AS String  
sLine = File.Load(system.home & "/test.txt")  
PRINT sLine  
END
```

Weiteres Beispiel:

Sie haben eine Datei *Datei.text* in Ihrem Verzeichnis und wollen diese Datei in Gambas laden:

Wie das geht zeigt das folgende Beispiel:

```
PUBLIC SUB Button1_Click()  
DIM sLine AS String  
sLine = File.Load("Datei.text")  
'Die Datei muss existieren,  
'sonst kann sie nicht geladen werden.  
PRINT sLine  
END
```

Sie brauchen eine Form mit einem Commandbutton. Die Ausgabe erfolgt im Direktfenster. Um das Programm in Gang zu bringen, müssen Sie erst mit einem Editor die Datei *Datei.text* erstellen. Dann speichern Sie die Datei in ihrem Gambas Projektverzeichnis ab. Bei mir steht die Datei zb in dem Verzeichnis :

```
file:/home/rhoff/gambas-1.0/laden2/
```

## 4.95 Datei laden mit Standarddialog

Mit dem Standarddialog kann man Pfad und Dateiname beim Laden wählen.

Für das Beispiel brauchen Sie eine Form , einen Commandbutton, ein Labelfeld und eine Textarea.

```
PUBLIC SUB Button1_Click()  
  IF Dialog.OpenFile() THEN RETURN  
  Labell.Text = Dialog.Path  
  TextAreal.Text = File.Load(Dialog.Path)  
END
```

## 4.96 Datei laden und speichern mit Addyplus

In einem kleinen Anwendungsprogramm soll gezeigt werden wie man eine Zahlenkolonne laden und speichern kann. Dazu wird das kleine Summierprogramm Addy erweitert und zu Addyplus umfunktioniert.

Siehe auch [http://de.wikibooks.org/wiki/Gambas:\\_Rechnen#Addy\\_der\\_Summierer](http://de.wikibooks.org/wiki/Gambas:_Rechnen#Addy_der_Summierer)

So schaut die Form jetzt aus:

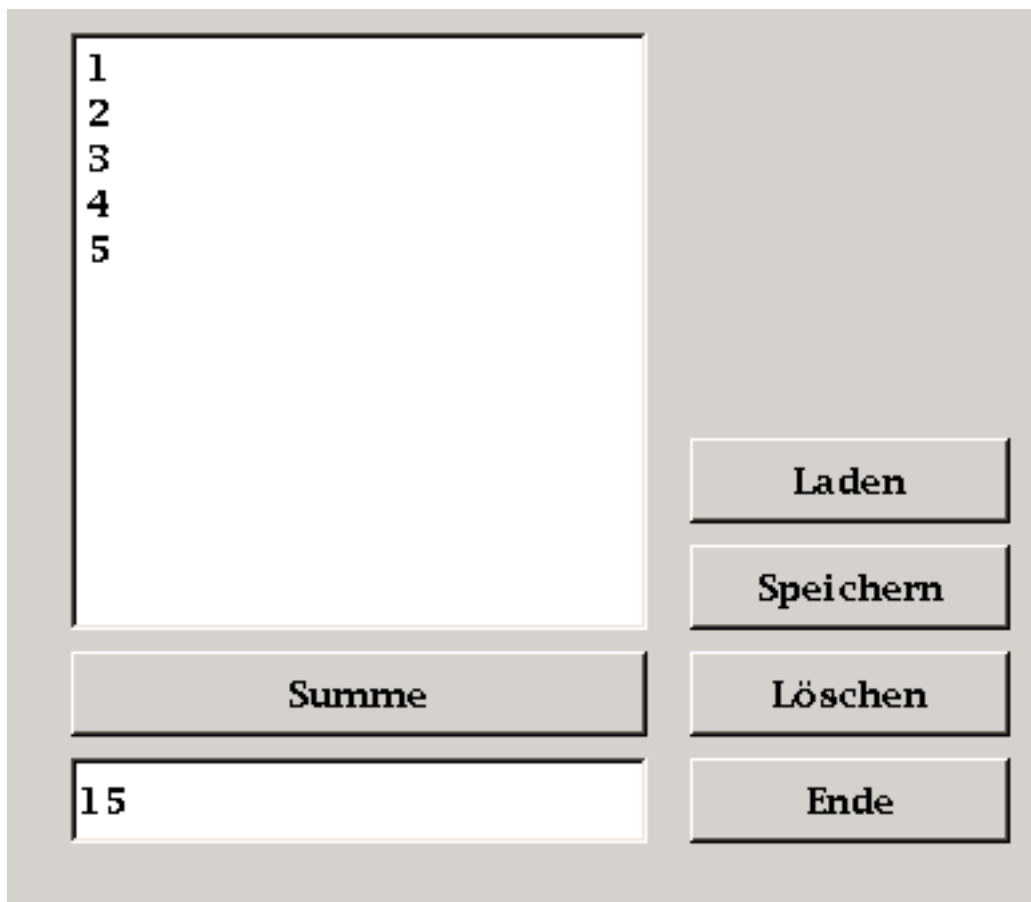


Abbildung 71: Bild:Gambasaddyplus.png

Sie brauchen also

- eine Textarea
- eine Textbox
- und 5 Befehlsknöpfe auf Ihrer Form, um das Programm zu starten.

So schaut der Code aus:

```
PUBLIC SUB Form_Open()
  ME.Text = "Addy, der Summierer"
  Textareal.Text = ""
  Textbox1.Text = ""
END
PUBLIC SUB Button1_Click()
  DIM text AS String
  DIM summe AS Float
  DIM liste AS String[]
  DIM posten AS String
  text = textareal.Text
  liste = Split(text,Chr(10))
  FOR EACH posten IN liste
    summe = summe + Val(posten)
  NEXT
  textbox1.Text = summe
  CATCH
  Message.Info("Bitte in der letzten Zeile kein Return eingeben ! Drücken Sie die Rücktaste!")
END
PUBLIC SUB Button3_Click()
  Dialog.SaveFile()
  File.Save(Dialog.Path, TextAreal.Text)
  CATCH
  IF ERROR THEN RETURN
END
PUBLIC SUB Button2_Click()
  IF Dialog.OpenFile() THEN RETURN
  TextAreal.Text = File.Load(Dialog.Path)
END
PUBLIC SUB Button4_Click()
  SELECT Message.Question("Vorsicht! Das Textfeld wird gelöscht.", "Abbrechen", "Löschen")
    CASE 1
      RETURN
    CASE 2
      Textareal.Text = ""
  END SELECT
END
PUBLIC SUB Button5_Click()
  ME.Close
END
```

Geben Sie in die Textarea irgendeine Zahlenreihe ein, beispielsweise

```
1
2
3
```

Dann drücken Sie auf Speichern. Geben Sie einen Dateiname ein zb *s1* und speichern sie die erste Zahlenreihe ab.



## 4.97 Dir Befehl

File name array = Dir ( Directory [ , File pattern ] )

Dieser Befehl gibt einen String Array zurück, der die Namen der Dateien des Verzeichnisses enthält, welches mit *File pattern* übereinstimmt. Falls kein *File pattern* spezifiziert wurde, dann werden alle Dateien ins Array geladen. File pattern kann dieselben allgemeinen Zeichen enthalten wie der LIKE Befehl.

Beispiel:

```
PUBLIC SUB Form_Open()
DIM File AS String
FOR EACH File IN Dir(system.home, "**")
  PRINT File
NEXT
END
```

In diesem Beispiel werden alle Dateien im Home Verzeichnis des aktuellen Benutzers im Direktfenster ausgegeben. Dabei werden allerdings auch die Unterverzeichnisse als Dateien gehandhabt.

Beispiel

```
' Print a directory
SUB PrintDirectory(Directory AS String)
  File AS String
  FOR EACH File IN Dir(Directory, "**.*")
    PRINT File
  NEXT
END
```

Beispielvariante : Sie brauchen eine Form mit einem Button auf der Form. Fügen Sie in das Programm statt */home/rho/gambastext/* ein existierendes Verzeichnis auf Ihrem PC ein.

```
' Gambas class file
PUBLIC SUB Button1_Click()
  PrintDirectory ("/home/rho/gambastext/")
END
SUB PrintDirectory(Directory AS String)
  DIM File AS String
  FOR EACH File IN Dir(Directory, "**.*")
    PRINT File
  NEXT
END
```

## 4.98 Neues Verzeichnis anlegen

Mit dem Befehl **MakeDir** kann man ein neues Verzeichnis anlegen. Vorsicht: Es klappt nur,

- wenn Sie einen korrekten Pfad eingegeben haben,
- wenn Sie die Schreibberechtigung für diesen Pfad haben
- wenn Sie nicht in das Verzeichnis des offenen Gambasprojektes ein neues Verzeichnis schreiben wollen. ( siehe Application.Name)
- keine Datei gleichen Namens in diesem Pfad existiert.

Beispiel:

Sie brauchen einen Befehlsbutton, um das Programm in Gang zu bringen:

```
PUBLIC SUB Button1_Click()  
  MKDIR system.home & "/test"  
END
```

system.home gibt dabei das aktuelle Nutzerverzeichnis /home/zbirname an.

Schauen Sie mit dem Konqueror nach, ob das Verzeichnis wirklich angelegt wurde.

Am besten fügt man eine Fehlerroutine ein, für den Fall das der Pfad schon existiert oder kein Schreibzugriff besteht.

## 4.99 Datei kopieren

Der **COPY** Befehl kopiert eine Datei von einem Verzeichnis in ein anderes.

**COPY** Quellverzeichnis/Datei TO Zielverzeichnis/Datei

Obiger Befehl kopiert eine Datei aus dem Quellverzeichnis in das Zielverzeichnis. Dabei braucht die Datei nicht unbedingt denselben Namen zu haben, wie die Zielfile. Unterverzeichnisse werden dabei allerdings nicht kopiert. (Note that you cannot copy directories recursively with this function. )

Beispiel:

```
' Sichert die versteckte Gambaskonfigurationsdatei  
' Das Verzeichnis muß vorhanden sein  
' und man muß den Zugriff darauf haben  
COPY system.name & "/.gambas/gambas.conf" TO system.name & "/gambas.conf.save"
```

System.name entspricht dabei ihrem Homeverzeichnis.

Das Beispiel ergibt eine Fehlermeldung, wenn :

- Das Verzeichnis nicht vorhanden ist

- Kein Schreibzugriff auf das Verzeichnis existiert
- Die Zieldatei schon vorhanden ist.

#### Beispiel2:

Sie brauchen einen Befehlsbutton auf Ihrer Form, um das Beispiel zu starten.

```
PUBLIC SUB Button1_Click()
  COPY "/home/ihrname/.gambas/gambas.conf" TO "/home/ihrname/test/gambas.conf.save"
END
```

Wenn man mehrere Dateien auf einmal kopieren will, hilft einem der *Dir* Befehl. Siehe [http://de.wikibooks.org/wiki/Gambas:\\_Datei](http://de.wikibooks.org/wiki/Gambas:_Datei)

Wie **Verzeichnisse kopiert oder verschoben** werden können ist hier [http://de.wikibooks.org/wiki/Gambas:\\_Fragen#Verzeichnis\\_kopieren](http://de.wikibooks.org/wiki/Gambas:_Fragen#Verzeichnis_kopieren) beschrieben

## 4.100 Wie kann man das Home Verzeichnis des aktuellen Benutzers abfragen ?

Der Befehl *Application.home* funktioniert dazu nicht mehr. Der Befehl *Application.Name* ergibt den Pfad des Programmes , welches gerade läuft, aber nicht das Homeverzeichnis.

Das Homeverzeichnis des aktuellen Benutzers erhält man mit dem Befehl **system.home** in Gambas 2 funktioniert auch dieser Befehl nicht mehr. Hiefür muss nun die 'User' Klasse hergenommen werden. Also lautet der Befehl: **user.home**

#### Beispiel1

```
PUBLIC SUB Form_Open()
  PRINT system.Home
END
```

#### Beispiel2:

```
PUBLIC SUB Form_Open()
  ME.Text = "Systeminfo"
  textareal.text = "Charset = " & system.Charset & Chr(13) & Chr(10)
  textareal.text = textareal.text & "Domain = " & system.Domain & Chr(13) & Chr(10)
  textareal.text = textareal.text & "Homeverzeichnis = " & system.Home & Chr(13) & Chr(10)
  textareal.text = textareal.text & "Host = " & system.Host & Chr(13) & Chr(10)
  textareal.text = textareal.text & "Sprache = " & system.Language & Chr(13) & Chr(10)
  textareal.text = textareal.text & "Pfad = " & system.path & Chr(13) & Chr(10)
  textareal.text = textareal.text & "Benutzer = " & system.User
  ' PRINT system.Charset
  ' PRINT system.Domain
  ' PRINT system.Home
  ' PRINT system.Host
```

```
' PRINT system.Language
' PRINT system.Path
' PRINT system.User
END
PUBLIC SUB Button1_Click()
    ME.Close
END
```

Man braucht eine Textarea und einen Befehlsbutton, um das Programm zu starten.

### Beispiel3

```
PUBLIC SUB Form_Open()
PRINT user.Home
END
```

## 4.101 Existiert eine Datei ?

Im folgenden Beispiel wird überprüft, ob die Bash vorhanden ist oder nicht.

Beispiel:

```
PUBLIC SUB Button1_Click()
    IF Exist("/bin/bash") = TRUE THEN
        Labell.Text="Bash ist vorhanden"
    END IF
    IF Exist("/bin/bash") = FALSE THEN
        Labell.Text="Fehler: Bash ist nicht vorhanden"
    END IF
END
```

Sie brauchen einen Befehlsbutton und ein Labelfeld, um das Programm in Gang zu bringen.

Ähnlich funktioniert:

```
GB.ExistFile(file_name)
```

## 4.102 Datei laden und speichern in einem kleinen kombinierten Programm

Im folgenden Programm wird eine Datei *test.txt* geladen und gespeichert.

So schaut das fertige Programm aus:

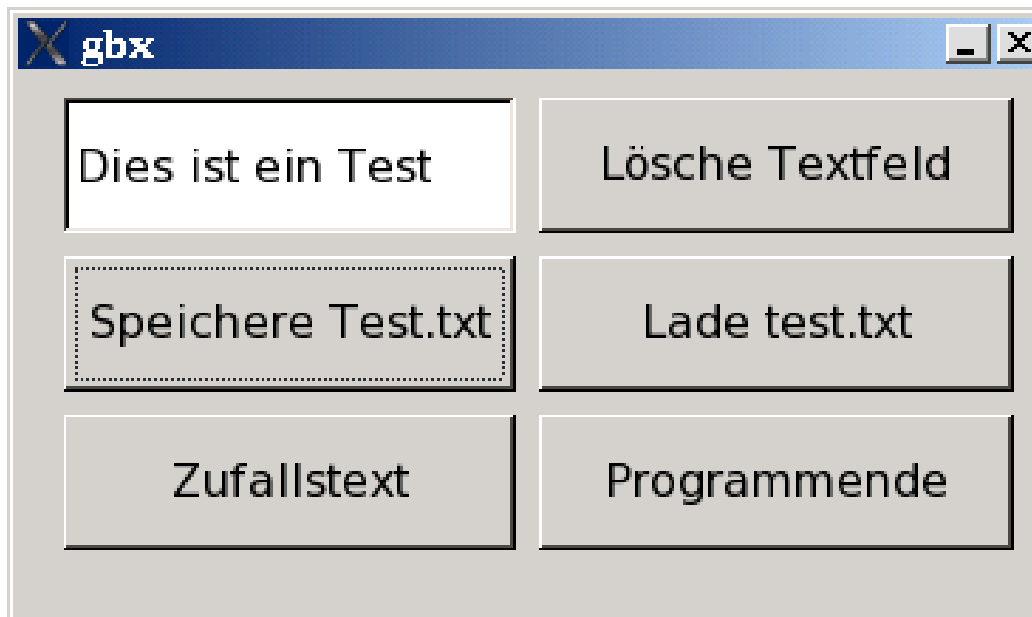


Abbildung 72: Bild:Gbdateiladen.png

Sie brauchen also:

- Eine Form
- Eine Textbox
- 5 Befehlsbuttons

Der Code schaut so aus:

```

PUBLIC SUB Form_Open()
    Textbox1.Text = "Dies ist ein Test"
END
PUBLIC SUB Button1_Click()
'Gespeichert wird im Dialog d.h es sind auch andere Dateinamen möglich
Dialog.SaveFile()
File.Save(Dialog.Path, TextBox1.Text)
'File.Save("test.txt", TextBox1.Text)
'test.txt muss schon vorhanden sein
'sonst gibt es eine Fehlermeldung.
END
PUBLIC SUB Button2_Click()
'Geladen wird ohne Dialog
IF Exist("test.txt") THEN Textbox1.Text = File.Load("test.txt")
'test.txt muss vorhanden sein!
END
PUBLIC SUB Button3_Click()
'Textbox wird gelöscht
    Textbox1.Text = ""
END
PUBLIC SUB Button4_Click()

```

```
'Programmende
  ME.Close
END
PUBLIC SUB Button5_Click()
'Zufallstext erzeugen
DIM Z AS Integer
DIM Buchstabe AS String
Randomize()
Textbox1.Text = ""
FOR Z = 1 TO 10
  Buchstabe = Chr$(Int(Rnd(65,91)))
  TextBox1.Text = TextBox1.Text & Buchstabe
NEXT
END
```

## 4.103 Datei schreiben mit WRITE

## 4.104 Daten einlesen mit READ

Der READ befehl liest Daten aus den verschiedensten Quellen in den Computer ein. Am einfachsten läßt er sich erklären, wenn man eine Datei von der Festplatte in die Anzeige lädt. Wie das geht zeigt folgendes Programm.

Sie brauchen dazu:

- eine Form
- einen Befehlsbutton

Der Code dazu lautet folgendermaßen:

```
PUBLIC SUB Form_Open()
  Button1.Text = "Read test.txt"
'Der Befehlsbutton1 wird umbenannt
File.Save(application.path & "/test.txt","Dies ist ein Test")
'Es wird eine Testdatei im aktuellen Verzeichnis angelegt und gespeichert
CATCH
'Mögliche Fehler abfangen
IF ERROR THEN RETURN
END
PUBLIC SUB Button1_Click()
'Die Testdatei wird bitweise eingelesen und wieder zusammengeklebt
hFile AS File
s AS String
OPEN "test.txt" FOR READ AS #hFile
WHILE NOT Eof(hFile)
  LINE INPUT #hFile, s
  PRINT s
'Ausgabe im Direktfenster
WEND
FINALLY ' Immer ausführen, auch bei einem Fehler
CLOSE #hFile
CATCH
```

```
' Nur bei einem Fehler ausführen
PRINT "Die Datei test.txt kann nicht ausgegeben werden."
END
```

Dieses Programm speichert einen eingegebenen Namen in einer Datei ab.

Beim nächsten Programmstart wird man dann mit diesem Namen begrüßt.

Sie benötigen dazu:

- 1 Form
- 2 Text Label
- 1 Eingabefeld
- 1 Commandbutton

```
' Gambas class file

PUBLIC SUB Button1_Click()

    settings["Name"] = Eingabefeld.Text
    Settings.Save

END

PUBLIC SUB Form_Open()

    TextLabel2.Text = "Hallo "&settings["Name", "No Name"]&"!"

END
```

So sieht das gestartete Programm dann aus:



Abbildung 73: Bild:Settings.png

## 4.105 Standarddialoge

Standarddialoge erleichtern die Programmierarbeit:

Im folgenden Beispielprogramm werden Standarddialoge für die Wahl

- der Schriftart
- der Farbe
- des Dateinamens
- des Dateipfades

benutzt.

Man braucht 1 Textarea und 3 Commandbuttons um das Programm in Gang zu bringen.



```

PUBLIC SUB Button1_Click()
  IF Dialog.SelectFont() THEN RETURN
  Textareal.Font = Dialog.Font
END
PUBLIC SUB Button2_Click()
  IF Dialog.SelectColor() THEN RETURN
  Textareal.BackColor = Dialog.Color
END
PUBLIC SUB Button3_Click()
Dialog.SaveFile()
File.Save(Dialog.Path, TextAreal.Text)
CATCH
IF ERROR THEN RETURN
END

```

Im folgenden Beispielprogramm wird der Standarddialog zur Verzeichnisauswahl verwendet. Das ausgewählte Verzeichnis wird in die Textbox gestellt. Es wird ein Button und eine Textbox verwendet.

```

PUBLIC SUB Button1_Click()
  Dialog.SelectDirectory()
  textbox1.Text = dialog.Path
CATCH
  IF ERROR THEN RETURN
END

```

## 4.106 Theorie

Diese Klasse **Dialog** enthält statische Methoden um Standarddialoge aufzurufen. Sie steht in der GB.QT Komponente.

Eigenschaften

```
Color Filter Font Path Title
```

Methoden

```
OpenFile SaveFile SelectColor SelectDirectory SelectFont
```

## 4.107 Größe eines Dialogwindows ändern

In einen Dialogfenster kann man die Größe des Fensters nicht einfach so ändern

```
dlgDialog.Height = neuerWert..
```

Dazu muß man folgenden Weg wählen: Window.Border zu Window.Resizable ändern. Dann kann man die Größe ändern. Dann muß man wieder Window.Border zu Window.Fixed zurückändern. Etwas umständlich, aber so funktioniert es.

Selbstverständlich ist es auch unter Gambas möglich, eine Menüführung zu programmieren, denn eine Befehlseingabe über Menüs macht ein Programm oft übersichtlicher und leichter bedienbar.

## **4.108 Erstes Menü**

Für unser erstes Menü beginnen wir mit einer leeren Form. Im Formfenster klicken wir mit der rechten Maustaste und wählen aus dem dort erscheinenden Kontextmenü den Punkt *Menüeditor...* oder drücken die Tastenkombination *Strg + E*.

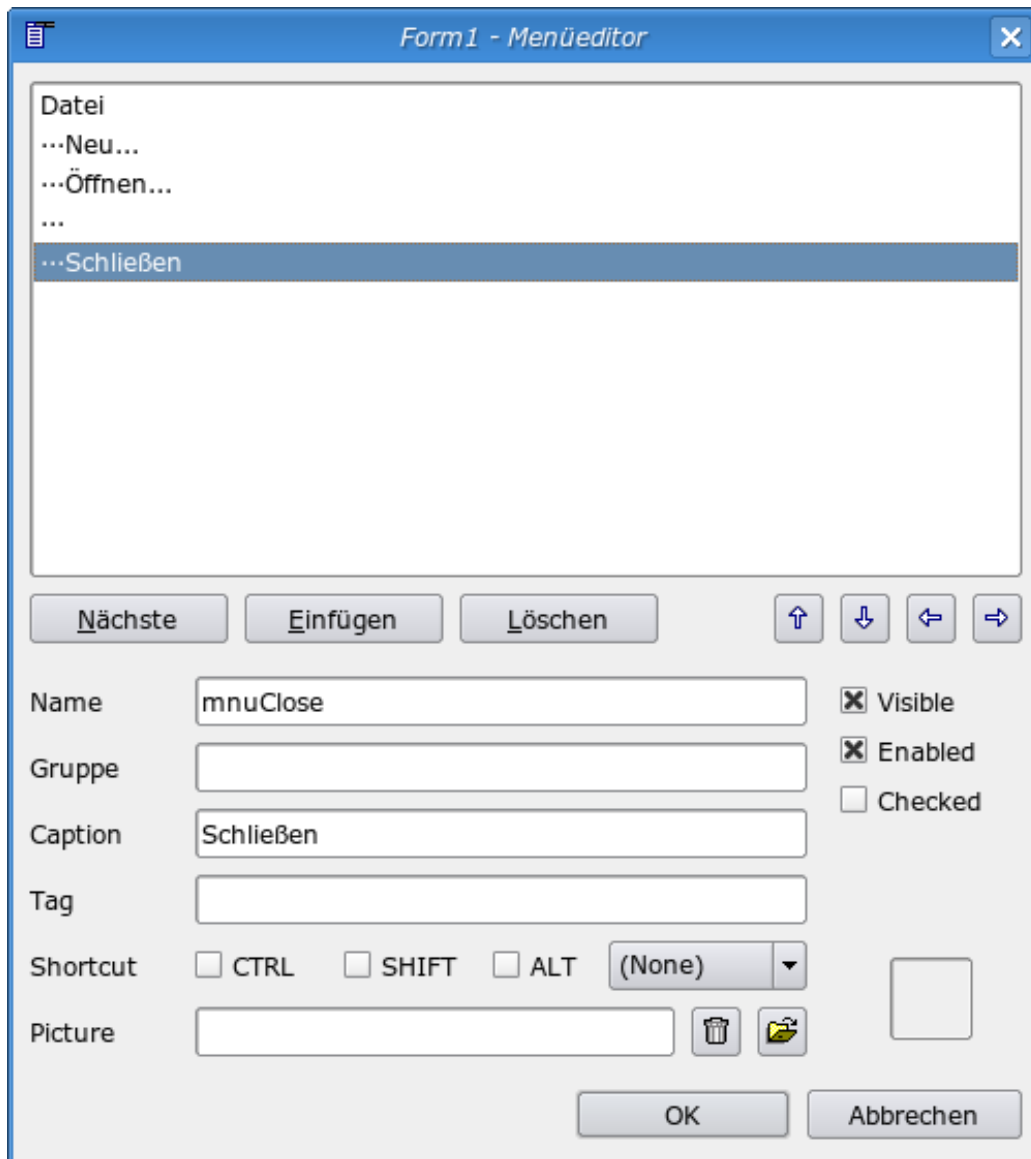


Abbildung 74: Der Gambas Menüeditor

- Ein Klick auf die Taste *Einfügen* erzeugt einen neuen Menüeintrag
- Der Menüeditor zeigt jetzt unter dem großen weißen Übersichtsfeld weiter unten mehrere Textboxen an (Name, Gruppe, Caption, Tag)
- Bei *Name* geben wir dem Menüpunkt einen Namen: **mnuFile** unter dem wir ihn innerhalb des Programms ansprechen können

- und bei *Caption* geben wir eine Beschriftung ein: **Datei**

Mit den Befehlsfeldern *Nächste* und *Einfügen* kann man jetzt jeweils neue Menüpunkte erzeugen. Wenn man dabei den Eintrag *Caption* leer lässt, wird statt eines Menüpunktes eine Trennlinie erzeugt.

### 4.108.1 Untermenüs

Jetzt müssen wir noch festlegen, dass die weiteren Menüpunkte keine eigenen Menüs sind, sondern Untermenüs des ersten Eintrags. Dazu drücken wir die Taste mit dem Pfeil nach rechts. Oben im Menüeditor werden jetzt drei Punkte ... vor der Bezeichnung des Eintrags angezeigt.

Durch mehrmaliges drücken dieser Taste können dann entsprechend auch Unteruntermenüs, Unterunteruntermenüs, usw. erzeugt werden.

Wenn wir dann alle fünf Menüeinträge angelegt haben, können wir den Menüeditor mit *OK* beenden.

### 4.108.2 Codeeingabe

Als letztes müssen wir noch den Programmcode eingeben, der beim Anklicken eines Menüpunktes ausgeführt werden soll. Das ist aber nicht weiter schwer, da sich ein angeklickter Menüeintrag wie ein angeklickter Button verhält.

Wir geben im Codefenster einfach den folgenden Code ein:

```
PUBLIC SUB mnuClose_Click()  
    ME.Close  
END
```

Jetzt können wir unser Programm mit F5 starten und das Menü testen. Beim Klick auf *Schließen* wird das Formular geschlossen.



Abbildung 75: Unser erstes Menü

## 4.109 Optionsmenü

Ein Optionsmenü ist ein Menüpunkt, der wie eine CheckBox zwischen zwei Zuständen hin und her geschaltet werden kann. Um dies in unserem Programm zu realisieren benutzen wir die *Checked*-Eigenschaft des Menüpunktes.

Nehmen wir an, wir haben ein Programm, das eine Reihe von Daten ausgibt und wir wollen dem Benutzer die Möglichkeit geben, sich die Daten auch sortiert ausgeben zu lassen. Dazu erzeugen wir im Menüeditor zunächst einen neuen Hauptmenüpunkt **mnuData** mit der Beschriftung **Daten** und einen Untermenüpunkt **mnuSorted** mit der Beschriftung **Sortiert**.

Schließlich benötigen wir noch den folgenden Programmcode:

```
PUBLIC SUB mnuSorted_Click()
  'Wert umschalten
  mnuSorted.Checked = NOT mnuSorted.Checked

  IF mnuSorted.Checked = TRUE THEN
    PRINT "Sortierung an"
  ELSE
    PRINT "Sortierung aus"
  ENDIF
END
```

Wird nun dieser Menüeintrag aufgerufen, dann schaltet der Befehl in der dritten Zeile den Wert der *Checked*-Eigenschaft jeweils um. In der fünften Zeile ist zu

sehen, wie wir den aktuellen Zustand des Menüpunktes im restlichen Programm auswerten können.



Abbildung 76: Ein Optionsmenü

## 4.110 Dynamisches Menü

Wenn man im Gambas-Projektfenster das Menü *Datei* und dann den Punkt *Zuletzt geöffnet* anwählt, erscheint ein Menü mit den Projekten, an denen man zuletzt gearbeitet hat. Aus offensichtlichen Gründen ist es nicht möglich, derartige Menüs schon zur Programmierzeit im Menüeditor zu erstellen, sondern sie müssen zur Laufzeit des Programms dynamisch erzeugt werden.

Dazu erzeugen wir im Menüeditor zunächst unterhalb des *Öffnen...*-Eintrags ein neues Untermenü **mnuRecent** mit der Beschriftung **Zuletzt geöffnet**. Dann ergänzen wir unser Programm noch um den folgenden Programmcode:

```
$aRecent AS NEW String[]

PUBLIC SUB Form_Open()
  'Beispieldaten
  $aRecent = ["Datei12", "Datei34", "Datei56", "Datei78", "Datei90"]

  'Menü erzeugen
  UpdateMenu
END

SUB UpdateMenu()
  DIM sFile AS String
```

```

DIM hMenuItem AS Object

'Die alten Einträge löschen
mnuRecent.Children.Clear

'Deaktivieren, falls keine Einträge vorhanden sind
IF $aRecent.Length = 0 THEN mnuRecent.Enabled = FALSE

'Die einzelnen Einträge erzeugen
FOR EACH sFile IN $aRecent
  'neuer Eintrag
  hMenuItem = NEW Menu(mnuRecent) AS "mnuRecentFile"

  'Beschriftung
  hMenuItem.Text = sFile
NEXT
END

PUBLIC SUB mnuRecentFile_Click()
  PRINT LAST.Text
END

```

Die von uns erzeugten Einträge haben dann alle den gleichen Namen **mnuRecentFile**. Damit wir innerhalb der Funktion *mnuRecentFile\_Click()* unterscheiden können welcher der Einträge angeklickt wurde, verwenden wir das *LAST*-Objekt. *LAST* ist eine Referenz auf das letzte Objekt, welches ein Ereignis (hier das *Click()*-Ereignis) ausgelöst hat. Mit *LAST.Text* greifen wir also auf die Text-Eigenschaft des zuletzt angeklickten Eintrags zu.

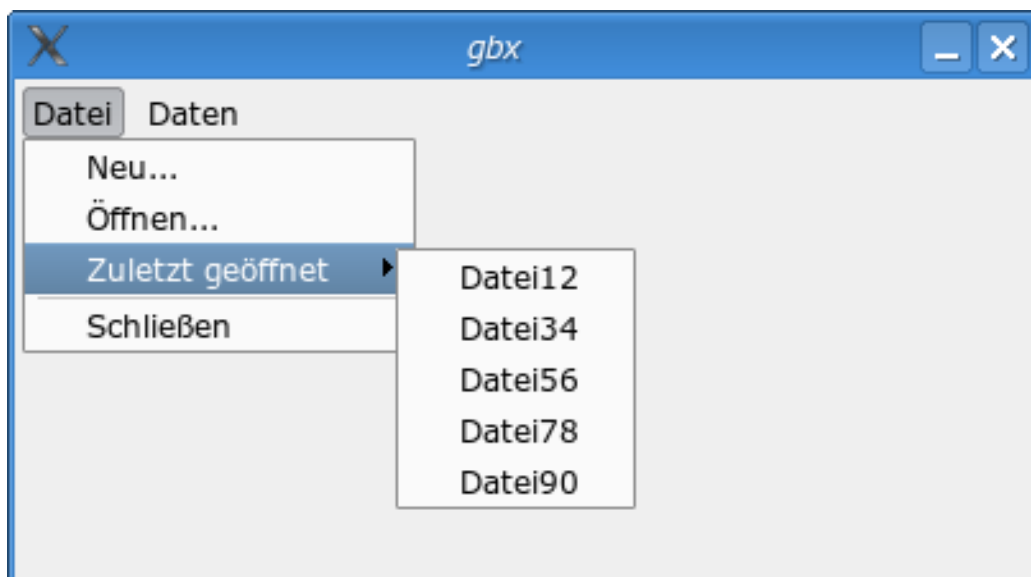


Abbildung 77: Ein dynamisches Menü

## 4.111 Kontextmenü

In vielen Programmen erscheint nach einem Klick mit der rechten Maustaste ein Kontextmenü. Im folgenden wollen wir in unserem Gambasprogramm ein Steuerelement mit einem einfachen Kontextmenü versehen.

Dazu platzieren wir auf der Form einen Button und ergänzen den folgenden Programmcode:

```
PUBLIC SUB Button1_MouseDown()  
    DIM hContext AS Object  
    DIM hMenuItem AS Object  
  
    IF Mouse.Right = TRUE THEN  
        'Das Kontextmenü selbst erzeugen  
        hContext = NEW Menu(ME) AS "mnuContext"  
  
        'Die einzelnen Menüeinträge erzeugen  
        hMenuItem = NEW Menu(hContext) AS "mnuRot"  
        hMenuItem.Text = "Rot"  
        hMenuItem = NEW Menu(hContext) AS "mnuBlau"  
        hMenuItem.Text = "Blau"  
        hMenuItem = NEW Menu(hContext) AS "mnuGruen"  
        hMenuItem.Text = "Grün"  
  
        'Das Menü anzeigen  
        hContext.Popup  
    ENDIF  
END  
  
PUBLIC SUB mnuRot_Click()  
    Button1.Background = Color.Red  
END  
  
PUBLIC SUB mnuBlau_Click()  
    Button1.Background = Color.Blue  
END  
  
PUBLIC SUB mnuGruen_Click()  
    Button1.Background = Color.Green  
END
```

Wenn man auf einem Steuerelement eine Maustaste drückt, wird die zugehörige *MouseDown()*-Funktion aufgerufen. Innerhalb derer müssen wir dann noch prüfen, ob die rechte Maustaste gedrückt wurde. Falls dem so ist, erzeugen wir das Menü und zeigen es an.



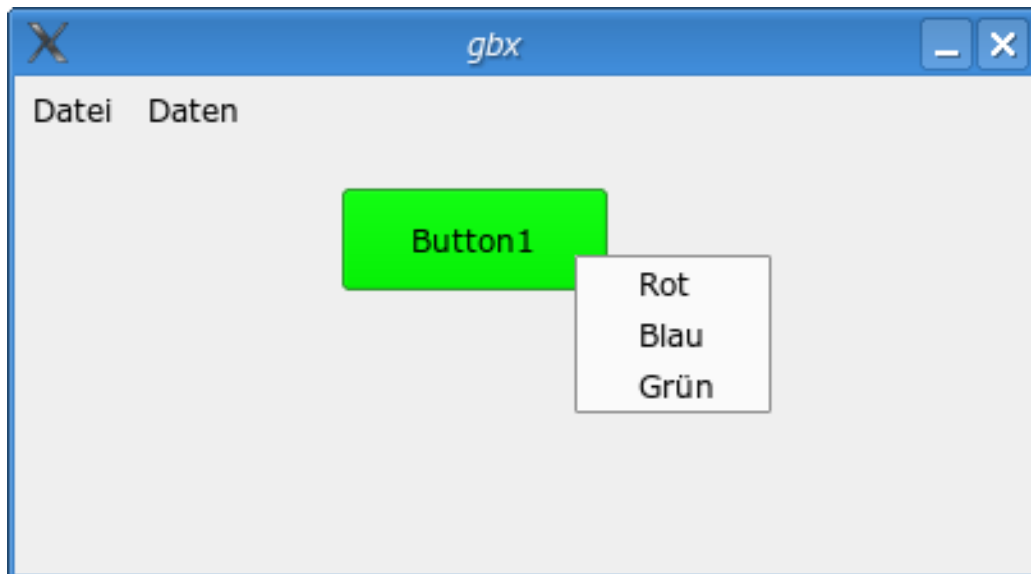


Abbildung 78: Ein Kontextmenü

**Vorsicht ! In diesem Abschnitt ist noch einiges in Arbeit !!**

## 4.112 Datenbankprogrammierung nur mit Gambas

Für das Anlegen einer simplen Adress- und Telefonnummerdatenbank brauchen Sie kein mysql. Probieren Sie es einfach Schritt für Schritt ohne mysql.

Fangen wir mit einer ganz simplen Tabelle an

```
1 Eins
2 Zwei
```

Sie soll abzuspeichern und zu verändern und zu verlängern sein. Sonst brauchen wir erst einmal keine weiteren Funktionen.

Das Ganze kann man als Stringarray machen und in einer Textarea darstellen.

Siehe [Gambas: Arrays](#)

Beispiel:

Sie brauchen eine Form eine Textarea und 2 Commandbuttons um das Programm in Gang zu bringen.

```
' Gambas class file
PUBLIC SUB Form_Open()
```

```
Textareal.Text = "1,Eins"  
Textareal.Text = Textareal.Text & "2,Zwei"  
END  
PUBLIC SUB Button2_Click()  
Dialog.SaveFile()  
File.Save(Dialog.Path, TextAreal.Text)  
'Geben Sie db.txt als Dateinamen ein  
CATCH  
IF ERROR THEN RETURN  
END  
PUBLIC SUB Button1_Click()  
DIM sLine AS String  
sLine = File.Load("db.txt")  
'Die Datei muss existieren,  
'sonst kann sie nicht geladen werden.  
textareal.Text = sLine  
END
```

Fügen Sie jetzt noch ein paar Tabellen Einträge dazu und speichern Sie wieder ab:

```
3,Drei  
4,Vier  
5,Fünf
```

Markieren Sie diese 3 weiteren datensätze und kopieren Sie sie mit Strg + C in den Zwischenspeicher. Mit Strg + V geben Sie sie wieder in Ihrer Textarea aus und speichern Sie ab.

Löschen Sie die Textarea und Laden Sie die Datenbank erneut. Siehe da, Ihre Minitabelle ist noch vorhanden.

Das ganze schaut immer noch wie eine Textdatei aus, aber auch eine richtige Tabelle wird zunächst als hintereinanderfolgende Datei abgespeichert. In unserer Minitabelle sind die einzelnen Datenfelder mit Kommas getrennt, die einzelnen Reihen (= Datensätze) werden durch ein Return (= Chr(10)) getrennt.

## 4.113 CSV Datei

Vielleicht wollen Sie obige einfache Zahlentabelle in ein Standard CSV Datenbankformat umwandeln, wie es zb Excel oder Openoffice als Importmöglichkeit nutzt.

Das CSV-Dateiformat wird oft benutzt, um Daten zwischen unterschiedlichen Computerprogrammen auszutauschen (es ist also ein Austauschformat). Die Verwendung des Formates in den von Microsoft Excel verwendeten Formen hat sich als Pseudostandard etabliert.

Das CSV Excel Format schaut so aus:

```
"1","Eins"  
"2","Zwei"
```

D.h. im Gegensatz zu unsere Darstellung sind die Eingaben in die Datenfelder noch mit einem Anführungszeichen versehen. Man muß jetzt nur aufpassen , das im Datenfeld selbst kein Anführungszeichen und kein Komma vorkommt, denn diese 2 Zeichen braucht man für die Tabellenstruktur.

Eine CSV-Datei ist eine Text-Datei, die tabellarisch strukturierte Daten enthält und vorrangig zum Datenaustausch verwendet wird. Das Kürzel CSV steht dabei für Character Separated Values oder Comma Separated Values, weil die einzelnen Werte durch ein spezielles Trennzeichen - in vielen Fällen das Komma - getrennt werden. Einzelne Datensätze werden in der Regel durch einen Zeilenumbruch (bei Windows: CR LF = carriage return, line feed; bei Unix: nur LF; bei Mac OS: nur CR) getrennt. Das CSV-Format ist unabhängig von Zeichencodierung, Byte-Reihenfolge und Zeilenumbruchszeichen.

CSV-Dateien tragen auch oft die Dateiendung .txt statt .csv und können auch in jedem Texteditor erstellt und bearbeitet werden.

Versuchen wir die CSV - Umwandlung zu programmieren: Dazu brauchen wir einen neuen Commandbutton den wir CSV nennen und folgenden Code:

```
PUBLIC SUB Button3_Click()
text AS String
liste AS String[]
posten AS String
reihe AS String
text = textareal.Text
liste = Split(text,Chr(10))
textareal.Text = ""
FOR EACH posten IN liste
IF posten = "" THEN
reihe = ""
ELSE
reihe = Chr(34) & posten & Chr(34) & Chr(10)
END IF
textareal.Text = textareal.Text & reihe
NEXT
```

Das schaut schon ganz schön aus. Vor allem wird die leere zeil am Schluß nicht mit Anführungszeichen versehen. Aber leider finden sich keine Anführungszeichen um die Kommas. Das Programm muß also noch verbessert werden:

Der String posten muß nach Kommas durchsucht werden und dann das Komma durch ”,“ ersetzt werden. das geht am besten mit dem Replace Befehl in Gambas. Schauen Sie mit der Hilfetaste F1 und dem Alphabetischen Verzeichnis der Befehle nach, wie er verwendet wird.

Für unsere Anwendung ersetzen wir noch die Anführungszeichen durch chr(34).

```
PUBLIC SUB Button3_Click()
text AS String
liste AS String[]
posten AS String
```

```
reihe AS String
text = textareal.Text
liste = Split(text,Chr(10))
textareal.Text = ""
FOR EACH posten IN liste
  IF posten = "" THEN
    reihe = ""
  ELSE
    reihe = Chr(34) & Replace$(posten, ",", Chr(34) & "," & Chr(34)) & Chr(34) & Chr(10)
  END IF
textareal.Text = textareal.Text & reihe
NEXT
```

Probieren Sie es aus. Speichern Sie Ihre Minitabelle beispielsweise als *dbl.csv* ab und versuchen Sie einmal, die Tabelle in Openoffice als Tabelle in der Tabellenkalkulation zu importieren. Klappt es ??

Verändern Sie die Tabelle in Open Office, speichern Sie diese wieder als CSV - Excel datei ab und versuchen Sie die Datei wieder in Ihr Programm zu holen, Klappt es ?

Das erscheint als überflüssige Spielerei, aber wer schon einmal versucht hat alte Datenbanken zu retten, der weiß wie wichtig Standarddatenformate und wie wichtig- Import und Exportmöglichkeiten sind.

## 4.114 Datenbankprogrammierung mit Gambas und MySQL

### 4.114.1 Vorarbeiten Mysql

Im Gegensatz zu MS Access ist Mysql zwar die Standarddatenbank im Linuxbereich, aber noch nicht ganz so leicht zu bedienen wie Access. Trotzdem ist Mysql sehr gut, da sie open source ist, sehr weit verbreitet ist und stabil läuft. Auch scheinen die Dateiformate nicht willkürlich verändert zu werden, wie dies bei MS Access ohne ersichtlichen Grund öfter der Fall war.

Sie sollten sich also etwas mit Mysql vertraut machen, bevor sie mit Gambas auf Mysql losgehen. Ansonsten werden Sie von *Access denied* Fehlern schnell frustriert sein.

Wenn Sie nicht wissen was eine Datenbank ist schauen Sie sich einmal folgende Seite an:

<http://www.madeasy.de/2/datbank.htm>

Die Einrichtung von mysql ist heute unter Linux kein allzu großes Problem mehr, da alle aktuellen Standarddistributionen die wichtigsten mysql Pakete enthalten.

#### 4.114.2 Can't connect to local MySQL server through socket

...

Wenn man Mysql unter Linux in Gang bringen will, nervt einen folgende Fehlermeldung immer wieder.

```
Can't connect to local MySQL server through socket ...
```

Was ist der Grund für diese Fehlermeldung? Meistens liegt das Problem bereits beim Starten von Linux. Der mysql daemon wird nicht gestartet. Um den Fehler zu beheben, müssen Sie in Ihren Runleveleditor gehen. Dort aktivieren Sie Mysql, so dass die Datenbank bereits beim Booten mitgestartet wird.

Unter Suse starten Sie am besten YAST mit dem Rootpasswort. Dann geben Sie folgendes ein:

```
YAST - SYSTEM/RUNLEVEL-EDITOR/ Dienste
```

und aktivieren mysql, falls es nicht schon aktiv ist.

#### 4.114.3 Mysql von der Konsole

Nach der Installation starten Sie ihr mysql von der Konsole mit:

```
/etc/init.d/mysql start
```

Es erscheint die Meldung: failed ( nicht erfolgreich) oder

```
Starting service MySQL done ( erfolgreich)
```

Ich kann auf meinem Rechner mysql nur mit root Rechten starten.

Stoppen können Sie mysql mit:

```
/etc/init.d/mysql stop
```

Meldung:

```
Shutting down service mysql done
```

Starten Sie mysql erneut und geben Sie dann ein

```
mysql
```

Meldung:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer
mysql>
```

Schauen Sie sich den mysql status an und geben Sie *status* ein:

```
mysql> status
```

**Meldung:**

```
-----  
mysql Ver 12.22 Distrib 4.0.18, for suse-linux (i686)  
Connection id:      2  
Current database:  
Current user:       root@localhost  
SSL:                Not in use  
Current pager:     less  
Using outfile:  
Server version:    4.0.18  
Protocol version:  10  
Connection:        Localhost via UNIX socket  
Client characterset: latin1  
Server characterset: latin1  
UNIX socket:       /var/lib/mysql/mysql.sock  
Uptime:            3 min 37 sec  
Threads: 1  Questions: 3  Slow queries: 0  Opens: 6  Flush tables: 1  Open tables: 0  Queries per  
second avg:  
0.014
```

Wichtig für Gambas sind hier die current database ( noch keine geladen), der current user und die connection.

Haben Sie bereits ein Mysqpasswd vergeben kommen Sie in mysql nur mit

```
mysql -p
```

und dann der Eingabe des Passwortes hinein. Dieses Passwort hat nichts mit dem Linux Passwort zu tun und wird außerdem vom mysql noch verschlüsselt, steht also nicht im Klartext zur Verfügung. Merken Sie es sich gut, Sie brauchen es auch für Gambas und mysql.

Schauen Sie sich dann einmal die vorhandenen Datenbanken an zb die Testdatenbank test und geben Sie ein:

```
USE test;
```

**Meldung:**

```
mysql> USE test;  
Database changed
```

Lassen Sie sich nicht durch die Meldung: *Database changed* irritieren.

Lassen Sie sich nun die vorhandenen Tabellen in mysql anzeigen, aber vergessen Sie nicht den Strichpunkt hinter Ihrer Eingabe sonst wird mysql zickig.

```
SHOW TABLES;
```

Da auf meinem Rechner schon einiges in Test zu finden ist erscheint folgende Meldung:

```
+-----+  
| Tables_in_test |
```

```

+-----+
| NewTable |
| __RekallObjects |
| gewuerz |
| t1 |
| t2 |
| test |
+-----+
6 rows in set (0.00 sec)

```

Jetzt kann man sich die einzelne Tabelle beispielsweise gewuerz anschauen:

```
SELECT * FROM gewuerz;
```

Und ich erhalte eine wunderbare Gewürztabelle: `mysql> SELECT * FROM gewuerz;`

```

+-----+-----+-----+-----+-----+
| nr | name      | preis | waehrung | menge | einheit |
+-----+-----+-----+-----+-----+
| 1 | anis      | 4.6   | EUR      | 6.8   | kg      |
| 2 | pfeffer   | 5     | euro     | 2     | 4       |
| 3 | safran    | 5     | e        | 3     | 3       |
| 4 | salz      | 4     | e        | 4     | 4       |
| 5 | curry     | 5     | e        | 5     | 5       |
| 6 | knoblauch | 5     | e        | 5     | 5       |
+-----+-----+-----+-----+-----+
6 rows in set (0.02 sec)

```

Versuchen Sie sich doch auch so eine schöne Gewuerztabelle anzulegen.

Zur weiteren mysql Syntax empfehle ich folgende sehr hilfreiche Website:

<http://www.selflinux.org/selflinux/html/mysql.html> - Link leider defekt :-( Am besten vollziehen Sie das dortige Kunden Beispiel nach.

#### 4.114.4 Arbeiten mit dem Datenbankmanager von Gambas

Der Datenbankmanager von Gambas ist gut geeignet, um sehr schnell und übersichtlich mysql Datenbanken darzustellen, neu anzulegen und zu bearbeiten.

Sie finden ihn im Projektfenster unter dem Menüpunkt *Werkzeuge* und *Datenbank-Manager*.

Wenn Sie ihn starten fragt er Sie nach einem kryptischen Password , daß sie eingeben müssen, auch wenn Sie sonst mysql noch als root und ohne Passwort nutzen.

Geben Sie hier einfach Return ein , gibt es eine Fehlermeldung: *Please enter at least 8 characters* auf deutsch: Sie sollten ein Passwort mit mindestens 8 Buchstaben nutzen.

Haben Sie diese Passwörthürde erfolgreich gemeistert , erscheinen alle Datenbankserver in einer Treeviewansicht.

Bei mir funktioniert nur der Eintrag *root@localhost(mysql)*. Klicken Sie auf diesen, dann zeigen sich alle dort erreichbaren Datenbanken zb auch *test*. Klicken Sie auf eine der Datenbanken, dann zeigen sich die dortigen Tabellen. Klicken Sie auf eine Tabelle, dann können Sie diese nun im rechten Bildschirm in der Struktur ändern, neue Indizes setzen und auch neue Daten eingeben.

Probieren Sie alles aus und speichern Sie ihre Änderungen durch einen Klick auf das Diskettensymbol ab.

#### **4.114.5 MySQL Nutzerrechte**

#### **4.114.6 MySQL, Apache und phpMyAdmin**

Zur Darstellung von MySQL Datenbanken haben sich die Programme Apache als Webserver und phpMyAdmin als Zugriffsoberfläche sehr bewährt. Versuchen Sie diese beiden Programme auf ihrem Rechner zu installieren. Machen Sie sich dann etwas mit phpMyAdmin vertraut.

Es erleichtert die Fehlersuche enorm, wenn man eine schnelle weitere Zugriffsmöglichkeit auf die MySQL Datenbanken in Form des phpMyAdmin hat.

### **4.115 Datenbankprogrammierung mit Gambas und SQLite**

Nicht immer ist es notwendig, für eine kleine Datenbanklösung einen kompletten Server aufzusetzen. SQLite bietet die Möglichkeit mittels der notwendigen Treiber, welche für Gambas zur Verfügung stehen, eine DB in einer Datei ohne Server zu erzeugen.

Eine SQLite Datenbank kann man mittels dem Konsolenprogramm *sqlite*, nach Installation der notwendigen Treiber, mit Knoda oder natürlich auch mit dem gambas-eigenen Datenbankmanager erstellen.

Eine genauere Beschreibung der Datenbank findet man unter <http://de.wikipedia.org/wiki/SQLite>.

Eingebunden wird die Datenbank durch die Variablen *.Type* und *.Name* der Connection, wobei für *.Type* "sqlite" und für *.Name* der absolute Pfad zur Datenbankdatei steht. Die Benutzerrechte richten sich nach den Dateirechten.



## 4.116 Erstellen einer Datenbankverbindung

Beispiel für ein Datenbank-Connect. Hierbei muss die gb.db Komponente in das Projekt mit eingebunden werden:

```
PRIVATE $hConn AS Connection
PUBLIC SUB GoConnect ()
  DIM sName AS String
  TRY $hConn.Close
  sName = "DatenbankName"
  WITH $hConn
    .Type = "mysql"
    .Host = "DatenbankServer"
    .Login = "Datenbank-Benutzerkennung"
    .Password = "Datenbank-Benutzerpasswort"
  END WITH
  $hConn.Name = sName
  $hConn.Open
CATCH
Message.Error(Error.Text)
END
PUBLIC SUB Form_Open ()
  $hConn = NEW Connection
  GoConnect
END
PUBLIC SUB Form_Close ()
  $hConn.Close
END
```

## 4.117 Verfügbare Datenbanken anzeigen

Es wird eine Datenbankverbindung mit der unter dem jeweiligen Datenbanksystem immer vorhandenen Standarddatenbank (postgresql: 'template1'; mysql: 'mysql' ...) hergestellt und die Anzahl und durch Abfrage der 'Collection' 'Databases' die dem angemeldetem Benutzer zur Verfügung stehenden Datenbanken angezeigt. Dazu wird als Datenbankname eine leere Zeichenkette angegeben. Der '.Type' ist entsprechend dem Datenbanktyp zu setzen.

Z.B. als Procedure in einem Modul:

```
PRIVATE Datenbankverbindung AS Connection
PRIVATE Datenbankname AS String
PROCEDURE ListeDatenbanken ()
  DIM Element AS Variant
  Datenbankverbindung = NEW Connection
  WITH Datenbankverbindung
    .Type = "postgresql"
    .Host = "Rechnername"
    .Login = "Benutzername"
    .Password = "Schutzwort"
    .Name = ""
  END WITH
  Datenbankverbindung.Open
```

```
PRINT "Anzahl Datenbanken: " & DB.Databases.Count
FOR EACH Element IN DB.Databases
  Print "Datenbankname: " & Element.Name
NEXT
Datenbankverbindung.Close
END
```

## 4.118 Links

- <http://software.newsforge.com/software/04/11/30/1653243.shtml?tid=72&tid=132>
  - Beispiel einer Gambas mysql Anwendung
- <http://www.postgresql.de/> Datenbanksystem PostgreSQL
- <http://dev.mysql.com/doc/refman/5.1/de/index.html> Handbuch Datenbanksystem MySQL

Drucken funktioniert unter Gambas mit dem Printer Objekt. Es findet sich in der Komponente GB.QT.

Siehe Gambas Komponenten.

Wenn Sie mehr Informationen zum Printer Objekt brauchen, dann wählen Sie in Gambas mit F1 die Hilfe Funktion. Im Hilfe Browser können Sie im SUCHEN Feld das Stichwort Printer eingeben.

## 4.119 Ein paar Zeilen im Direktausdruck

Man kann den Drucker ohne Dialog ansteuern. Wie das geht zeigt folgendes Programm. Sie brauchen dazu:

- eine Form
- einen Befehlsknopf
- eine Textarea

Holen Sie sich diese Steuerelemente mit F6 aus der Werkzeugkiste.

Dann fügen Sie folgenden Code ein:

```
PUBLIC SUB Form_Open()
ME.Text = "Druckprogramm mit dem Printer Befehl"
ME.Width = 400
ME.Height = 400
ME.Center
```

```

Button1.Text = "Ausdruck im Querformat"
Button1.Width = 200
Button1.X = 16
Button1.Y = 24
Textareal.Text = "Dies ist ein Test für den Drucker!"
Textareal.X = 16
Textareal.Y = 64
Textareal.Width = 312
Textareal.Height = 248
END
PUBLIC SUB Button1_Click()
Printer.Colormode = 0
'Stellt den Schwarzweißdruck ein
Printer.Orientation = 0
'Hochformat
Draw.Begin(Printer) ' Initialisiert den Ausdruck
Draw.Text(TextAreal.Text, 200, 200)
' Druckt den Inhalt der TextArea in die linke obere Ecke
Draw.End ' Die Seite wird an den Drucker abgeschickt.
END

```

Der erste Teil des Programmes ist für den Ausdruck überflüssig. Es erleichtert Ihnen nur die Einstellung der Steuerelemente auf der Form und wird direkt beim Start des Programmes ausgeführt. ( Sub Form Open ) Der eigentliche Ausdruck erfolgt erst nach anklicken von Button1. Dort finden Sie die Druckerbefehle.

## 4.120 Drucken mit Dialog

Man kann den Drucker mit einem Auswahl Dialog ansteuern, um den gewünschten Drucker vor dem Druck zu wählen. Hierzu erweitern Sie das Beispiel "Ein paar Zeilen im Direktausdruck" um den Befehl: **Printer.Setup()**

Wie das geht zeigt folgendes Programm. Sie brauchen dazu:

- eine Form
- einen Befehlsknopf
- eine Textarea

Holen Sie sich diese Steuerelemente mit F6 aus der Werkzeugkiste.

Dann fügen Sie folgenden Code ein:

```

PUBLIC SUB Form_Open()
ME.Text = "Druckprogramm mit dem Printer Befehl"
ME.Width = 400
ME.Height = 400
ME.Center
Button1.Text = "Ausdruck im Querformat"
Button1.Width = 200
Button1.X = 16
Button1.Y = 24

```

```
Textareal.Text = "Dies ist ein Test für den Drucker!"
Textareal.X = 16
Textareal.Y = 64
Textareal.Width = 312
Textareal.Height = 248
END
PUBLIC SUB Button1_Click()
Printer.Setup() ' Auswahl Drucker
Printer.Colormode = 0
' Stellt den Schwarzweißdruck ein
Printer.Orientation = 0
' Hochformat
Draw.Begin(Printer) ' Initialisiert den Ausdruck
Draw.Text(Textareal.Text, 200, 200)
' Druckt den Inhalt der TextArea in die linke obere Ecke
Draw.End ' Die Seite wird an den Drucker abgeschickt.
END
```

Der erste Teil des Programmes ist für den Ausdruck überflüssig. Es erleichtert Ihnen nur die Einstellung der Steuerelemente auf der Form und wird direkt beim Start des Programmes ausgeführt. ( Sub Form Open ) Der eigentliche Ausdruck erfolgt erst nach anklicken von Button1. Dort finden Sie die Druckerbefehle.

## 4.121 Drucken im Querformat

Hierzu können Sie das oben erwähnte Beispiel "Ein paar Zeilen im Direktausdruck" anpassen. Ändern Sie den Befehl **Printer.Orientation = 0** nach **1** Der Wert für das **Hochformat = 0** und für das **Querformat = 1**

## 4.122 Drucken einer Grafik

## 4.123 Drucken in eine Postscript PS Datei

## 4.124 Drucken Abbrechen

## 4.125 Printer Klasse Theorie

In Gambas ist das Printer Objekt ein grafisches Gerät, deswegen nutzen Sie die Methoden der Draw Klasse um damit auszudrucken.

Beispiel:

```
IF Printer.Setup() THEN RETURN
'Dialog anschalten

Draw.Begin(Printer) ' Initializes the draw
Draw.Text(TextArea.Text, 200, 200) ' Druckt den Inhalt der Textarea in die linke obere Ecke des
Druckers.
Draw.Foreground = Color.Red
Draw.Rect(500, 200, 32, 32) ' Druckt ein kleines Rechteck
Draw.End ' Die Seite wird an den Drucker gesendet.

Printer.NewPage ' Eine neue Seite wird angesteuert.

Die Printer Klasse ist statisch.
```

### 4.125.1 Eigenschaften von Printer

- ColorMode
- 'Schwarzweiß oder Farbe
- Copies
- 'Zahl der Kopien
- File
- 'In eine PS datei drucken
- FromPage
- ' Ausdruck erst mit dieser Seite beginnen
- ToPage
- 'Ausdruck bei dieser Seite bereits beenden
- Height
- 'Seitenlänge
- Width
- 'Seitenbreite
- MaxPage
- 'Die größte Seitennummer die man wählen kann
- MinPage
- 'Die kleinsten Seitennummer die man wählen kann
- Name

'Name des Druckers

- Orientation

'Hoch- oder Querformat

- Resolution

'Auflösung in DPI

- Size

'DIN A3,A4 Letter etc

- 

#### **4.125.2 Methoden von Printer**

- Abort

'Druck abbrechen

- Setup

'Druckdialog starten

- NewPage

'Seitenvorschub Neue Seite anfangen

#### **4.125.3 Konstanten von Printer**

Black 'Schwarzweiß

Color 'Farbausdruck

Landscape 'Querformat

Portrait 'Hochformat

### **4.126 Printer.Print**

Den beliebten VB Befehl Printer.Print gibt es leider in Gambas noch nicht. Das kann man vielleicht ergänzen.

## 4.127 Html Browser

Man kann mit Gambas recht einfach HTML Seiten darstellen. Wie das geht zeigt folgendes Programm. Es wird genutzt, um das Gambaswikibook lokal auf dem Rechner aufrufen zu können.

Sie brauchen ein Form mit

- einer Textbox
- einem Labelfeld
- einem Webbrowserfeld

Um das Webbrowserfeld zu bekommen müssen sie erst die Gambas Komponenten

- gb.qt.kde und
- gb.qt.kde.html

zu ihrem Projekt dazuladen.

Wie das geht, ist unter [Gambas-Komponenten](#) erklärt.

So schaut dann Ihre Form im Entwurfsmodus aus.

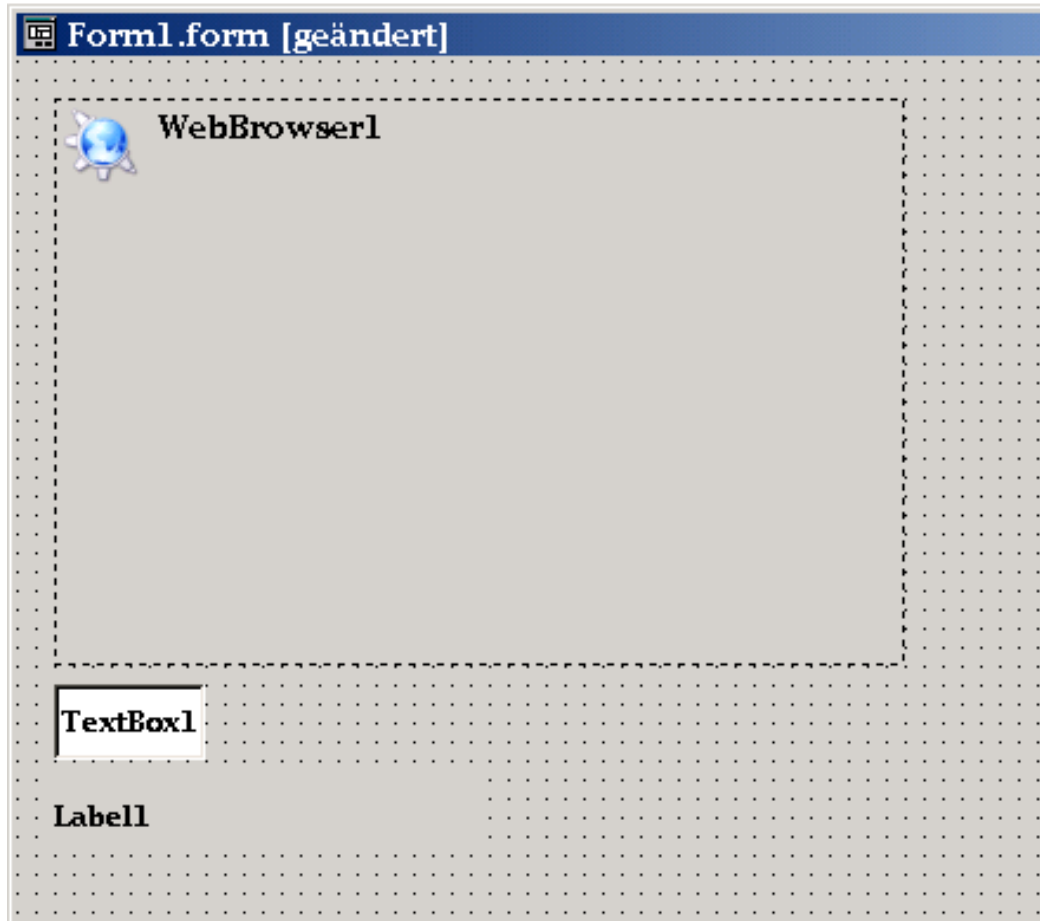


Abbildung 79: Bild:Htmlbrowser.png

Fügen Sie folgen Code in Ihr Programm:

```
' Gambas class file
PUBLIC SUB Form_Resize()
  textbox1.Move(0, 0, ME.ClientW)
  WebBrowser1.Move(0, textbox1.H, ME.ClientW, ME.ClientH - textbox1.H - labell.H)
  labell.Move(0, WebBrowser1.Y + WebBrowser1.H, ME.ClientW)
END
PUBLIC SUB Form_Open()
  textbox1.Text = "file:/home/BENUTZER/wiki/Gambas"
  txtLink_Activate
END
PUBLIC SUB WebBrowser1_Change()
  textbox1.Text = LAST.Path
  'DEC Application.Busy
END
PUBLIC SUB WebBrowser1_Click()
  textbox1.Text = " " & LAST.Link
  'INC Application.Busy
```



```
END
PUBLIC SUB WebBrowser1_Link()
    labell1.Text = LAST.Link
END
PUBLIC SUB txtLink_Activate()
    DIM sPath AS String
    sPath = textbox1.Text
    WebBrowser1.Path = Trim(sPath)
END
```

Damit das Programm läuft müssen Sie auf ihrem Rechner eine lokale Kopie des Gambas-Wikibooks einrichten, siehe [Einrichten einer lokalen Kopie](#).

Dann können Sie das Programm als Gambas Hilfebrowser auf ihrem Rechner einrichten und immer die neuesten Gambasseiten aus dem GambasWikibook herunterladen.

Nur das Zurückblättern funktioniert noch nicht. Man muss in der obersten Zeile im auf *Zurück zu Gambas* klicken , dann kommt man wieder zum Hauptmenü.

Ähnlich kann man sich andere Wikibooks auf dem eigenen Rechner einrichten.

Das Ganze funktioniert natürlich auch mit dem Konqueror. Im Konqueror ist dann auch ein Zurückblättern möglich.

## 4.128 Erweiterter Browser

Um das Problem der fehlenden Möglichkeit zum Zurückblättern zu lösen, ergänzen wir nun die obige Form um drei Buttons: **Button1**, **Button2** und **Button3**.

Schließlich übernehmen wird den folgenden Code in unser Programm: [Wikibrowser2 Form1.class](#)



Abbildung 80: Wiki Browser 2

## 4.129 Gambas Web Browser

Bei den mitgelieferten Gambas Beispielprogrammen ist ein kleiner Webbrowser dabei. Erst ist sehr praktisch und leicht erweiterbar.

## 4.130 Codebeispiel für die RS 232 Schnittstelle

```
PUBLIC sub Write_Serial (a AS String)

IF Sport.status = Net.Inactive THEN
    Message ("open port first")
ELSE
    chkrts.Value = TRUE
    WAIT 0.01
```

```

WRITE #Sport, a, Len(a)
WAIT 0.5
chkrts.Value = FALSE
ENDIF

```

Das Serielle Gerät benutzt hier RTS UP um Daten zu akzeptieren. Wenn RTS aus ist, werden diese Daten übertragen.

Man kann die Linux Shell von Gambas aus ansprechen und nutzen. Wie das geht zeigen die folgenden Beispiele:

## 4.131 Ls unter Gambas

Das folgende Beispielprogramm zeigt, wie man sich die Dateien eines Verzeichnisses der Festplatte in einer Textarea anschauen kann.

Man braucht eine Textarea und einen Befehlsbutton, um das Programm in gang zu bringen.

```

' Gambas class file
PUBLIC Content AS String
'Get the content of a directory
PUBLIC SUB Button1_Click()
  SHELL "ls -la > /tmp/result" WAIT
  Content = File.Load("/tmp/result")
  Textareal.text = content
END

```

Man kann den String mit dem Namen Content aus dem Programm entfernen, wenn man die Textarea direkt mit dem Fileload füllt.

```

Textareal.text = File.Load("/tmp/result")

```

### 4.131.1 Noch ein Beispiel: Unterverzeichnisse anschauen

Man braucht eine Textarea und 2 Befehlsbuttons, um das Programm in Gang zu bringen.

```

PUBLIC SUB Form_Open()
  ME.Text = "Unterverzeichnisse anzeigen - Wählen Sie einen Pfad"
  ME.X = 200
  ME.Y = 50
  ME.Width = 1200
  ME.Height = 600
  Button1.Text = "Unterverzeichnisse anzeigen"
  Button1.X = 16
  Button1.Y = 16
  Button1.Width = 300
  textareal.Text = ""

```

```
textareal.x = 16
textareal.y = 56
textareal.width = 700
textareal.height = 500
Button2.Text = "Ende"
Button2.X = 340
END
PUBLIC SUB Button1_Click()
DIM t AS String
IF Dialog.SelectDirectory() THEN RETURN
t = Dialog.Path
SHELL "ls -ld " & t & "*/. > /tmp/result" WAIT
Textareal.text = File.Load("/tmp/result")
END
PUBLIC SUB Button2_Click()
ME.Close
END
```

Im ersten Unterprogramm `Form_Open` werden nur einige Einstellungen für die 3 Steuerelemente vorgenommen. Im 2. Unterprogramm `Button1_Click` steht der eigentlich wichtige Code: Zunächst wird ein string `t` deklariert. Dann wird eine Dialogform zur Auswahl des Verzeichnisses geöffnet. Das ausgewählte Verzeichnis (`Dialog.path`) wird dem String `t` übergeben. Mit dem Shell Befehl werden alle Unterverzeichnisse von `Dialogpath` in eine temporäre Datei gespeichert. Diese Datei wird geladen und in der Textarea angezeigt.

## 4.132 wc = wordcount unter Gambas

Im folgenden Programm wird der Linuxbefehl `wc`, der die Zeilen, die Worte und die Zeichen eines beliebigen Textes zählt, innerhalb eines Gambasprogrammes genutzt.

Um das Programm in Gang zu bringen braucht man:

- 2 Befehlsbuttons
- 1 Textarea ( Hier wird der zu untersuchende Text eingegeben)
- 1 Textbox ( Hier wird das Ergebnis des wordcount ausgegeben)

Der Code dazu:

```
PUBLIC t AS String
PUBLIC SUB Form_Open()
Textareal.Text = "Dies ist ein Test"
Textbox1.Text = ""
END
PUBLIC SUB Button1_Click()
File.Save(system.Home & "/test.txt",TextAreal.Text)
CATCH
IF ERROR THEN RETURN
END
```

```
PUBLIC SUB Button2_Click()  
  SHELL "wc " & system.Home & "/test.txt > /tmp/result" WAIT  
  t = File.Load("/tmp/result")  
  textbox1.Text = t  
END
```

- Mit dem Befehl 1 wird der Inhalt der Textarea in einer Datei test.txt abgespeichert.
- Mit dem Befehl 2 wird diese Datei test.txt einem wordcount unterzogen. das Ergebnis in der Datei /tmp/result abgespeichert.
- Diese Datei wird dann wieder in die Textbox geladen und angezeigt.

Etwas umständlich aber es geht. Der String t ist eigentlich überflüssig und kann aus dem Programm entfernt werden wenn man

```
textbox1.Text = File.Load("/tmp/result")
```

schreibt.

## 4.133 Laden der Kompressions-Komponente

Um mit Gambas komprimierte Datei zu erstellen, brauchen Sie die Gambas-Komponente **gb.compress**. Sie können diese über die Gambas-Projektseite zuschalten. Dazu klicken Sie auf den Menüpunkt *Projekt* und dann weiter auf *Eigenschaften*. Von den dort angebotenen Reitern wählen Sie *Komponenten*. Suchen Sie aus der Liste **gb.compress** heraus und versehen Sie die Box davor mit einem Häkchen.

Siehe auch [Komponenten](#)

## 4.134 Beispielprogramm

Im folgenden Beispiel wird eine Textdatei erzeugt. Diese wird unkomprimiert abgespeichert. Dann wird sie komprimiert und gespeichert. Die komprimierte Datei wird geladen und ausgegeben. Dann wird die komprimierte Datei wieder entkomprimiert und mit PRINT im Direktfenster ausgegeben.

Sie brauchen:

- 1 Textarea
- 7 Befehlsfelder (Commandbuttons)

um das Programm in Gang zu bringen.

Der Quellcode:

```
' Gambas class file
PUBLIC DateiTxt AS String
PUBLIC DateiZip AS String
PUBLIC DateiUnzip AS String

PUBLIC SUB Form_Open()
  Textareal.Text = ""
  ME.Text = "Kompression mit Gambas"
  DateiTxt = System.Home & File.Separator & "ziptest.txt"
  DateiZip = System.Home & File.Separator & "ziptest.txt.gz"
  DateiUnzip = System.Home & File.Separator & "unziptest.txt"
END

PUBLIC SUB Button1_Click()
  s AS String
  sp AS String
  n AS Integer
  s = " Dies ist ein Test "

  FOR n = 1 TO 10
    s = s & Str(n)
    sp = sp & s
  NEXT

  PRINT sp
  TextAreal.Text = sp
END

PUBLIC SUB Button2_Click()
  'Testdatei speichern
  File.Save( DateiTxt, TextAreal.Text )
  CATCH
  IF ERROR THEN RETURN
END

PUBLIC SUB Button3_Click()
  DIM Cp AS Compress
  'DIM Cp AS NEW Compress
  Cp = NEW Compress
  Cp.Type = "zlib"
  Cp.File ( DateiTxt, DateiZip, Cp.Max )
END

PUBLIC SUB Button4_Click()
  DIM sLine AS String
  sLine = File.Load(DateiZip)
  'Die Datei muss existieren,
  'sonst kann sie nicht geladen werden.
  PRINT sLine
  'TextAreal.Text = sline
END

PUBLIC SUB Button5_Click()
  DIM UCp AS Uncompress
  UCp = NEW Uncompress
  UCp.Type = "zlib"
  UCp.File ( DateiZip, DateiUnzip )
```

```
END

PUBLIC SUB Button6_Click()
    DIM sLine AS String
    sLine = File.Load(DateiUnzip)
    'Die Datei muss existieren,
    'sonst kann sie nicht geladen werden.
    PRINT sLine
    'TextArea1.Text = sline
END

PUBLIC SUB Button7_Click()
    ME.Close
END
```

## 4.135 In den mitgelieferten Beispielen finden sich mehrere Spiele

- DeepSpace
- RobotFindsKitten
- Snake
- Solitaire

## 4.136 Die Türme von Hanoi

Kommt demnächst, oder email an rho54@gmx.de

## 4.137 Sprint eine Minispiel für 2

Ein einfaches Spiel für 2 Spieler. Wer denn letzten Stein setzt, hat gewonnen.

Man braucht:

- 40 Textboxen (stöhn)
- 2 Textareas
- 4 Commandbuttons

Quellcode (Quick and dirty):

- [Sprint Form1.class](#)
- [Sprint Form1.form](#)

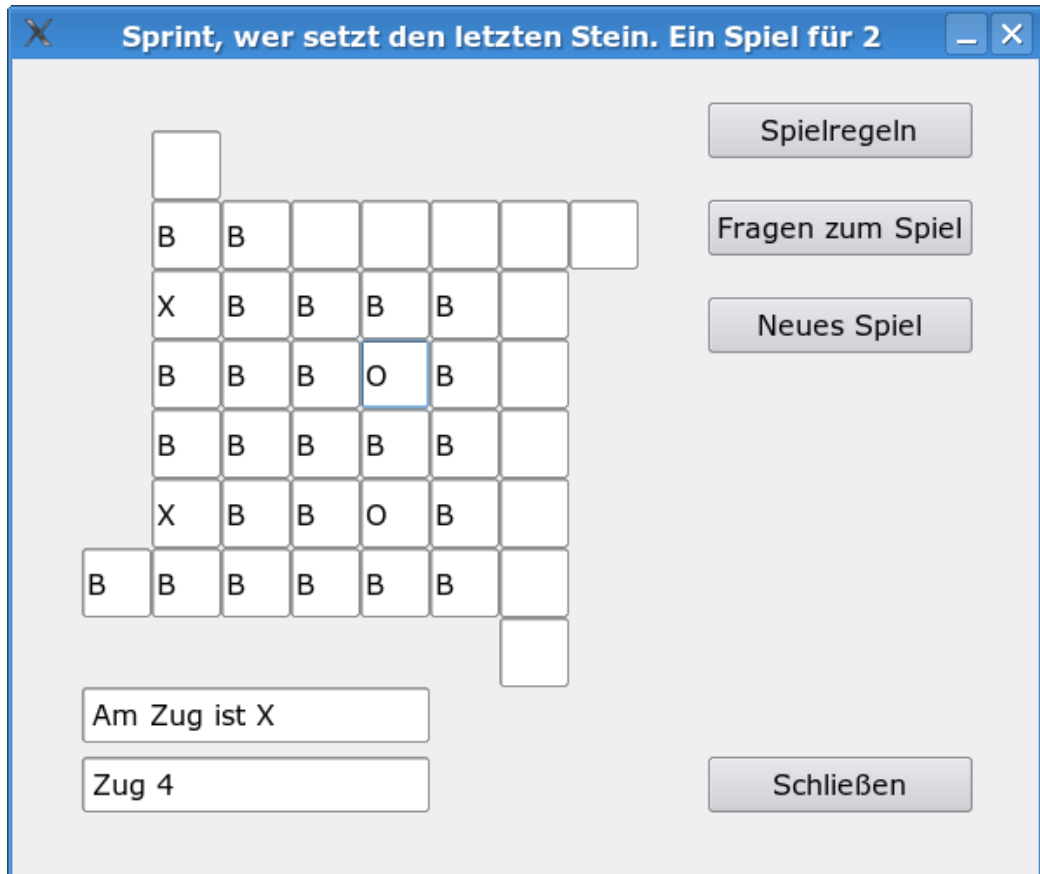


Abbildung 81: Bild:Gambassprint.png

Diese Programmierung ist stark verbesserungsbedürftig. Es liegt eine typische Anfängerlösung vor. Trotzdem kann man sie erst einmal stehen lassen, denn sie funktioniert. Als Hausaufgabe kann man jetzt ans verbessern gehen.

#### 4.137.1 Verbesserungen

Durch die vielen Textboxen lohnt es sich hier besonders die Textboxen in einem Array unterzubringen, dadurch kann man den Programmcode stark vereinfachen. Man braucht jetzt nur noch:

- 2 Textboxen



- 4 Commandbuttons

Quellcode: [Sprint2 Form1.class](#)



Abbildung 82: Bild:Gambassprint2.png

## 4.138 TicTacToe

Im folgenden Programm wird eine ganze einfache Version des Spiels für 2 Spieler gezeigt.

Man braucht dazu folgende Steuerelemente auf der Form

- 9 Textboxen
- 2 Textareas

- 4 Commandbuttons

Quellcode:

- [TicTacToe Form1.class](#)
- [TicTacToe Form1.form](#)

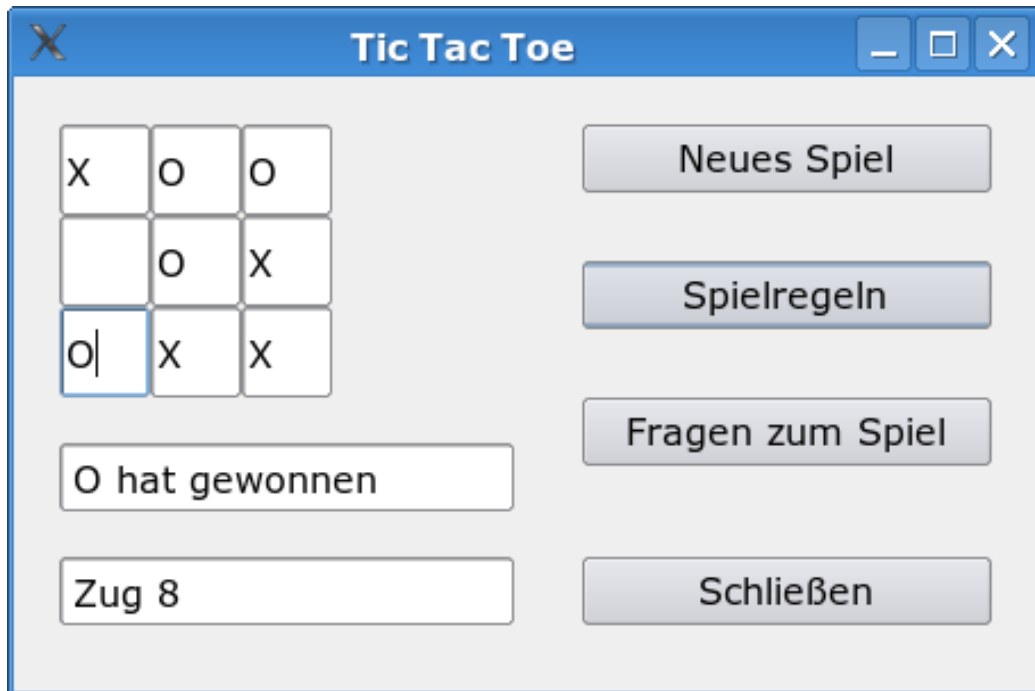


Abbildung 83: Bild:Gambastictactoe.png

Die Gliederung ist noch verbesserungsbedürftig. Das Programm arbeitet mit zwei Unterprogrammen *wechsel* und *gewonnen*. Die ganze Programmierung kann sicher noch stark verbessert werden. Man könnte beispielsweise ein Array für die Textfelder einbauen. Als erste lauffähige Lösung kann man obiges Programm aber durchaus brauchen und sollte sie aus didaktischen Gründen stehen lassen. Sinnvolle Verkürzungen sind für Anfänger schwer zu verstehen und sollten Schritt für Schritt weiter unten erfolgen.

#### 4.138.1 Verbesserungen

Wenn wir jetzt, wie oben angemerkt, die Textboxen in einem Array unterbringt, kann man den Programmcode vereinfachen. Man braucht jetzt nur noch:

- 2 Textboxen
- 4 Commandbuttons

Quellcode: [TicTacToe2 Form1.class](#)



Abbildung 84: Bild:Gambastictactoe2.png

## 4.139 Links

- <http://gamebas.tuxfamily.org/download/> (funktioniert nicht mehr)

## 4.140 Konverter (Text zu HTML-Code)

Dieses Programm verwandelt Text in Html-Zeichencode.

Sie Benötigen dazu:

- 1 Form

- 1 Button
- 1 Textlabel
- 2 Textboxen

Ergebnis AS String

```
PUBLIC SUB Button1_Click()
    i AS Integer
    Ergebnis = ""

    'Umwandlung vom "utf-8" Zeichensatz zum "ISO-8859-1" Zeichensatz
    Textbox2.Text = Conv$(Textbox1.Text, "utf-8", "ISO-8859-1")

    FOR i = 1 TO Len(Textbox2.Text)
        ' "Len" gibt die Länge des übergebenen Strings zurück

        Ergebnis = Ergebnis & Chr$(38) & "#" & Asc( Textbox2.Text, i ) & ";"
        ' "Asc" gibt den ASCII-Code des Zeichen an der angegebenen Stelle des Stings zurück
        ' Chr$(38) ergibt das & Zeichen, hier nur verwendet um mögliche
        ' Darstellungsprobleme im Browser zu vermeiden
    NEXT

    Textbox2.Text = Ergebnis
END

PUBLIC SUB TextBox1_Enter()
    Button1_Click
END
```

**So sieht es dann aus:**

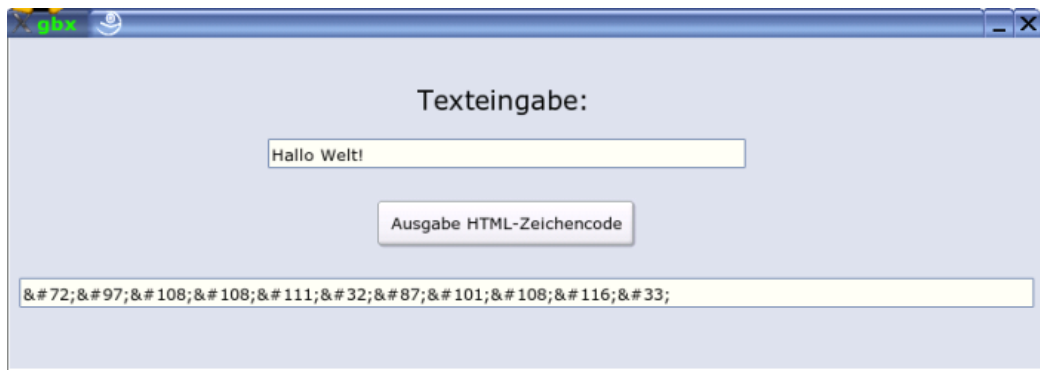


Abbildung 85: Text zu HTML-Code

## 4.141 Youtube Downloader

Ermöglicht das Runterladen von Youtubevideos anhand eines Youtubelinks. Die Komponenten `gb.net`, `gb.net.curl`, `gb.desktop`, `gb.gui`, `gb.form` sind notwendig.

Hier die Form

```
# Gambas Form File 2.0

{ Form Form
  MoveScaled(0,0,38.5714,8.5714)
  Text = ("YoutubeDownloader")
  Border = Window.Fixed
  { TextBox1 TextBox
    MoveScaled(1,1,21,3)
    ToolTip = ("Here goes the link")
    Text = ("")
  }
  { PictureBox1 PictureBox
    MoveScaled(23,1,14.1429,7.1429)
  }
  { Button1 Button
    MoveScaled(1,5,21,3)
    ToolTip = ("click here if you are ready to download")
    Text = ("Download")
  }
}
```

Hier der Quellcode

```
' Gambas class file
CONST PICTURENAME AS String = "youtube.jpg"
CONST YOUTUBE AS String = "http://www.youtube.com"

PUBLIC SUB Form_Open()
  DIM pic AS Picture

  pic = Picture.Load(Application.Path & "/" & PICTURENAME)
  pic.Resize(100, 50)
  PictureBox1.Picture = pic
  Application.MainWindow.Icon = pic
END

PUBLIC SUB Button1_Click()
  getYoutubeFile(TextBox1.Text)
END

PUBLIC SUB PictureBox1_MouseDown()
  Desktop.Open(YOUTUBE)
END

' Öffnet den Standardbrowser mit dem Video dessen
' Youtubeurl als Parameter übergeben wird
PUBLIC SUB getYoutubeFile(url AS String)
  DIM contents AS String

  url = parseYoutube(url)
  IF url = NULL THEN
```

```
        Message.Error("Error: problems with parsing the page!")
        RETURN
    END IF

    Desktop.Open(url)
END

' parsed eine Youtubeseite um den tatsächlichen Lageort des Videos zu finden
PUBLIC FUNCTION parseYoutube(url AS String) AS String
    DIM contents AS String
    DIM tmp AS String
    DIM tmp2 AS String
    DIM tval AS String
    DIM id AS String

    ' seite laden
    contents = getFile(url)
    IF contents = NULL THEN
        RETURN NULL
    END IF

    ' tval
    tmp = String.InStr(contents, "\"t\": \"", 0) + String.Len("\"t\": \"")
    tmp2 = String.InStr(contents, "\", ", tmp)
    tval = String.Mid(contents, tmp, tmp2 - tmp)

    ' id
    tmp = String.InStr(url, "v=", 0) + String.Len("v=")
    id = String.Mid(url, tmp)

    RETURN "http://www.youtube.com/get_video?video_id=" & id & "&t=" & tval
END

' Läd eine Seite runter und gibt diese als String zurück
PUBLIC FUNCTION getFile(url AS String) AS String
    DIM h AS HttpClient
    DIM buffer AS String

    buffer = NULL
    h = NEW HttpClient AS "h"
    h.URL = url
    h.Get()

    DO WHILE h.Status <> 0
        WAIT 0.01
    LOOP

    IF h.Status >= 0 THEN
        IF Lof(h) THEN
            READ #h, buffer, Lof(h)
        END IF
    END IF

    RETURN buffer
END
```

## 4.142 Lernprogramm zum Addieren ( Zusammenzählen)

Mit diesem kleinen Lernprogramm können Sie Addieren ( Zusammenzählen) lernen

Sie brauchen für das Programm eine grafische Form und

- 6 Textboxen
- 3 Commandbuttons

Das Programm erzeugt 2 Zufallszahlen a und b. Diese werden addiert mit dem Ergebnis  $c = a + b$ . In einer Textbox (hier: textbox5) gibt der Nutzer sein Ergebnis d ein. Diese wird durch den Befehl TESTEN mit dem wahren Ergebnis c überprüft.

Sie sollten die Commandbuttons folgendermassen bezeichnen:

- Button1 = Ende
- Button2 = Neue Aufgabe
- Button3 = Testen

Ändern Sie den Text der Commandbuttons im Eigenschaftsfenster entsprechend um.

Das Programm vergleicht d mit dem korrekten Ergebnis c in einem Unterprogramm bewertung und gibt das Ergebnis in einer der Textboxen aus. Die Gesamtzahl der Aufgaben , die Zahl der richtigen Aufgaben und die Zahl der falschen Aufgaben wird in 3 weiteren Textboxen ausgegeben.

Der Code dazu:

```
' Gambas class file
PUBLIC a AS Integer
PUBLIC b AS Integer
PUBLIC c AS Integer
PUBLIC SUB Form_Open()
z AS Integer
'zahl der Aufgaben
r AS Integer
'zahl der richtigen Aufgaben
f AS Integer
'Zahl der falschen Aufgaben
z = 0
r = 0
f = 0
Textbox1.Text = ""
TextBox2.Text = Str(z)
TextBox3.Text = Str(r)
TextBox4.Text = Str(f)
```

```
Textbox5.Text = ""
Textbox6.Text = ""
END
PUBLIC SUB Button1_Click()
ME.Close
END
PUBLIC SUB Button2_Click()
z AS Integer
r AS Integer
f AS Integer
d AS Integer
z = Val(textbox2.text)
z = z + 1
Randomize()
' Rem Zähler für die Zahl der Aufgaben
a = Int(Rnd(1) * 10)
'Rem Zufalls Zahl holen
b = Int(Rnd(1) * 10)
'Rem zweite Zufallszahl holen
c = a + b
PRINT c
textbox1.Text = " Addiere " & Str$(a) & " und " & Str$(b) & " überprüfe das Ergebnis"
textbox2.Text = Str(z)
END
PUBLIC SUB Button3_Click()
d AS Integer
d = Val(Textbox5.text)
bewertung
CATCH
IF ERROR THEN message.Info("Bitte geben Sie ihr Ergebnis in der Textbox neben dem Button TESTEN ein!")
END
SUB bewertung ()
r AS Integer
d AS Integer
f AS Integer
r = Val(textbox3.text)
d = Val(textbox5.text)
f = Val(textbox4.text)
IF d = c THEN textbox6.Text = "Richtig: " & Str$(a) & " + " & Str$(b) & " = " & Str(c)
IF d = c THEN r = r + 1
textbox3.Text = Str(r)
IF d <> c THEN textbox6.Text = "Falsch !! Richtig wäre gewesen "
IF d <> c THEN textbox6.Text = textbox6.Text & Str$(a) & " + " & Str$(b) & " = " & Str$(c)
IF d <> c THEN f = f + 1
textbox4.Text = Str(f)
END
```

Haben Sie das Programm in gang gebracht ? Was hier fehlt ist ein Übersichtsbild der Form. Das wird noch nachgeliefert. Wenn Sie die Variablen d,r,f,z noch als public vor den Unterprogrammen deklarieren, können Sie den Code noch deutlich vereinfachen.

Jetzt versuchen Sie einmal das Programm zu variieren:

- Erstellen Sie ein neues Programm mit Zufallszahlen bis 100.
- Erstellen Sie ein neues Programm mit der Multiplikation als Ergebnis.



## 4.143 Lernprogramm zum Subtrahieren ( Abziehen)

Mit diesem kleinen Lernprogramm können Sie subtrahieren ( Abziehen) lernen

Sie brauchen für das Programm eine grafische Form und

- 3 Textboxen
- 3 Commandbuttons

Das Programm erzeugt 2 Zufallszahlen a und c. Diese werden von einander abgezogen mit dem Ergebnis  $c = a - b$ . In einer Textbox (hier: textbox5) gibt der Nutzer sein Ergebnis d ein. Diese wird durch den Befehl TESTEN mit dem wahren Ergebnis c überprüft.

Sie sollten die Commandbuttons folgendermassen bezeichnen:

- Button1 = Ende
- Button2 = Neue Aufgabe
- Button3 = Testen

Ändern Sie den Text der Commandbuttons im Eigenschaftsfenster entsprechend um.

Das Programm vergleicht d mit dem korrekten Ergebnis c in einem Unterprogramm bewertung und gibt das Ergebnis in einer der Textboxen aus.

Im Unterschied zum Addier-Lernprogramm erfolgt keine Zählung der Aufgaben.

Der Code dazu:

```
' Gambas class file
PUBLIC a AS Integer
PUBLIC b AS Integer
PUBLIC c AS Integer
PUBLIC d AS Integer
PUBLIC SUB Form_Open()
  Textbox1.Text = ""
  Textbox5.Text = ""
  Textbox6.Text = ""
END
PUBLIC SUB Button1_Click()
  ME.Close
END
PUBLIC SUB Button2_Click()
  Randomize()
  a = Int(Rnd(1) * 10)
  'Rem Zufalls Zahl holen
  c = Int(Rnd(1) * 10)
  'Rem zweite Zufallszahl holen
```

```
b = a - c
textbox1.Text = " Subtrahiere " & Str$(a) & " minus " & Str$(b) & " und überprüfe das
Ergebnis"
END
PUBLIC SUB Button3_Click()
d = Val(Textbox5.text)
bewertung
CATCH
IF ERROR THEN message.Info("Bitte geben Sie ihr Ergebnis in der Textbox neben dem Button TESTEN
ein!")
END
SUB bewertung ()
IF d = c THEN textbox6.Text = "Richtig: " & Str$(a) & " - " & Str$(b) & " = " & Str(c)
IF d <> c THEN textbox6.Text = "Falsch !! Richtig wäre gewesen "
IF d <> c THEN textbox6.Text = textbox6.Text & Str$(a) & " - " & Str$(b) & " = " & Str$(c)
END
```

Wieso wurde die Zahl  $c$  als Zufallszahl definiert und nicht  $b$  ? Wie kann man das Programm ändern , so daß keine negativen Zahlen für  $b$  herauskommen können ? Hier fehlt eine Schleife , so daß gilt  $c < a$ . Dann wird auch  $b$  immer eine positive Zahl sein.

## 4.144 Lernprogramm zum Multiplizieren

Mit diesem kleinen Lernprogramm können Sie Multiplizieren ( Malnehmen) lernen

Sie brauchen für das Programm eine grafische Form und

- 6 Textboxen
- 3 Commandbuttons

Das Programm erzeugt 2 Zufallszahlen  $a$  und  $b$ . Diese werden malgenommen mit dem Ergebnis  $c = a * b$ . Das Sternchen ist in der Computersprache das Zeichen der Multiplikation. In einer Textbox (hier: `textbox5`) gibt der Nutzer sein Ergebnis  $d$  ein. Dieses wird durch den Befehl TESTEN mit dem wahren Ergebnis  $c$  verglichen.

Sie sollten die Commandbuttons folgendermassen bezeichnen:

- Button1 = Ende
- Button2 = Neue Aufgabe
- Button3 = Testen

Ändern Sie den Text der Commandbuttons im Eigenschaftsfenster entsprechend um.

Das Programm vergleicht  $d$  mit dem korrekten Ergebnis  $c$  in einem Unterprogramm `bewertung` und gibt das Ergebnis in einer der Textboxen aus. Die Gesamtzahl der Aufgaben, die Zahl der richtigen Aufgaben und die Zahl der falschen Aufgaben wird in 3 weiteren Textboxen ausgegeben.

Das Programm läßt sich auch ohne Maus bedienen. Deswegen wurden die `KeyPress` Unterprogramme hinzugefügt. Etwas weniger funktioniert die `Backspace` Taste in der Textbox für die Ergebniseingabe. Deswegen wurde hier eine Umleitung programmiert.

Der Code dazu:

```
' Gambas class file
PUBLIC a AS Integer
PUBLIC b AS Integer
PUBLIC c AS Integer
PUBLIC r AS Integer
PUBLIC d AS Integer
PUBLIC f AS Integer
PUBLIC z AS Integer
PUBLIC SUB Form_Open()
  z = 0
  r = 0
  f = 0
  TextBox1.Text = ""
  TextBox2.Text = Str(z)
  TextBox3.Text = Str(r)
  TextBox4.Text = Str(f)
  TextBox5.Text = ""
  TextBox6.Text = ""
  Button2.Default = TRUE
END
PUBLIC SUB Button1_Click()
  ME.Close
END
PUBLIC SUB Button2_Click()
  TextBox5.Text = ""
  TextBox5.SetFocus
  z = Val(TextBox2.Text)
  z = z + 1
  Randomize()
  a = Int(Rnd(1) * 10)
  'Rem Zufalls Zahl holen
  b = Int(Rnd(1) * 10)
  'Rem zweite Zufallszahl holen
  c = a * b
  'PRINT c
  TextBox1.Text = " Multipliziere " & Str$(a) & " und " & Str$(b) & " und überprüfe das
Ergebnis"
  TextBox2.Text = Str(z)
END
PUBLIC SUB Button3_Click()
  d AS Integer
  d = Val(TextBox5.Text)
  Button2.Default = TRUE
  bewertung
CATCH
  IF ERROR THEN message.Info("Bitte geben Sie ihr Ergebnis in der Textbox neben dem Button TESTEN
```

```
ein!")
END
SUB bewertung ()
r = Val(textbox3.text)
d = Val(textbox5.text)
f = Val(textbox4.text)
IF d = c THEN textbox6.Text = "Richtig: " & Str$(a) & " mal " & Str$(b) & " = " & Str(c)
IF d = c THEN r = r + 1
textbox3.Text = Str(r)
IF d <> c THEN textbox6.Text = "Falsch !! Richtig wäre gewesen "
IF d <> c THEN textbox6.Text = textbox6.Text & Str$(a) & " mal " & Str$(b) & " = " & Str$(c)
IF d <> c THEN f = f + 1
textbox4.Text = Str(f)
END
PUBLIC SUB TextBox5_KeyPress ()
t AS String
l AS Integer
IF key.Code >= 48 AND key.Code <= 57 THEN
    Textbox1.text = Textbox1.text
ELSE IF key.Code = key.Return THEN
    Button3.setfocus
    Button3.Default = TRUE
ELSE IF key.Code = key.Backspace THEN
    t = textbox1.Text
    l = Len(t)
    t = Left$(t, l)
    textbox1.Text = t
    'etwas widersinnige Funktion von Backspace
ELSE IF key.Code = key.Delete THEN
    Textbox1.Text = ""
ELSE
    STOP EVENT
ENDIF
END
PUBLIC SUB Button3_KeyPress()
IF key.Code = key.Return THEN Button2.setfocus
END
```

## 4.145 Lernprogramm für die Vergleichszeichen >< und ist gleich

Mit diesem kleinen Lernprogramm können Sie die Bedeutung der Vergleichszeichen trainieren

- < kleiner als
- > größer als
- = ist gleich

Sie brauchen für das Programm eine grafische Form und

- 5 Textboxen

- 5 Commandbuttons

Die Form wurde F genannt. Sie können aber auch einfach den Namen Form1 beibehalten.

Das Programm erzeugt 2 Zufallszahlen. Diese werden verglichen und man muß das richtige Zeichen > oder < oder = wählen. Sie sollten die Commandbuttons folgendermassen bezeichnen:

- Button1 = Ende
- Button2 = Neue Aufgabe
- Button3 = >
- Button4 = "="
- Button5 = <

Ändern Sie den Text der Commandbuttons im Eigenschaftswindow entsprechend um.

Das Programm vergleicht die Eingabe über die Tasten >,< und = mit dem korrekten Ergebnis in einem Unterprogramm bewertung und gibt das Ergebnis in einer der Textboxen aus. Diese Ergebnistextbox wurde *bewert* genannt. Sie müssen den Namen der Textbox mit F4 in den Eigenschaften entsprechend verändern.

Der Code dazu:

```
' Gambas class file
PUBLIC a AS Integer
PUBLIC b AS Integer
PUBLIC c AS Integer
PUBLIC d AS Integer
PUBLIC z AS Integer
PUBLIC r AS Integer
PUBLIC f AS Integer
PUBLIC SUB Form_Open()
  z = 0
  r = 0
  f = 0
  TextBox2.Text = Str(z)
  TextBox4.Text = Str(f)
  TextBox3.Text = Str(r)
END
PUBLIC SUB Button1_Click()
  ME.Close
END
PUBLIC SUB Button2_Click()
  z = z + 1
  ' Zähler für die Zahl der Aufgaben
  Randomize()
  a = Int(Rnd(1) * 10)
  'Rem Zufalls Zahl holen
  b = Int(Rnd(1) * 10)
  'Rem zweite Zufallszahl holen
```

```
IF a > b THEN c = 62
' Rem Ascii Code für Größer als
IF a = b THEN c = 61
IF a < b THEN c = 60
textbox1.Text = "Vergleiche " & Str$(a) & " und " & Str$(b) & " und setze das richtige Zeichen
>, < oder = "
textbox2.Text = Str(z)
END
PUBLIC SUB Button3_Click()
d = 62
'Ascii Code für >
bewertung
END
PUBLIC SUB Button5_Click()
d = 60
bewertung
END
PUBLIC SUB Button4_Click()
d = 61
bewertung
END
SUB bewertung ()
IF d = c THEN bewert.Text = "Richtig: " & Str$(a) & " " & Chr$(c) & " " & Str$(b)
IF d = c THEN r = r + 1
textbox3.Text = Str(r)
IF d <> c THEN bewert.Text = "Falsch !! Richtig wäre gewesen "
IF d <> c THEN bewert.Text = bewert.Text & Str$(a) & " " & Chr$(c) & " " & Str$(b)
IF d <> c THEN f = f + 1
textbox4.Text = Str(f)
END
```

## 4.146 Harry's Rechnertrainer

ACHTUNG : Programm noch in Arbeit !!

Mit diesem Programm soll man die Grundrechnungsarten trainieren können.

### 4.146.1 Form und Steuerelemente

Sie brauchen dazu eine neue Form und

- 2 Textboxen
- 2 Textlabelfelder
- 2 Commandbuttons

## 4.146.2 Code

```
' Gambas class file
Zahl1 AS Integer
Zahl2 AS Integer
Ergebnis AS Integer
Antwort AS Integer
PUBLIC SUB StartAdditionen_Click()
  Randomize()
  Zahl1 = Int(10*Rnd()+1)
  Zahl2 = Int(10*Rnd()+1)
  Ergebnis = Zahl1+Zahl2
  Antwort = AntwortTextBox.Text
  AusgabeRechnungLabel.Text=Zahl1 & " + " &Zahl2 & " = " 'Anzeige der Rechnung
  TextLabelFeedback.Text = Ergebnis 'Ausgabe nur zur Kontrolle
  AntwortTextBox.SetFocus
END
PUBLIC SUB OkButton_Click()
  IF Antwort = Ergebnis THEN
    TextLabelFeedback.Text= "Richtig"
  ELSE
    TextLabelFeedback.Text = "Falsch"
  ENDIF
END
PUBLIC SUB AntwortTextBox_keypress()
  IF key.Code=key.Enter THEN
    TextLabelFeedback.Text = Ergebnis 'Ausgabe nur zur Kontrolle
  ENDIF
END
```

## 4.147 Eine Datei herunterladen

Um eine Datei herunterzuladen verwendet man Http, für Gambas gibt es dementsprechend die Klasse HttpClient. Diese ermöglicht einem mit den beiden typischen Methoden Post und Get Websites anzufordern. Hier ein Beispiel welches eine Website herunterläd und den Quellcode der Seite ausgibt (erfordert das Einbinden der [Komponenten](#) gb.net und gb.net.curl):

```
PUBLIC SUB Main()
  DIM contents AS String
  contents = getFile("http://gambas.sourceforge.net/")
  IF contents = NULL THEN
    PRINT "Fehler beim herunterladen!"
  ELSE
    PRINT contents
  END IF
END

' Läd eine Seite runter und gibt diese als String zurück
PUBLIC FUNCTION getFile(url AS String) AS String
  DIM h AS HttpClient
  DIM buffer AS String

  buffer = NULL
  h = NEW HttpClient AS "h"
```

```
h.URL = url
h.Get() ' Mit der Get-Methode die Website anfordern

DO WHILE h.Status <> 0
    WAIT 0.01
LOOP

' Wenn die Antwort OK ist gehts weiter
IF h.Status >= 0 THEN
    ' Wenn die Datei was beinhaltet...
    IF Lof(h) THEN
        ' lade sie runter -> in den string buffer
        READ #h, buffer, Lof(h)
    END IF
END IF

RETURN buffer
END
```

## 4.148 Erweiterter Taschenrechner

Formular



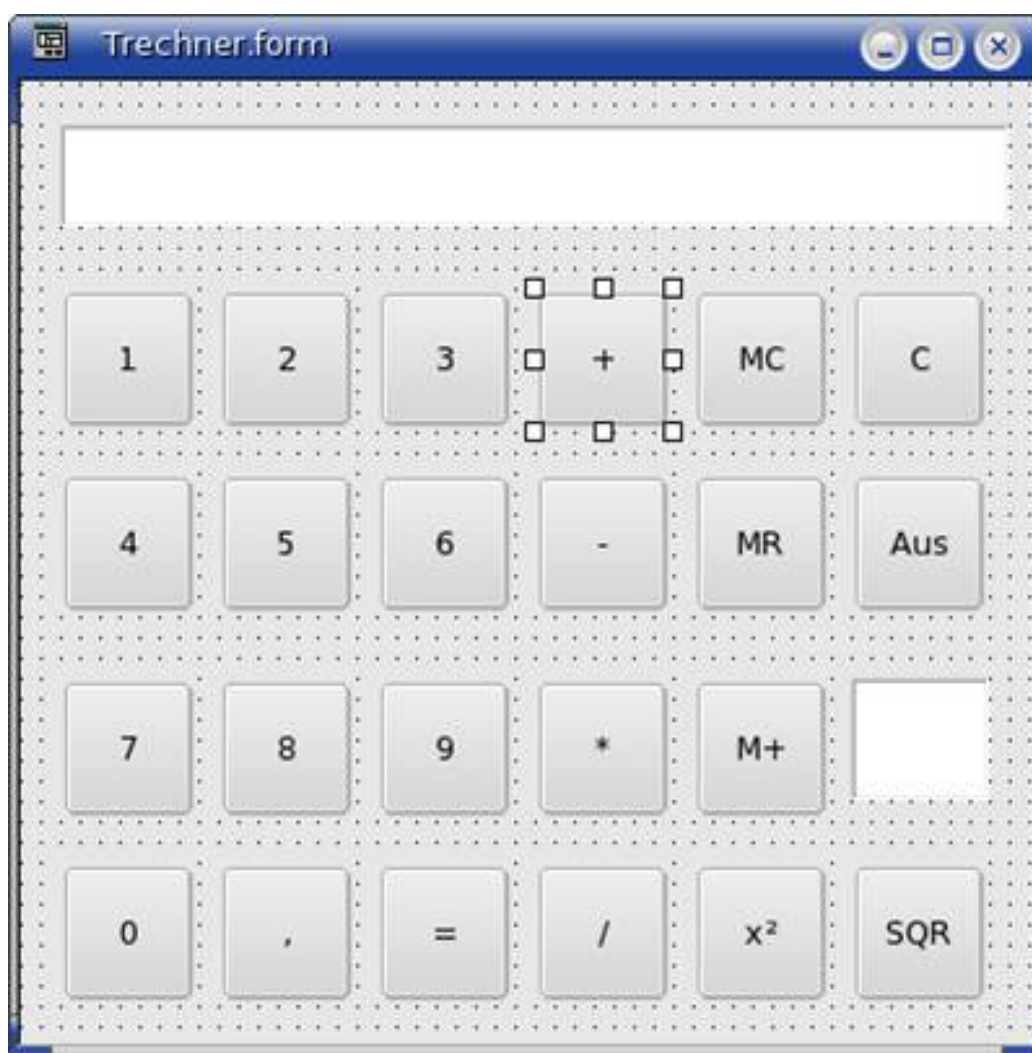


Abbildung 86: Bild:Image00GAMBAS.jpg

```
' Gambas class file
PUBLIC ergebnis AS Variant
PUBLIC speicher AS Variant
PUBLIC rechz AS Byte

PUBLIC SUB Button1_Click()
  Textbox1.Text = textbox1.Text & "1"
END
PUBLIC SUB Button2_Click()
  Textbox1.Text = textbox1.Text & "2"
END
PUBLIC SUB Button3_Click()
  Textbox1.Text = textbox1.Text & "3"
END
PUBLIC SUB Button7_Click()
```

```
Textbox1.Text = textbox1.Text & "4"
END
PUBLIC SUB Button8_Click()
Textbox1.Text = textbox1.Text & "5"
END
PUBLIC SUB Button9_Click()
Textbox1.Text = textbox1.Text & "6"
END
PUBLIC SUB Button13_Click()
Textbox1.Text = textbox1.Text & "7"
END
PUBLIC SUB Button14_Click()
Textbox1.Text = textbox1.Text & "8"
END
PUBLIC SUB Button15_Click()
Textbox1.Text = textbox1.Text & "9"
END
PUBLIC SUB Button18_Click()
Textbox1.Text = textbox1.Text & "0"
END

PUBLIC SUB Button19_Click()
Textbox1.Text = textbox1.Text & ","
END

PUBLIC SUB Button4_Click()
rechz=1
ergebnis=Val(Textbox1.Text)
Textbox1.Text=" "
END

PUBLIC SUB Button10_Click()
rechz=2
ergebnis=Val(Textbox1.Text)
Textbox1.Text=" "
END

PUBLIC SUB Button16_Click()
rechz=3
ergebnis=Val(Textbox1.Text)
Textbox1.Text=" "
END

PUBLIC SUB Button21_Click()
rechz=4
ergebnis=Val(Textbox1.Text)
Textbox1.Text=" "
END

PUBLIC SUB Button22_Click()
rechz=5
ergebnis=Val(Textbox1.Text)
Textbox1.Text=" "
END

PUBLIC SUB Button20_Click()
IF rechz=1 THEN
    ergebnis=ergebnis + Val(Textbox1.Text)
    textbox1.Text=ergebnis
ELSE IF rechz=2 THEN
    ergebnis=ergebnis - Val(Textbox1.Text)
    textbox1.Text=ergebnis
ELSE IF rechz=3 THEN
```

```
    ergebnis=ergebnis * Val(Textbox1.Text)
    textbox1.Text=ergebnis
ELSE IF rechz=4 THEN
    ergebnis=ergebnis / Val(Textbox1.Text)
    textbox1.Text=ergebnis
ELSE IF rechz=5 THEN
    ergebnis=ergebnis ^ Val(Textbox1.Text)
    textbox1.Text=ergebnis
CATCH
message.Error("Bitte nicht durch 0 teilen!")
END IF
END

PUBLIC SUB Button12_Click()
    ME.Close
END

PUBLIC SUB Button5_Click()
    speicher=" "
    textbox2.BackColor=&HD9ECEB&
END

PUBLIC SUB Button11_Click()
    textbox1.Text=speicher
END

PUBLIC SUB Button6_Click()
    rechz=0
    ergebnis=" "
    textbox1.Text=" "
END
```

## 4.149 Berechnung der Miete

The screenshot shows a window titled "BerechnungDerMiete.form" with the following content:

1. Bitte geben Sie die Fläche der Wohnung ein:  
 m<sup>2</sup> **al = Textbox1**

2. Wohnlage auswählen:  
 einfache Wohnlage **rb1 - Radiobutton1**  
 mittlere Wohnlage **rb2 - Radiobutton2**  
 gute Wohnlage **rb3 - Radiobutton3**

3. Miete ermitteln:

4. Ergebnis anzeigen:  
**Miete:** **textlabel7**  
 einfache Wohnlage 4 € pro m<sup>2</sup>  
 mittlere Wohnlage 7 € ---"----  
 gute Wohnlage 10 € ----"-----

Abbildung 87: Bild:IMG\_Mietberechnung.jpg

```

`Gambas class file
PUBLIC SUB Button1_Click()
  DIM al AS Integer
  al = Val(textbox1.Text)
  IF rb1.Value=TRUE THEN
    al=al*4
  ELSE IF rb2.Value=TRUE THEN
    al=al*7
  ELSE IF rb3.Value=TRUE THEN
    al=al*10
  END IF
  textlabel7.Text=al
END
PUBLIC SUB Button3_Click()
  ME.Close
END
PUBLIC SUB Button2_Click()

```

```

Textlabel7.Text=""
textbox1.Text=""
rb3.Value=TRUE
END
PUBLIC SUB Form_Open()
  ME.Caption="Berechnung der Miete" --- Überschrift des Fensters
END

```

## Fahrenheit und Celsius Umrechner

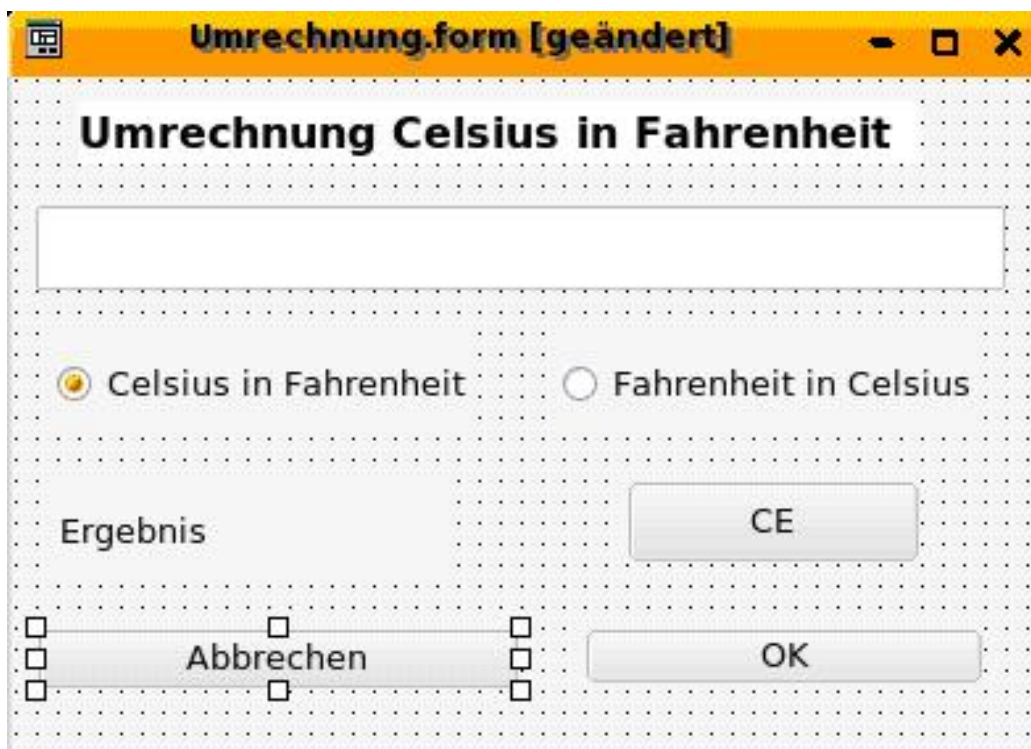


Abbildung 88: Bild:Celsius-Fahrenheit-umrechner.JPG

```

' Gambas class file
' Fahrenheit und Celsius Umrechner
PUBLIC Ergebnis AS Variant
PUBLIC SUB Button1_Click()
  ME.Close
END

PUBLIC SUB Button2_Click()
  IF RadioButton1.Value= TRUE THEN ergebnis= Val(textbox1.Text)*9/5+32
  IF RadioButton2.Value= TRUE THEN ergebnis= (Val(textbox1.Text)-32)/9*5
  Label1.Text=ergebnis
END

PUBLIC SUB Button3_Click()
  textbox1.Text=""
  Label1.Text=""

```

END

Dieses Programm ist ein einfacher Potenzrechner. Es nutzt das mathematische Zeichen  $^$  für die Hochrechnung. Beispiel:  $2^3 = 2 * 2 * 2 = 8$

$2^3 = 8$



Abbildung 89: Bild:Potenzen.JPG

```
' Gambas class file
PUBLIC SUB Button1_Click()
  DIM Pota1 AS Integer
  DIM Pota2 AS Integer
  DIM Pota3 AS Integer

  IF textbox2.Text = 0 THEN
    labell1.Text = 1
  ELSE
    Pota1 = Textbox1.Text
    Pota2 = Textbox2.Text
    Pota3 = Pota1^Pota2
    labell1.Text = Pota3
  END IF
END
PUBLIC SUB Button2_Click()
  Textbox1.Text = ""
  Textbox2.Text = ""
  labell1.Text = ""
END
```

```

PUBLIC SUB Button3_Click()
    ME.Close
END

```

Es folgen zwei Beispiele zur Schleifenprogrammierung an Hand der Zinsrechnung.

## 4.150 Einfache Schleife

```

' Gambas class file
PUBLIC kapital AS Float
PUBLIC zins AS Float
PUBLIC dauer AS Integer
PUBLIC SUB Button3_Click()
    ME.Close
END
PUBLIC SUB Button2_Click()
    Textbox1.text = ""
    Textbox2.text = ""
    Textbox3.text = ""
END
PUBLIC SUB Button1_Click()
    DIM X AS Integer
    kapital=textbox1.Text
    zins=textbox2.Text
    dauer=textbox3.Text
    FOR x = 1 TO dauer STEP 1
        kapital= kapital+(kapital*zins/100)
    NEXT
    label1.Text= "Endkapital " & Round (kapital,-2) & " Euro"
END
PUBLIC SUB Label1_MouseDown()
END
PUBLIC SUB Form_Open()
END

```

## 4.151 Erweitere Schleife

```

PUBLIC euro AS Float
PUBLIC prozent AS Float
PUBLIC SUB Button3_Click()
    ME.Close
END
PUBLIC SUB Button2_Click()
    Textbox1.Text = ""
    Textbox2.Text = ""
END
PUBLIC SUB Button1_Click()
    DIM Jahre AS Integer
    euro=Val(Textbox1.Text)
    prozent=Val(Textbox2.Text)
    DO WHILE euro < 1000000
        euro = euro+(euro*prozent/100)
    LOOP

```

```
    Jahre=Jahre+1
LOOP
textlabel5.Text= "Nach " & Jahre & " Jahre sind sie Millionaer und haben ein Kapital von " & euro
& " Euro."
END
PUBLIC SUB TextBox1_KeyPress()
END
```

## 4.152 Blutgasanalyse

siehe <http://www.madeasy.de/7/ph.htm>

Die Blutgasanalyse ist in der Medizin schnell und wichtig. Wie kann man die Meßwerte interpretieren ? Was noch fehlt ist der Lactatwert und die Hämoglobinwerte.

### Der Programmcode

```
' Gambas class file
PUBLIC SUB Button7_Click()
ME.Close
END
PUBLIC SUB Button2_Click()
textbox1.text = 7
textbox2.text = 30
textbox3.text = -10
textbox4.text = 90
END
PUBLIC SUB Button3_Click()
textbox1.text = "7,6"
textbox2.text = 40
textbox3.text = 10
textbox4.text = 90
END
PUBLIC SUB Button6_Click()
textbox1.text = "7,4"
textbox2.text = 40
textbox3.text = 0
textbox4.text = 90
END
PUBLIC SUB Button4_Click()
textbox1.text = "7,4"
textbox2.text = 30
textbox3.text = 0
textbox4.text = 50
END
PUBLIC SUB Button5_Click()
textbox1.text = "7,1"
textbox2.text = 65
textbox3.text = 0
textbox4.text = 50
END
PUBLIC SUB TextBox1_KeyRelease()
IF key.Code = key.Return THEN textbox2.setfocus
END
PUBLIC SUB TextBox2_KeyRelease()
```



```
IF key.Code = key.Return THEN textbox3.setfocus
END
PUBLIC SUB TextBox3_KeyRelease()
IF key.Code = key.Return THEN textbox4.setfocus
END
PUBLIC SUB TextBox4_KeyRelease()
IF key.Code = key.Return THEN button1.Default = TRUE
IF key.Code = key.Return THEN button1.setfocus
END
PUBLIC SUB Button1_Click()
' befehl3.SetFocus
DIM pc AS Float
DIM po AS Float
DIM be AS Float
DIM ph AS Float
DIM ergeb AS String
DIM b AS String
IF textbox1.text = "" THEN textbox1.text = 0
IF textbox2.text = "" THEN textbox2.text = 0
IF textbox3.text = "" THEN textbox3.text = 0
IF textbox4.text = "" THEN textbox4.text = 0
ph = Val(textbox1.text)
pc = Val(textbox2.text)
be = Val(textbox3.text)
po = Val(textbox4.text)
' PRINT ph,pc,be,po
ergeb = metabol(ph,pc,be)
b = sauerstoff(po,pc)
ergeb = ergeb & Chr$(13) & Chr$(10) & b
textareal.text = ergeb
' PRINT ergeb
END
PUBLIC FUNCTION metabol(ph AS Float,pc AS Float,be AS Float) AS String
' Rem *****
' Rem * AUSSAGE 1 PH,BE,PCO2 *
' Rem *****
DIM a1 AS String
DIM a2 AS String
DIM a3 AS String
DIM a4 AS String
DIM a5 AS String
DIM a6 AS String
DIM a7 AS String
DIM a8 AS String
DIM a9 AS String
DIM a10 AS String
DIM a11 AS String
DIM a12 AS String
DIM e AS String
a1 = "Werte nicht mit dem Leben vereinbar"
a2 = "Metabolische Azidose " & Chr$(13) & Chr$(10) & "respiratorisch kompensiert Hyperventilation
!"
a3 = "Metabolische Azidose nicht kompensiert"
a4 = "Respiratorische Azidose nicht kompensiert"
a5 = "Respiratorische + Metabolische Azidose "
a6 = "Hyperkapnie + Kompensatorische Alkalose"
a7 = "Ausgeglichener Säure Basen Haushalt "
a8 = "Metabolische Alkalose"
a9 = "Hyperventilation"
a10 = "Werte nicht plausibel "
a11 = "Hyperkapnie"
a12 = "metabolische Azidose"
```

```
' Rem * BEDINGUNGEN PH,BE,PCO2 *
IF be > 50 THEN
e = a1
RETURN e
ELSE IF be < -50 THEN
e = a1
RETURN e
ELSE IF ph > 7.9 THEN
e = a1
RETURN e
ELSE IF ph < 6.5 THEN
e = a1
RETURN e
ELSE IF pc < 1 THEN
e = a1
RETURN e
ELSE IF pc > 120 THEN
e = a1
RETURN e
ELSE IF ph < 7.35 AND be > -4 AND pc < 35 THEN
e = a10
RETURN e
ELSE IF ph < 7.45 AND be < -3 AND pc < 35 THEN
e = a2
RETURN e
ELSE IF ph < 7.35 AND be < -3 AND pc > 35 THEN
e = a3
RETURN e
ELSE IF ph < 7.35 AND be < 4 AND pc > 45 THEN
e = a4
RETURN e
ELSE IF ph < 7.35 AND be < -3 AND pc > 45 THEN
e = a5
RETURN e
ELSE IF ph > 7.34 AND be > 3 AND pc > 44 THEN
e = a6
RETURN e
ELSE IF ph < 7.45 AND ph > 7.35 AND be > -5 AND be < 5 AND pc > 35 AND pc < 45 THEN
e = a7
RETURN e
ELSE IF ph > 7.45 AND be > 3 THEN
e = a8
RETURN e
ELSE IF ph > 7.45 AND be < -5 THEN
e = a10
RETURN e
ELSE IF ph < 7.35 AND be < -5 THEN
e = a12
RETURN e
ELSE IF pc < 36 THEN
e = a9
RETURN e
ELSE IF pc > 44 THEN
e = a11
RETURN e
END IF
END
PUBLIC FUNCTION sauerstoff(po AS Float,pc AS Float) AS String
DIM b AS String
' Rem *****
' Rem * PO2 *
' Rem *****
```

```
IF po > 700 THEN
b = "Werte nicht mit dem Leben vereinbar"
RETURN b
ELSE IF pc > 300 THEN
b = "Werte nicht mit dem Leben vereinbar"
RETURN b
ELSE IF po < 1 AND pc < 1 THEN
b = "Werte nicht mit dem Leben vereinbar"
RETURN b
ELSE IF po < 1 THEN
b = "Werte nicht mit dem Leben vereinbar"
RETURN b
ELSE IF pc < 1 THEN
b = "Werte nicht mit dem Leben vereinbar"
RETURN b
ELSE IF po > 100 THEN
b = "Hyperventilation durch Beatmung oder Sauerstoffgabe ??"
RETURN b
ELSE IF po < 55 AND pc < 35 THEN
b = "Hypoxie und Hyperventilation Lungenembolie?"
RETURN b
ELSE IF po < 55 THEN
b = "Hypoxie Beatmungspflichtig ?"
RETURN b
ELSE IF po < 75 THEN
b = "Hypoxie "
RETURN b
ELSE IF pc > 50 THEN
b = "Hyperkapnie "
RETURN b
END IF
END
PUBLIC SUB ToggleButton1_Click()
DIM Help AS String
IF TextArea2.Visible = FALSE THEN
ToggleButton1.Text = "Info weglöschen"
Help = "Autor: " & Chr(13) & Chr(10)
Help = "Rho = Reiner Hoffmann " & Chr(13) & Chr(10)
Help = Help & "www.madeasy.de" & Chr(13) & Chr(10)
TextArea2.Visible = TRUE
TextArea2.text = Help
ELSE
TextArea2.Visible = FALSE
ToggleButton1.Text = "Info anschauen"
ENDIF
END
PUBLIC SUB Form_Open()
textbox1.setfocus
END
```

## 4.153 EKG-Befundung

In der Medizin muß man immer wieder dieselben Textbausteine zu einem Befund zusammenfügen. Um dies zu beschleunigen und einen sauber lesbaren Befund zu bekommen, wurde dieses Programm geschrieben.

**Man braucht dazu:**

- Eine Klasse als Startklasse mit dem Namen t
- 4 Formen

**Code für die Klasse t:**

```
' Gambas class file
STATIC PUBLIC t AS String
STATIC PUBLIC SUB Main()
form1.show
END
```

**Code für die Form1:**

```
PUBLIC SUB Form_Open()
'Clipboard.Clear
' Zwischenablage löschen.
Textarea2.Visible = FALSE
Textarea2.Text = ""
ME.Text = "EKG - Befundung"
' Beim Beenden ist der Text im Zwischenspeicher, Leider nicht
textareal.Text = ""
textareal.Text = t.t
END
PUBLIC SUB Button1_Click()
Clipboard.Copy (Textarea2.Text)
ME.Close
END
PUBLIC SUB Button2_Click()
textareal.text = t.t
Clipboard.Copy (Textarea1.Text)
END
PUBLIC SUB Button24_Click()
t.t = t.t & "Sonst normales RuheEKG"
textareal.Text = t.t
Clipboard.Copy(t.t)
END
PUBLIC SUB Button3_Click()
t.t = t.t & "Sinusrhythmus" & Chr(10)
END
PUBLIC SUB Button10_Click ()
t.t = t.t & "Herzfrequenz 50 - 60/min" & Chr(10)
END
PUBLIC SUB Button11_Click ()
t.t = t.t & "Herzfrequenz 40 - 50/min" & Chr(10)
END
PUBLIC SUB Button12_Click ()
t.t = t.t & "=> Tachyarrhythmie" & Chr(10)
END
PUBLIC SUB Button13_Click ()
t.t = t.t & "=> Bradyarrhythmie" & Chr(10)
END
PUBLIC SUB Button14_Click ()
t.t = t.t & "Normallagetyp, normales Ruhe EKG " & Chr(10)
textareal.Text = t.t
Clipboard.Copy(t.t)
END
PUBLIC SUB Button15_Click ()
t.t = t.t & "=> Tachykardie > 120/min " & Chr(10)
END
```

```
PUBLIC SUB Button16_Click ()
    t.t = t.t & "=> Bradykardie < 30/min !!" & Chr(10)
END
PUBLIC SUB Button17_Click ()
    t.t = t.t & "=> Bradykardie < 40/min" & Chr(10)
END
PUBLIC SUB Button4_Click ()
    t.t = t.t & "Herzfrequenz 60 - 80/min" & Chr(10)
END
PUBLIC SUB Button5_Click ()
    t.t = t.t & "Herzfrequenz 80 - 100/min" & Chr(10)
END
PUBLIC SUB Button6_Click ()
    t.t = t.t & "=> Herzfrequenz 100 - 120/min !!" & Chr(10)
END
PUBLIC SUB Button7_Click ()
    t.t = t.t & "Absolute Arrhythmie" & Chr(10)
END
PUBLIC SUB Button8_Click ()
    t.t = t.t & "Vorhofflimmern" & Chr(10)
END
PUBLIC SUB Button9_Click ()
    t.t = t.t & "Vorhofflattern" & Chr(10)
END
PUBLIC SUB Button19_Click()
    t.t = t.t & "Linkslagetyp, normales Ruhe EKG " & Chr(10)
    textareal.Text = t.t
    Clipboard.Copy(t.t)
END
PUBLIC SUB Button18_Click()
    t.t = t.t & "Steillagetyp, normales Ruhe EKG " & Chr(10)
    textareal.Text = t.t
    Clipboard.Copy(t.t)
END
PUBLIC SUB Button20_Click()
    t.t = t.t & "=> Tachykardie > 150/min " & Chr(10)
END
PUBLIC SUB Button21_Click()
    t.t = t.t & "einzelne Kammerextraschläge" & Chr(10)
END
PUBLIC SUB Button22_Click()
    t.t = t.t & "einzelne Vorhofextraschläge" & Chr(10)
END
PUBLIC SUB Button23_Click()
    ' zur Rhythmusform
END
PUBLIC SUB Button25_Click()
    Textareal.Text = ""
END
PUBLIC SUB Button27_Click()
    t.t = ""
END
PUBLIC SUB ToggleButton1_Click()
    DIM Help AS String
    IF TextArea2.Visible = FALSE THEN
        ToggleButton1.Text = "Info weglöschen"
        Help = "Rho = Reiner Hoffmann " & Chr(13) & Chr(10)
        Help = Help & "www.madeasy.de" & Chr(13) & Chr(10)
        Help = Help & "rho54@gmx.de"
        TextArea2.Visible = TRUE
        TextArea2.text = Help
    ELSE
```

```
    TextArea2.Visible = FALSE
    ToggleButton1.Text = "Info anschauen"
ENDIF
END
PUBLIC SUB Button28_Click()
Textareal.text = Textareal.text & "Dieses Programm wurde mit Gambas erstellt" & Chr(10)
Textareal.text = Textareal.text & "Es dient der EKG Befundung." & Chr(10)
Textareal.text = Textareal.text & "Angeklickte Kürzel werden zusammengeklebt." & Chr(10)
Textareal.text = Textareal.text & "Der Gesamttext wird im Zwischenspeicher abgelegt." & Chr(10)
Textareal.text = Textareal.text & "Mit STRG und V- Taste kann der Text " & Chr(10)
Textareal.text = Textareal.text & "an beliebiger Stelle in jedem anderen Programm eingefügt
werden."
END
PUBLIC SUB Button26_Click()
Form1.Close
Form2.Show
END
```

**4.154 RöntgenThoraxbefundung****4.155 LangzeitEKG-Befundung****4.156 Endoskopiebefundung****4.157 Ultraschallbefundung****4.158 Aufnahmebefund****4.159 Kurzarztentlassbrief****4.160 CT-Befundung****4.161 QT-Zeit****4.162 Umrechnung Herzfrequenz - RR Abstand****4.163 Gambas und SVG**

Gambas sollte Unterstützung für SVG Icons und Grafiken haben. Bis jetzt finde ich nichts.

Nach Angaben im Changelog von Gambas kann das Picture Steuerelement SVG Dateien laden.

*The Picture class can manage SVG files now ! Its interface has gained a Type property and behaves a bit differently.*

Dies kann man mit der Picturebox überprüfen. Es funktioniert nicht, sondern nur für folgende Bildformate: JPEG, PNG, BMP, GIF und XPM. Siehe PictureLoad ( Bild Laden ) [Gambas:\\_Bilder#Laden\\_Sie\\_ein\\_Bild\\_mit\\_Dialog\\_in\\_eine\\_Picturebox](#)

Ein SVG File wird bislang nur als Textfile in eine Textarea geladen.

Vielleicht muß man die Gambas XML Komponenten dazuladen:

- gb.xml.libxml - XML tools based on libxml
- gb.xml.libxml.rpc - XML-RPC client based on libxml and libcurl
- gb.xml.libxml.xslt - XSLT tools based on libxslt

Ich würde die Frage auch hier posten:

- <http://sourceforge.net/mailarchive/forum.php?forum=gambas-user>

### 4.163.1 Quellen

Scalable Vector Graphics – or SVG – is the new XML-based graphics standard from the W3C that will enable Web documents to be smaller, faster and more interactive.

Siehe

- [http://de.wikipedia.org/wiki/Scalable\\_Vector\\_Graphics](http://de.wikipedia.org/wiki/Scalable_Vector_Graphics)
- <http://de.wikipedia.org/wiki/Sodipodi>
- <http://www.adobe.com/svg/viewer/install/main.html>
- <http://www.oreilly.com/catalog/svgess/chapter/ch03.html>
- <http://www.w3.org/TR/SVG/intro.html>
- [http://entwickler.com/itr/online\\_artikel/show.php3?nodeid=97&id=325](http://entwickler.com/itr/online_artikel/show.php3?nodeid=97&id=325)

Mit Gambas werden eine Reihe von Beispielprogrammen installiert. Diese kann man sich im Quelltext anschauen und dadurch über die Programmierung einiges lernen. Diese Programme sind aber für Anfänger meist schon zu schwer. Im folgenden werden diese Beispielprogramm in Miniprogramme zerlegt und kommentiert.

Siehe auch: <http://www.madeasy.de/7/zb gambas.htm>

## 4.164 Blights

Blights ist eine Blinkleiste mit mehreren kleinen Lichtern die per Zufall blinken.



Was kann man aus dem Programm lernen?

#### 4.164.1 Wie bindet man eine Grafik ein?

Dazu ein einfaches Beispiel angelehnt an Blights.

Man startet ein neues Gambasprojekt **blights0** und platziert auf der Form eine PictureBox und 2 Commandbuttons.

Dann gibt man folgenden Code ein:

```
PUBLIC SUB Button1_Click()
  PictureBox1.Picture = Picture["bloff.xpm"]
END

PUBLIC SUB Button2_Click()
  PictureBox1.Picture = Picture["blon.xpm"]
END
```

Startet man das Programm, dann passiert gar nichts, da die 2 Grafiken **bloff.xpm** und **blon.xpm** fehlen. Diese kauft man sich aus Blights und kopiert, sie sich ins eigene Projektverzeichnis.

Starten Sie Ihre Shell. Der Button mit der Muschel.

Gehen Sie in das Verzeichnis des obigen Projektes, etwa mit

```
cd gambas/blights0
```

Versuchen Sie dann zunächst folgendes:

```
locate bloff.xpm
```

Funktioniert dies nicht, weil das Programm `locate` bei ihnen nicht installiert ist, dann geben Sie stattdessen folgendes ein:

```
find / -name bloff.xpm 2>/dev/null
```

(Dabei ist dann allerdings etwas Geduld erforderlich.) Als Ausgabe erhalten Sie schließlich etwas in der Art:

```
/opt/gambas/share/gambas/examples/Basic/Blights/bloff.xpm
/usr/share/doc/packages/gambas/examples/Basic/Blights/bloff.xpm
```

Jetzt markieren Sie eines der Verzeichnisse mit der Maus und können die Dateien kopieren. Dabei hilft Ihnen die [Zwischenablage](#) und das Einfügen mit der mittleren Maustaste bzw. der Rolltaste der Maus.

```
cp /opt/gambas/share/gambas/examples/Basic/Blights/bl* .
```

Sie sich mit einem Malprogramm eigene Bilder erstellen. Ein Trick wäre auch das Clipboard. Man kann die Blightsanzeige mit Strg und der Drucktaste als Screenshot in den Zwischenspeicher laden und dann in einem Malprogramm die Bildchen ausschneiden.

**4.164.2** Wie schaltet man eine Grafik an und aus?

**4.165** Snake

**4.166** Webbrowser

**4.167** Datenbankanbindung

# Kapitel 5

## Codierung

### 5.1 Apostroph als Kommentarzeichen

Ein gutes Programm enthält erklärende Kommentare. Diese sind für andere sehr hilfreich, um ein Programm zu verstehen. Die Kommentare werden bei der Interpretation oder Compilierung des Programmes nicht berücksichtigt. Man kann also in die Kommentare im Klartext alles hineinschreiben, was man will.

Wenn Sie in Gambas eine Zeile als Kommentar eingeben wollen, dann schreiben Sie ganz an den Anfang der Zeile ein **Apostroph Zeichen**. ' Sie finden das Apostrophzeichen links neben der Return Taste. Sie müssen die Rautentaste # und die Hochtaste drücken, um das Apostrophzeichen zu erzeugen. Der Tastencode für das Apostrophzeichen ist Chr(39).

In Wikibooks und Wikipedia wird das Apostrophzeichen für **Fettschrift** oder *Kursivschrift* verwendet.

Bei der Programmierung in Gambas ist im Unterschied zu VB folgendes zu beachten:

- Apostroph funktioniert
- Rem funktioniert nicht

Beispiel:

```
'Das ist ein Kommentar und wird beim Programmablauf nicht berücksichtigt.
```

## 5.2 Programmtests mit Kommentaren

Das Aus- und Einkommentieren von Befehlen ist eine gute Methode, um Programme zu testen und Fehler zu beseitigen.

Sie haben z.B. eine Programmzeile folgender Art geschrieben. (Formel für die Berechnung der Kreisfläche.)

```
F = Pi * r * r
```

Da ihr Programm hier noch einen Fehler macht, kommentieren Sie die Zeile aus und versuchen den Fehler zu finden. Ihr Code bleibt aber erhalten und Sie müssen ihn später nicht noch einmal eingeben.

Im Codefenster kann man auch mit der rechten Maustaste kommentieren und mit den Abkürzungen **Strg + K** und **Strg + U** arbeiten.

Beispielprogramm mit der Kreisberechnung. Man braucht dazu auf der Form:

- 1 Commandbutton
- 2 Textboxen
- 1 Labelfeld

Die Erste und die neunte Zeile sind auskommentiert.

```
' Gambas class file

PUBLIC SUB Form_Open()
  Textbox1.Text = ""
  Textbox2.Text = ""
  Button1.Text = "Kreisberechnung mit Radius:"
  Label1.Text = "Fläche:"
  Label1.Alignment = 66
  'Textausrichtung Rechts
END

PUBLIC SUB Button1_Click()
  DIM r AS Float
  DIM F AS Float

  IF Textbox1.Text = "" THEN
    r = 0
    Textbox1.Text = Str(0)
  ELSE
    r = Val(textbox1.text)
  ENDF

  F = Pi*(r^2)
  Textbox2.Text = Str(Round(F,-2))
END
```

Verändern Sie das Programm so, dass Sie die Zeile

```
F = Pi*(r^2)
```

auskommentieren und dafür

```
F = (Pi*r)^2
```

eingeben.

Welche Version ist richtig? Kann man auf die Klammer in der Zeile verzichten oder nicht?

```
F = Pi*r^2
```

Was bedeutet der Befehl Round in dem Programm? Schauen Sie mit F1 im Alphabetischen Befehlsverzeichnis von Gambas nach. Der Befehl rundet eine Kommazahl auf eine gewünschte Zahl von Stellen ab.

## 5.3 Einfache Meldung

Eine Meldung ist sehr einfach zu programmieren. Geben Sie folgenden Code in eine leere Form ein und starten Sie das Programm.

```
PUBLIC SUB Form_Open()  
    Message("Dies ist eine Meldung", "ButtonText")  
END
```

Alternativ können Sie auch folgenden Code verwenden:

```
PUBLIC SUB Form_Open()  
    Message.Info("Hallo, das ist die Info Meldung", "OK")  
END
```

## 5.4 Meldung mit mehreren Antwortmöglichkeiten

Neben der einfachen Message.Info gibt es vier weitere Meldungsarten:

- Message.Delete (Löschen)
- Message.Error (Fehlermeldung)
- Message.Question (Frage)
- Message.Warning (Warnmeldung)

Bei diesen Meldungsboxen sind bis zu drei Antworten möglich. Außerdem kommt ein Icon zur Anzeige:

- Delete = "Mülltonne",
- Error = "Roter Punkt mit X",

- Question = "?" Fragezeichen,
- Warning = "Gelbes Dreieck mit Ausrufungszeichen"

### 5.4.1 Beispiel

Wie diese Meldungsboxen funktionieren zeigt folgendes Beispiel. Sie brauchen einen Button und ein Labelsteuerelement auf Ihrer Form um das Programm in gang zu bringen. Beides finden Sie in der Werkzeugkiste mit F6.

### 5.4.2 Layout

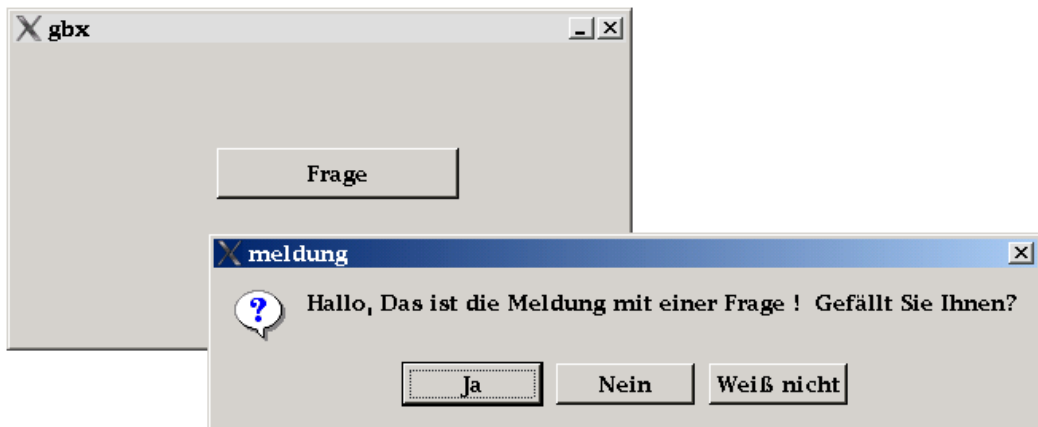


Abbildung 90: Bild:Gambasmeldung.png

### 5.4.3 Code

```
PUBLIC SUB Button1_Click()  
    SELECT Message.Question("Hallo, das ist die Meldung mit einer Frage! Gefällt Sie Ihnen?", "Ja",  
        "Nein", "Weiß nicht")  
        CASE 1  
            Label1.Text = "Erfreulich!"  
        CASE 2  
            Label1.Text = "Bedauerlich!"  
        CASE 3  
            Label1.Text = "Sie werden sie bald mögen!"  
    END SELECT  
END
```

Das Gleichheitszeichen in Basic wird anders verwendet als in der Mathematik. Es wird als Zuweisung für eine Variable verwendet.

Man sollte wenn ein Gleichheitszeichen im Programmcode auftaucht in Gedanken immer sagen:

**Es wird gleichgesetzt.** Oder man sagt **Fülle den Speicherplatz der Variable mit,** statt ist gleich. Dann vermeidet man unlogische Folgerungen.

Folgender Code ist z.B. in Basic korrekt. Probieren Sie es aus. Was ist das Ergebnis?

```
a = 5
a = a * 5
```

Wie das in einem Programm geht, zeigt folgendes Beispiel:

```
PUBLIC SUB Button1_Click()
    DIM a AS Integer
    a = 5
    a = a * 5
    PRINT a
END
```

Einem Mathematiker würden sich die Haare sträuben. Logischer wäre hier die Verwendung von 2 verschiedenen Variablen.

```
PUBLIC SUB Button1_Click()
    DIM a AS Integer
    DIM b AS Integer
    a = 5
    b = a * 5
    PRINT b
END
```

## 5.5 Theorie der Zuweisung (Assignment)

Folgender Code

```
Variable = Expression
```

weist den Wert eines Ausdrucks einem der folgenden Elemente zu:

- Einer lokalen Variablen (A local variable)
- Einem Funktionsparameter (A function parameter)
- Einer Globalen Klassen Variable (A global (class) variable)
- Einem Array (An array slot)
- Einer Öffentlichen Objekt Variablen (An object public variable)
- Einer Objekt Eigenschaft (An object property)

**Beispiele :**

```

iVal = 1972
Name = "Gambas"
hObject.Property = iVal
cCollection[sKey] = Name

```

**5.6 Einfache Datentypen**

In der folgenden Tabelle werden die einfachen Datentypen von Gambas aufgelistet. Die wichtigsten sind Boolean, Integer, Float und String.

Name	Beschreibung	Speicherbedarf	Voreingestellter Wert
Boolean	Wahr oder Falsch	1 byte	False
Byte	0 ... 255	1 byte	0
Short	-32768 ... +32767	2 bytes	0
Integer	-2147483648 ... +2147483647	4 bytes	0
Float	Wie der <i>double</i> Datatyp in <i>C</i>	8 bytes	0.0
Date	Datum und Zeit, gespeichert als <i>integer</i>	8 bytes	Null
String	Ein beliebige Folge von Buchstaben.	4 bytes	Null
Variant	Jede Art von Da- tatyp	12 bytes	Null
Object	Eine anonyme Referenz auf ein Objekt	4 bytes	Null



## 5.7 Zusammengesetzte Datentypen

### 5.7.1 Klassen

### 5.7.2 Objekte

### 5.7.3 Arrays

Siehe [Gambas: Arrays](#)

### 5.7.4 Collections

## 5.8 Umwandlung von Datentypen

## 5.9 Ganz einfach

Da den meisten Programmierern die OOP beim ersten Kontakt erhebliche Probleme bereitet, sollte man ein ganz einfaches Beispiel zum Verständnis wählen:

Bei der Programmierung von [Steuerelementen](#) für eine grafische Bedienoberfläche werden beispielsweise die Schaltflächen (Befehlsknöpfe) als Objekte einer allgemeinen Klasse Schaltfläche gehandhabt.

- Klasse **Button**
  - Objekte
    - \* **Button1**
    - \* **Button2**
    - \* etc

Sie arbeiten also meist schon mit Klassen und Objekten, ohne daß sie es merken.

## 5.10 Links

### 5.10.1 Gambas

- <http://www.madeasy.de/7/prgmini.htm#klasse>

### 5.10.2 Allgemein

- <http://sepwww.stanford.edu/sep/josman/oop/oop1.htm>

## 5.11 Variablen

### 5.11.1 Was sind Variablen?

Eine Variable ist eine Größe, die verschiedene Werte annehmen kann. Sie ist also in ihrer Größe veränderlich. Variablen werden auch **Platzhalter** oder **Unbekannte** genannt. Sie kommen in Formeln und Termen vor.

Das Gegenteil einer Variablen ist die Konstante, also ein fester, unveränderlicher Wert. Die Kreiszahl  $\pi = 3,14\dots$  ist beispielsweise keine Variable, sondern eine Konstante.

Beispiele für Variable:

- $x, y, z$  im kartesischen Koordinatensystem
- $y = x^2$
- Temperatur  $t$
- Satz B

Für den Programmierer ist eine Variable ein Speicherbereich, auf den er (unter Beachtung des Datentyps) über einen eindeutigen Bezeichner (Namen) zugreifen kann. Das Variablen-Konzept ist der Kern der Imperativen Programmierung. Eine Variable muss in Gambas ausdrücklich deklariert werden.

## 5.11.2 Deklaration (Dimensionierung, Zuweisung des Datentyps)

Es gibt zwei Haupttypen von Information, die in einer Variablen gespeichert werden kann: Zahlen und Text. Bevor eine Variable benutzt werden kann, muß sie zuerst erstellt werden. Das erfolgt mit dem Befehl DIM AS .

### Beispiele für die Deklaration von Variablen:

```
DIM x AS integer
DIM y AS float
DIM z AS String
```

Man kann in einem Unterprogramm nicht dieselbe Bezeichnung für 2 verschiedene Variablen heranziehen.

Man kann Dim auch einfach weglassen:

```
x AS integer
y AS float
z AS String
```

Es gibt bisher keine Möglichkeit mehr als eine Variable in einer Zeile zu deklarieren:

```
intA, intB, intC AS Integer
```

Ist leider nicht möglich. Außerdem erfolgen die Deklarationen immer am Anfang eines Unterprogrammes und nicht irgendwo wild im Code. Benoit ist da ganz streng. Die korrekte Deklaration von Variablen zeigt folgendes Miniprogramm

```
PUBLIC SUB Button1_Click()
'Deklaration
x AS integer
y AS float
z AS String
'Zuweisung von Werten
x = 2
y = 2.378
z = "Das ist eine korrekte Deklaration"
'Benutzung der Variablen
print x,y,z
END
```

Folgendes Programm dagegen ergibt eine Fehlermeldung: x already declared

```
PUBLIC SUB Button1_Click()
DIM x AS integer
DIM x AS float
DIM x AS String
x = 2
x = 2.378
y = "Das ist keine korrekte Deklaration"
print x,x,x
END
```

Auch im folgenden Programm gibt es Ärger, da zwischen kleinen und großen Buchstaben nicht unterschieden wird:

```
PUBLIC SUB Button1_Click()  
DIM x AS integer  
DIM X AS float  
DIM x$ AS String  
x = 2  
X = 2.378  
x$ = "Das ist eine korrekte Deklaration"  
print x,X,x$  
END
```

Die Deklaration von x\$ ist korrekt.

### 5.11.3 Assignment (Wertzuweisung)

Sobald eine neue Variable erstellt worden ist, kann ihr ein Wert zum Speichern zugewiesen werden. Um dies zu tun wird der Befehl = verwendet. Die Wertzuweisung wird im Englischen Assignment genannt.

Das Beispiel weist einer Variable die als Gewicht bezeichnet einen Wert zu.

Beispiel :

```
Gewicht = 80
```

Bei der Wertzuweisung muss man eine bestimmte Reihenfolge einhalten. Erst werden die Variablen deklariert, dann kann Ihnen ein Wert zugewiesen werden:

Variablen müssen am Anfang einer Klasse, Methode oder Funktion festgelegt werden.

Beispiel korrekt:

```
PUBLIC SUB bla()  
gewicht AS Integer  
groesse AS Integer  
gewicht = 75  
groesse = 176  
END
```

Beispiel FALSCH !! (falsche Reihenfolge)

```
PUBLIC SUB bla()  
gewicht AS Integer  
gewicht = 75  
groesse AS Integer  
groesse = 176  
END
```

Das letzte Beispiel ergibt die Fehlermeldung: *Unknown identifier,groesse Zeile: xx in Form1.class*

## 5.11.4 Wie kann man Variablen begrenzen

Public, Private, Static

Globale Variablen soll es nicht geben. Optional wird nur bei der Methoden Deklaration verwendet.

## 5.11.5 Public - Private

Geben Sie folgendes kleine Programm ein und schauen Sie zu, was passiert, wenn Sie auf den Button1 klicken:

```
PRIVATE SUB Button1_Click()
    DIM Zahl AS Integer
    'Erstellen einer privaten lokalen Variable
    Zahl = Zahl + 1
    TextBox1.Text = TextBox1.Text & Zahl
END
```

Erstaunlicherweise wird in der Textbox kein Wert ausgegeben. Das liegt an der Kennzeichnung des Unterprogrammes mit **PRIVATE**. Tauschen Sie den Begriff **PRIVATE** mit **PUBLIC** aus und versuchen Sie Ihr Glück noch einmal:

```
PUBLIC SUB Button1_Click()
    DIM Zahl AS Integer
    'Erstellen einer oeffentlichen lokalen Variable
    Zahl = Zahl + 1
    TextBox1.Text = TextBox1.Text & Zahl
END
```

Jetzt wird eine 1 in der Textbox ausgegeben. Wenn Sie mehrmals den Button anklicken, erscheint die 1 mehrfach.

## 5.11.6 Variablen allgemein definieren

Variablen lassen sich auch außerhalb von Funktionen deklarieren, hier ein Beispiel:

```
PUBLIC meldung AS String
PUBLIC SUB Form_Open()
    meldung = "Dies ist ein Test"
END SUB
PUBLIC SUB Button1_Click()
    Message.Info(meldung)
END
```

Sie brauchen einen Commandbutton auf Ihrer Form, um das Beispiel in Gang zu bringen.

Form\_Open ist ein Ereignis wie Button1\_Click. Der Code zu diesem Ereignis wird bereits beim Programmstart aufgerufen, wenn die Form geladen und geöffnet.

net wird. Hier wird die Variable mit Public außerhalb eines Unterprogrammes deklariert. Diese Variable ist also so lange verfügbar, wie die Formular geladen ist.

Ersetzen Sie den Begriff *PUBLIC* durch den Begriff *PRIVATE* und probieren Sie das Ergebnis aus.

```
PRIVATE meldung AS String
PRIVATE SUB Form_Open()
    meldung = "Dies ist ein Test"
END SUB
PRIVATE SUB Button1_Click()
    Message.Info(meldung)
END
```

### 5.11.7 Static (Statische Variable)

Manchmal soll der Wert einer Variablen erhalten bleiben. Ruft man die Variable wieder auf, kann mit dem gemerkten Wert der Variablen weiter gearbeitet werden. Um dies zu ermöglichen gibt es den Deklarationsbefehl Static.

Wie er funktioniert zeigt folgendes Beispiel. Sie brauchen einen Befehlsknopf auf Ihrer Form um es in Gang zu bringen.

```
Static Public x As integer
Public Sub Button1_Click()
    x = x + 1
    print x
End
```

Drückt man den Befehlsknopf wird erst x als 0 definiert. Dann wird in der Zeile `x = x + 1` zur 0 die 1 dazugezählt. Mit `Print x` wird das Ganze im Direktfenster ausgegeben. Drückt man den Befehlsknopf mehrfach, dann erkennt man, daß das Programm sich den Wert für x gemerkt hat.

Ausgabe im Direktfenster:

```
1
2
3
4
5
```

Das Wort Static sorgt dafür, daß der Wert einer Variablen nicht einfach verschwindet. Wird sie ein zweites Mal aufgerufen, steht automatisch der alte Wert wieder zur Verfügung.

Versuchen Sie folgende Variante:

```
Public Sub Button1_Click()
    x As integer
    x = x + 1
```

```
print x
End
```

Drücken Sie den Befehlsknopf mehrfach. Welche Ausgabe ergibt sich?

```
1
1
1
1
```

Die Unterfunktion setzt den Wert der Variablen immer wieder auf Null zurück. Beachten Sie das die statische Variable im Gegensatz zu Visual Basic außerhalb des Unterprogrammes definiert werden muß.

Wären Sie enttäuscht, wenn das ganze auch ohne Static und Public funktioniert? Probieren Sie folgenden Code aus:

```
x As integer
Public Sub Button1_Click()
    x = x + 1
    print x
End
```

## 5.11.8 Globale Variablen in Gambas

Benoit Minsini mag sie nicht. Aber sie sind möglich und m.E. auch ganz praktisch.

Siehe [Gambas:\\_Form#Globale\\_Variablen\\_in\\_Gambas](#)

Eine andere Möglichkeit über den Umgang mit globalen Variablen zeigen die nachstehenden Beispiele:

In einem umfangreichen Gambas Projekt werden mehrere Formen und Module verwendet. Globale Variablen, die für das gesamte Projekt gelten, gibt es in Gambas nicht. Es ist jedoch sinnvoll von Formen oder Modulen auf die Variablen anderer Formen oder Module zugreifen zu können. Wie das funktioniert wird nachstehend gezeigt. Wir benötigen eine Form mit einem Befehlsknopf und einem Textfeld als Ergebnisanzeige. Des weiteren benötigen wir ein Modul, in dem mit den Variablen aus der Form gerechnet wird.

Auslesen von öffentlichen Variablen in einer Form1 aus einer Prozedur in einem Modul.

Programmcode der Form1:

```
PUBLIC a AS Float
PUBLIC b AS Float
' Die Variablen sind öffentlich und sind gültig in allen Prozeduren und Funktionen in der Form1.

PUBLIC SUB Button1_Click()
```

```
DIM c AS Float
ME.Text ="Beispiele"
a = 3.2
b = 7.4
c = Module1.RechnenVariable()
textbox1.Text = Str(c)
END
```

### Programmcode von Module1:

```
PUBLIC FUNCTION RechenVariable() AS Float
DIM c AS Float
c = form1.a + form1.b
RETURN c
END
```

Bei Betätigung der Befehlsschaltfläche werden zunächst den Variablen a und b Werte zugewiesen. Mit dem Aufruf der Function RechenVariable in Module1 werden die Werte der Variablen a und b der Form1 ausgelesen, dann addiert und als Rückgabewert der Function wieder an die aufrufende Prozedur Sub Button1 zurückgegeben. Das Ergebnis wird in der Textbox1 als Float-Variable ausgegeben.

Genau so einfach wie das Auslesen der Variablen ist das Zuweisen von Werten für die Variablen a und b der Form1 über eine Prozedur von Module1.

### Programmcode der Form1:

```
PUBLIC a AS Float
PUBLIC b AS Float
PUBLIC SUB Button1_Click()
ME.Text ="Beispiele"
Module1.RechenVariable()
textbox1.Text = Str(a + b)
END
```

### Programmcode von Module1

```
PUBLIC SUB RechenVariable()
form1.a = 10.2
form1.b = 2.5
END
```

Etwas komplizierter ist das Auslesen von Variablen aus einem Array, da das Ansprechen der Arrayvariable aus dem Modul mit dem Code `c = form1.a[5]` nicht funktioniert. Daher wird in Form1 eine öffentliche Prozedur programmiert, die auf das Array zugreifen kann. Zum Auslesen des Array wird von dem Modul auf diese öffentliche Prozedur zugegriffen. Hier ist der Code für die Form1:

```
d[10] AS Float
' das Array wird ohne PUBLIC deklariert. Mit Public d[10] AS Float gibt Gambas eine Fehlermeldung aus.
' ein Array kann nur im Kopf von Formen deklariert werden. Bei der Deklaration im Kopf von
' Modulen gibt Gambas ebenfalls eine Fehlermeldung aus.
' Die Zählung für den Arrayinhalt beginnt bei 0 (d[0] = ..), während die Zählung für die
' Deklaration bei 1 beginnt
' Das Array ist öffentlich und ist gültig in allen Prozeduren und Funktionen der Form1.
PUBLIC SUB Form_Open()
```



```

DIM i AS Integer
ME.Text ="Beispiele"
FOR i = 0 TO 9
    d[i] = i * 1.2    'Array wird mit willkürlichen Werten gefüllt.
NEXT
END
PUBLIC FUNCTION d_aus (x AS Integer) AS Float
    DIM wert AS Float
    wert = d[x]
    RETURN wert
END
PUBLIC SUB Button1_Click()
    DIM c AS Float
    c = Module1.RechnenArray()
    textbox1.Text = Str(c)
END

```

### Programmcode von Module1:

```

PUBLIC FUNCTION  RechnenArray() AS Float
    DIM c AS Float
    c = form1.d_aus(2)  + form1.d_aus(5)
    RETURN c
END

```

Zunächst wird in der Prozedur `Open()` das Array mit Werten gefüllt. Mit dem Aufruf der Funktion `RechenArray` in `Module1` wird zweimal die Funktion `d_aus` in der Form 1 aufgerufen und die Inhalte von `d[2]` bzw. `d[5]` ermittelt und an die Funktion `Rechenarray` zurückgegeben. Dort werden sie dann addiert und als Rückgabewert der Funktion `Rechenarray` wieder an die aufrufende Prozedur `Sub Button1` zurückgegeben. Das Ergebnis wird in der `Textbox1` als `Float-Variable` ausgegeben.

Das Zuweisen von Werten für das Array erfolgt ähnlich. Anstatt Funktionen werden in diesem Beispiel Prozeduren benötigt, da die Werte übergeben und nicht ausgelesen werden. Der dafür erforderliche Code für die `Form1` lautet:

```

d[10] AS Float
PUBLIC SUB Form_Open()
    DIM i AS Integer
    ME.Text ="Beispiele"
    FOR i = 0 TO 9
        d[i] = i * 1.2    'Array wird mit willkürlichen Werten gefüllt.
    NEXT
END
PUBLIC SUB d_ein (wert AS Float, x AS Integer)
    d[x] = wert
END
PUBLIC SUB Button1_Click()
    DIM c AS Float
    Module1.RechnenArray()
    textbox1.Text = Str(d[5])
END

```

### Programmcode von Module1:

```

PUBLIC SUB  RechnenArray()
    DIM wert AS Float
    wert = 12.7

```

```
form1.d_ein (wert, 5) '5 ist willkürlich gewählt
END
```

**Bei einem zweidimensionalen Array lautet der Code der Form1 für das Auslesen von Werten:**

```
d[10,5] AS Float
PUBLIC SUB Form_Open()
  DIM i AS Integer
  DIM j AS Integer
  ME.Text ="Beispiele"
  FOR i = 0 TO 9
    FOR j = 0 TO 4
      d[i,j] = i * 1.2 + j 'Array wird mit willkürlichen Werten gefüllt.
    NEXT
  NEXT
END
PUBLIC FUNCTION d_aus (x AS Integer, y AS Integer) AS Float
  DIM wert AS Float
  wert = d[x,y]
  RETURN wert
END
PUBLIC SUB Button1_Click()
  DIM c AS Float
  c = Module1.RechnenArray()
  textbox1.Text = Str(c)
END
```

**Programmcode von Module1:**

```
PUBLIC FUNCTION RechnenArray() AS Float
  DIM c AS Float
  c = form1.d_aus(5,2) + form1.d_aus(8,4)
  RETURN c
END
```

**Bei einem zweidimensionalen Array lautet der Code der Form1 für die Übergabe von Werten:**

```
d[10,5] AS Float
PUBLIC SUB Form_Open()
  DIM i AS Integer
  DIM j AS Integer
  ME.Text ="Beispiele"
  FOR i = 0 TO 9
    FOR j = 0 TO 4
      d[i,j] = i * 1.2 + j 'Array wird mit willkürlichen Werten gefüllt.
    NEXT
  NEXT
END
PUBLIC SUB d_ein (wert AS Float, x AS Integer, y AS Integer)
  d[x,y] = wert
END
PUBLIC SUB Button1_Click()
  Module1.RechnenArray()
  textbox1.Text = Str(d[5,2])
END
```

**Programmcode von Module1:**

```
PUBLIC SUB RechnenArray()
  DIM wert AS Float
  wert = 12.7
```

```
form1.d_ein (wert, 5,2) '5 und 2 sind willkürlich gewählt
END
```

In allen Beispielen wurde mit Fließkommazahlen experimentiert. Durch geringfügige Änderungen der Prozeduren und Funktionen kann natürlich auch mit Integervariablen, mit Strings, mit Datevariablen oder auch mit Booleanvariablen gearbeitet werden. Es ist auch nicht kompliziert auf die Inhalte von Textboxen zuzugreifen, die Textboxen mit Inhalten zu füllen oder auch Labeltexte auszulesen oder zu ändern.

Im folgenden werden einige Befehle beschrieben, mit denen man Variablen von einer Formatierung in eine andere umwandeln kann. Wichtig sind vor allem die Befehle Val,Str,Chr,Asc und Int.

## 5.12 val

Wie man Strings (Textfolgen) mit Val() umwandelt.

Der Befehl Val() wandelt einen String (= Folge von Asciizeichen) in einen Datentyp um, der dem Inhalt des Strings entspricht.

Wenn der String aussieht wie eine Fließkommazahl, dann macht Val() eine Fließkommazahl daraus. Val() benutzt die lokale Sprache ihrer Maschine (hier also deutsch), die in "locales" festgelegt wurde. Sie können Ihre Einstellung mit folgendem Konsolenbefehl überprüfen:

```
"locales" oder "echo $LANG".
```

Auf einer deutschen Maschine erhält man : de\_DE@euro Vorsicht: Mit Kommazahlen und der Ländereinstellung !! Steht in einer Textbox1 6,6 mit deutscher Ländereinstellung, dann liefert

```
val (textbox1.text)
```

eine Floatvariable mit dem Wert 6.6

Steht in der Textbox 6.6 dann liefert val(textbox1.text) ein Datum 06.06.04 Wollen Sie die Zahl wieder in einen deutschen Text verwandeln, dann nutzen Sie str(zahl) zum umwandeln.

Das Programm

Wenn Sie einen string eingeben dann wandelt das Programm den String mit val um und die Art der umgewandelten Variable wird überprüft.

Das Ergebnis wird ausgegeben.

Um das Programm in Gang zu bringen brauchen Sie auf Ihrer Form:

- 1 Command Button
- 1 Textlabelfeld
- 1 Textbox
- 1 Labelfeld

Holen Sie sich diese mit F6 aus dem Werkzeugkasten und platzieren Sie diese auf der Form.

Der Code:

```
PUBLIC SUB _new()  
TextLabel1.Text="Tippen Sie etwas ein das wie eine Zahl oder eine Kommazahl oder eine Boolesche  
Variable  
ausschaut"  
END  
PUBLIC SUB Button1_Click()  
x AS Variant  
x = Val(TextBox1.Text)  
IF IsBoolean(x) THEN  
    Label1.Text="Das ist eine Boolesche Variable!"  
ENDIF  
IF IsInteger(x) THEN  
    Label1.Text="Das ist eine Zahl!"  
ENDIF  
IF IsFloat(x) THEN  
    Label1.Text="Das ist eine Kommazahl!"  
ENDIF  
IF IsString(x) THEN  
    Label1.Text="Das ist ein String!"  
ENDIF  
END
```

Theorie Expression = Val ( String ) Converts a string into a boolean, a number or a date, according to the content of the string. The current localization is used to convert numbers and dates. The conversion algorithm is the following : If the string can be interpreted as a date & time (with date or time separators), then the date & time is returned. Else, if the string can be interpreted as a floating point number, then this floating point number is returned. Else, if the string can be interpreted as a integer number, then this integer number is returned. Else, if the string is TRUE or FALSE , then the matching boolean value is returned. Otherwise, NULL is returned.

Beispiel

```
PRINT Val("09/06/72 01:00")
```

=> 09/06/72 01:00:00

```
PRINT Val("3.1415")
```

=> 3.1415 oder 31415 je nach Ländereinstellung

```
PRINT Val("6.6")
```

=> 6.6 oder 06.06.04 je nach Ländereinstellung !!

```
PRINT Val("-25")
```

=> -25

```
PRINT Val("True")
```

=> True

```
PRINT IsNull(Val("Gambas"))
```

=> True

## 5.13 str\$

Str ist das genaue Gegenteil des Val Befehles. Str wandelt also alles in einen String um. Wenn Sie zb eine Zahl errechnet haben zb die Wurzel aus 3 und sie möchten diese in einer Textbox ausgeben , dann können Sie den Str Befehl nutzen:

```
Textbox1.Text = str$(sqr(3))
```

Vorsicht: Auch der Str Befehl nutzt wieder die lokalen Einstellungen ihres PCs , was Anlass für Fehler sein kann.

Schauen Sie sich das folgende einfache Beispiel an. Sie brauchen eine Textbox und einen Commandbutton auf Ihrer Form um es in Gang zu bringen:

```
PUBLIC SUB Button1_Click()
    textbox1.Text = Str(Sqr(3))
END
```

Der Befehl sqr zieht die Wurzel aus einer Zahl. Statt der 3 in der Klammer nach Sqr können Sie jede andere positive Zahl oder 0 eingeben. Wenn Sie eine negative Zahl eingeben , dann erscheint eine Fehlermeldung: Mathematic Error

Ein weiteres Beispiel:

```
PUBLIC SUB Button1_Click()
    textbox1.Text = Str( 8 * 37)
END
```

Das Sternchen entspricht dem deutschen Zeichen für Malnehmen ( Multiplikation)

Oder probieren Sie einmal folgenden Code:

```
PUBLIC SUB Button1_Click()
    textbox1.Text = Str(Pi)
END
```

In den Beispielen mit den Rechentrainern wird der Val und der Str Befehl mehrfach genutzt.

Siehe [Gambas: Rechentrainer](#)

Das Beispiel aus der Hilfefunktion von Gambas ist für Anfänger ziemlich unverständlich:

```
' Ausgabe in der Standard Ausgabe oder als Meldung
PUBLIC CONST ON_STDOUT AS Integer = 1
PUBLIC CONST ON_MESSAGE AS Integer = 2
SUB PrintOn(Where AS Integer, What AS Variant)
  IF Where = ON_STDOUT THEN
    PRINT What
  ELSE IF Where = ON_MESSAGE THEN
    Message(Str$(What))
  ENDF
END
```

## 5.14 asc

Asc gibt den ASCII code eines Buchstabens oder Zeichens zurück. Wenn die Position nicht angegeben wird, dann wird der ASCII code des ersten Zeichens zurückgegeben.

Syntax = Asc ( String [ , Position ] )

Beispiel:

```
PRINT Asc("Gambas")
```

=> Ausgabe 71

```
PRINT Asc("Gambas", 3)
```

=> Ausgabe 109

Beispiel 2 Sie brauchen einen Befehlsbutton um es in Gang zu bringen. Die Ausgabe erfolgt mit Print im Direktfenster.

```
PUBLIC SUB Button1_Click()
  x as integer
  For x = 1 to 6
    PRINT Asc("Messer", x)
  next
END
```

Ein Beispiel im Terminalmodus :

```
STATIC PUBLIC SUB Main()
test AS String
x AS Integer
test = "1234567 Dies ist ein Test"
FOR x = 0 TO Len(test)
  PRINT Asc(test,x)
NEXT
END
```

## 5.15 chr\$

Dieser Befehl gibt das Ascii Zeichen zurück, welches der Zahl in Klammern entspricht.

```
Zeichen = Chr$ ( Zahl )
```

Vorsicht ! Gambas nutzt intern das UTF-8 charset, so daß Zahlen > 128 andere Zeichen zurückgeben wie mit dem ISO8859-1 charset zu erwarten sind.

Beispiel:

```
PRINT Chr$(65)
```

==> A

Weiteres Beispiel, Sie brauchen einen Befehlsbutton um es in gang zu bringen. Die Ausgabe erfolgt mit Print im Direktfenster.

```
PUBLIC SUB Button1_Click()
  X AS Integer
  FOR X = 32 TO 127
    PRINT Chr$(X);
  NEXT
END
```

Mit einer Schleife werden alle Zeichen zwischen 32 und 127 hintereinander in einer Zeile ausgegeben. Siehe <http://www.madeasy.de/7/prgansi.htm>

Im reinen Terminalmodus können Sie folgendes Programm als Beispiel ausprobieren:

```
STATIC PUBLIC SUB Main()
  x AS Integer
  FOR x = 30 TO 133
    PRINT x,Chr(x)
  NEXT
  PRINT
  PRINT "Zeilenumbruch CHR(10)", Chr$(10)
  PRINT "Return CHR(13)", Chr$(13)
  PRINT "Leerzeichen Chr$(32)", "a";Chr$(32);"a"
  PRINT "0 CHR48", Chr$(48), Str(0)
  PRINT "9 CHR57", Chr$(57), Str(9)
  PRINT "@ CHR64", Chr$(64), Str("@")
  PRINT "A CHR65", Chr$(65), Str("A")
  PRINT "Z CHR90", Chr$(90), Str("Z")
  PRINT "a CHR97", Chr$(97), Str("a")
  PRINT "z CHR122", Chr$(122), Str("z")
  PRINT "A" + "A"
  PRINT "A" & "A"
END
```

## 5.16 int

Der Befehl Int ( Integer = ganze Zahl) gibt die ganze Zahl zurück, die unterhalb einer Kommazahl steht.

```
3,1 ==> 3
```

Vergleiche auch den Befehl Fix Der Befehl wird gerne beim Runden benutzt.

### Beispiel

```
PRINT Int (Pi)
```

```
=> 3
```

```
PRINT Int (-Pi)
```

```
=> -4
```

**Beispiel:** Sie brauchen einen Commandbutton. Die Ausgabe erfolgt im Direktfenster

```
PUBLIC SUB Button1_Click()  
x AS Integer  
a AS Float  
a = -3.456  
FOR x = 1 TO 10  
  a = a + 1  
  PRINT a, Int(a)  
NEXT  
END
```

Beachten Sie, daß die Kommazahl hier im Code in der englischen Notation mit einem Punkt eingegeben werden muß !

```
a = -3.456
```

### Ergebnis :

```
-2,456 -3  
-1,456 -2  
-0,456 -1  
0,544 0  
1,544 1  
2,544 2  
3,544 3  
4,544 4  
5,544 5  
6,544 6
```

## 5.17 fix

Der Befehl fix liefert den Ganzzahlteil einer Kommazahl. Alles was vor dem Komma steht bleibt erhalten, was hinter dem Komma steht wird weggeschnitten. Vergleichen Sie auch den Befehl Int.



**Syntax:**

```
Value = Fix ( Number )
```

**Einfache Beispiele :**

```
PRINT Fix(Pi)
```

=> 3

```
PRINT Fix(-Pi)
```

=> -3

**Programmbeispiel: Sie brauchen einen Commandbutton um es in Gang zu setzen.**

```
PUBLIC SUB Button1_Click()
x AS Integer
a AS Float
a = -3.456
FOR x = 1 TO 10
  a = a + 1
  PRINT a, Fix(a)
NEXT
END
```

**Ergebnisabgabe im Direktfenster:**

```
-2,456 -2
-1,456 -1
-0,456 0
0,544 0
1,544 1
2,544 2
3,544 3
4,544 4
5,544 5
6,544 6
```

## 5.18 sgn

Die Signum Funktion liefert einen Wert, der für das Vorzeichen einer Zahl steht.

**Syntax Sgn(Zahl)**

**Anmerkungen** Das Argument Zahl kann ein beliebiger zulässiger numerischer Ausdruck sein. Dessen Vorzeichen bestimmt den von der Sgn-Funktion ausgegebenen Wert:

- Ist die Zahl > 0 liefert Sgn(Zahl) den Wert 1.
- Ist die Zahl = 0 liefert Sgn(Zahl) den Wert 0.
- Ist die Zahl < 0 liefert Sgn(Zahl) den Wert -1.

**Beispiel: Sie brauchen dafür nur eine leere Form. Die Ergebnisausgabe erfolgt im Direktfenster.**

```
PUBLIC SUB Form_Open()  
x AS Integer  
y AS Float  
FOR x = 1 TO 100  
  y = Rnd(-5, 5)  
  SELECT CASE Sgn(y) ' ...auswerten.  
    CASE 0 ' Wenn Zahl null.  
      PRINT y & " " & Sgn(y) & " Signum = Null."  
    CASE 1 ' Wenn Zahl positiv.  
      PRINT y & " " & Sgn(y) & "Signum = positive Zahl."  
    CASE -1 ' Wenn Zahl negativ.  
      PRINT y & " " & Sgn(y) & "Signum = negative Zahl."  
  END SELECT  
NEXT  
END
```

**Beispiel 2**

## 5.19 bst

## 5.20 cbool

Dieser Befehl wandelt einen Ausdruck in eine Boolesche Variabel ( wahr,falsch) um.

Syntax;

Boolean = CBool (Ausdruck)

Das Ergebnis wird falsch, falls der Ausdruck:

- eine Boolesche Variable mit dem Wert falsch ist.
- eine Null ist.
- ein Textstring mit Länge Null ist zb a = ""
- ein Null Objekt ist.

In allen anderen Fällen ist der Ausdruck wahr. Auch ein Leerzeichen als Textstring ergibt mit CBool umgewandelt den Wert True !

Beispiel :

```
PRINT CBool(0); " "; CBool(1)
```

=> False True "" funktioniert hier nur als Abstandshalter bei der Ergebnisausgabe.

```
PRINT CBool("Gambas"); " "; CBool(""), " "; CBool(" ")
```

=> True False True

```
PRINT CBool(NULL)
```

=> False

## 5.21 cbyte

## 5.22 cdate

## 5.23 cfloat

## 5.24 cstr

## 5.25 cint

## 5.26 Einfache Arrays

Siehe auch: <http://www.madeasy.de/7/prgmini.htm#arr>

Arrays sind **Listen oder Tabellen, in die man zusammengehörige Texte oder Werte vom selben Datentyp** abspeichert. Jedes Element des Arrays ist einzeln ansprechbar.

Beispiel eines Arrays: Liste von 5 Vornamen

- Anna
- Anton
- Toni
- Max
- Klaus

Um Arrays zu definieren, reicht kein einfaches Ist-Gleich-Zeichen, sondern man braucht eine besondere Deklaration: Um eine Namensliste , wie die obige, in Gambas zu bekommen, nimmt man am besten einen Stringarray (Textarray):

**Beispiel:**

```
PUBLIC SUB Form_Open()  
a AS String[]  
x AS Integer  
a = NEW String[]  
FOR x = 0 TO 4  
    a.Add(Str(x))  
NEXT  
a[0] = "Anna"  
a[1] = "Anton"  
a[2] = "Toni"  
a[3] = "Max"  
a[4] = "Klaus"  
FOR x = 0 TO 4  
    PRINT a[x]  
NEXT  
END
```

Um dieses Beispielprogramm in Gang zu bringen brauchen Sie nur eine leere Form. Das Programm gliedert sich in 4 Teile:

- 1. Ein Stringarray wird deklariert.

```
a AS String[]
```

- 2. Der Stringarray wird initialisiert.

```
DIM x AS Integer  
a = NEW String[]  
FOR x = 0 TO 4  
    a.Add(Str(x))  
NEXT
```

- 3. Der Stringarray wird mit echten Elementen gefüllt:

```
a[0] = "Anna"  
a[1] = "Anton"  
a[2] = "Toni"  
a[3] = "Max"  
a[4] = "Klaus"
```

- 4. Um zu überprüfen, ob die Elemente wirklich im Array gespeichert wurden, werden sie dann wieder mit einer Schleife und dem Printbefehl im Direktfenster ausgegeben.

```
FOR x = 0 TO 4  
    PRINT a[x]  
NEXT
```

Das erscheint reichlich kompliziert oder was meinen Sie? Es geht auch etwas einfacher, wenn man den Array Befehl nutzt. Dies ist im nächsten Kapitel beschrieben.

Erstaunlicherweise funktioniert das obige Programm nur im Grafikmodus. Aber vielleicht fehlt mir auch nur ein Trick um es im Terminalmodus zu ermöglichen.

Außerdem ist die Deklaration ziemlich umständlich. `a = NEW String[5]` funktioniert nicht.

Man kann die einzelnen Elemente über ihre Position im Array ansprechen, wobei das erste Element die Position 0 hat:

Benennung der Einträge

```
a[0] = "Anna"  
a[1] = "Anton"  
a[2] = "Toni"  
a[3] = "Max"  
a[4] = "Klaus"
```

Um den Namen Max ausgeben zu lassen, muß man dann schreiben:

```
print a[3]
```

Beachten Sie bitte das **a** kein Element von Gambas ist , sondern oben im Beispielprogramm als Array definiert wurde.

Die eckigen Klammern sind hier nicht wie üblich bei Computertexten optionale Angaben, sondern man muß **bei Arrays wirklich eckige Klammern verwenden**.

Der **Index eines Arrays beginnt immer bei 0**, somit ergeben sich die gültigen Indizes bei einem Array der Größe n: 0, 1, 2, ..., n-1.

Der Umgang mit Arrays ist für Anfänger sehr gewöhnungsbedürftig. Trotzdem lohnt es sich, sich mit den verschiedenen Arten und Programmiermöglichkeiten von Arrays vertraut zu machen, denn sie machen die Programme übersichtlicher und kürzer. Insbesondere bei der Programmierung von Datenbanken oder Tabellen sind Arrays unverzichtbar.

Neben dem hier gezeigten Stringarray kann man auch für jeden anderen Dateityp ein Array definieren. Es gibt also Binärarrays, Textarrays, Ganzzahlenarrays, Arrays für Fließkommazahlen etc etc. Sogar für zusammengesetzte Dateitypen wie zb Objekte kann man Arrays verwenden.

Die Elemente eines Arrays haben dabei alle denselben Dateityp. Will man auch noch den Dateityp in der Liste variieren, dann muß man statt eines Arrays eine Collection heranziehen.

## 5.27 Schneller füllen mit dem Arraybefehl

Der Arraybefehl verkürzt das obige Programm enorm. Mit ihm kann man ein Array viel schneller füllen.

Wie das geht zeigt folgendes Beispiel. Sie brauchen nur eine leere Form um es in Gang zu bringen:

```
PUBLIC SUB Form_Open()  
x AS Integer  
a AS String[]  
a = Array("Anna", "Anton", "Toni", "Max", "Klaus")  
FOR x = 0 TO 4  
    print a[x]  
    'Ausgabe des Arrays im Direktfenster  
NEXT  
END
```

Wollen Sie die Ausgabe nicht in das Direktfenster umleiten, dann fügen Sie Ihrer Form eine Textarea hinzu und ändern den Code folgendermassen:

```
PUBLIC SUB Form_Open()  
x AS Integer  
a AS String[]  
a = Array("Anna", "Anton", "Toni", "Max", "Klaus")  
Textareal.Text = ""  
FOR x = 0 TO 4  
    Textareal.Text = Textareal.Text & a[x] & Chr(10)  
NEXT  
END
```

Das Zeichen `chr(10)` fügt einen Zeilenwechsel zwischen die Namen. Mit `Textareal.Text = ""` wird die Textarea am Start gelöscht.

## 5.28 Einfache Arrayroutinen

### 5.28.1 Erzeugung

Es soll ein Integer Array mit 20 Elementen erzeugt werden. Dazu nutzen sie folgende Zeile:

```
feld [20] AS Integer
```

Etwas umständlicher geht das auch so:

```
feld AS NEW Integer[]  
i AS Integer  
FOR i = 0 TO 19  
    feld.Add(i)  
NEXT
```

Zu beachten ist jedoch, dass jeder Array mit [0] beginnt. So ist das letzte Element `feld[19]` (0 bis 19 = 20 Elemente). Laut der Deklaration `feld [20] AS Integer` wäre zu vermuten, dass auch das Element `feld[20]` einen Wert hat. Das stimmt aber nicht, es liegt bereits außerhalb der Deklaration (out of Bound).

## 5.28.2 Operationen mit Elementen

Mit Arrayelementen können alle Operationen wie gewohnt ausgeführt werden.  
Beispiel:

```
feld[2] = 77
feld[3] = feld[2] - 7
```

Die erste Zeile weist dem 3. (dritten!) Element des Arrays die Zahl 77 zu. Die zweite Zeile weist dem 4. (vierten!) Element des Arrays die Zahl  $77 - 7 = 70$  zu.

## 5.28.3 Index

Als Index kann eine Variable oder Konstante angegeben werden:

```
n as integer
n = 1
feld[n] = 25
```

Dieser Code weist dem n-ten Element (dem 2.) den Wert 25 zu.

Als Programm:

```
PUBLIC SUB Form_Open()
feld[4] AS Integer
n AS Integer
feld[2] = 77
feld[3] = feld[2] - 7
n = 1
feld[n] = 25
FOR n = 0 TO 3
    PRINT feld[n]
NEXT
END
```

Ausgabe im Direktfenster:

```
0
25
77
70
```

Wenn Sie `print feld[4]` eingeben, erscheint die Fehlermeldung: Out of bounds

Probieren sie noch einmal folgende Variante, dann wird es vielleicht klarer:

```
PUBLIC SUB Form_Open()
feld[4] AS Integer
'Deklariert das Zahlenarray feld[0 bis 3] !!!
n AS Integer
feld[2] = 77
feld[3] = feld[2] - 7
n = 1
feld[n] = 25
FOR n = 0 TO 3
    PRINT "feld[" & n & "] = " & feld[n]
NEXT
```

```
'PRINT feld[4] ergibt einen Fehler !!!  
END
```

## 5.28.4 Mehrere Dimensionen

Weiterhin können Arrays mit mehr als einer Dimension erzeugt werden.

```
schachbrett[10, 10] AS Integer
```

Erzeugt ein 2-Dimensionales Integer-Feld der Größe 10 x 10.

Als Programmbeispiel:

```
PUBLIC SUB Form_Open()  
  i AS Integer  
  n AS Integer  
  schachbrett[10, 10] AS Integer  
  FOR i = 0 TO 9  
    FOR n = 0 TO 9  
      PRINT i, n  
      schachbrett[i, n] = i*10 + n  
      PRINT schachbrett[i, n]  
    NEXT  
  NEXT  
END
```

## 5.29 Fehler mit Arrays

Beispiel 2:

In diesem Beispiel wird ein Stringarray definiert und am Anfang mit fortlaufenden Zahlen gefüllt. Dabei entspricht die jeweilige Zahl der Position im Array. Bei der zweiten Füllung des Arrays ist das nicht mehr der Fall. Jetzt ist das Array mit Quadratzahlen gefüllt worden.

Um das Programm in Gang zu bringen, brauchen Sie 2 Befehlsbuttons und ein Textlabelfeld.

```
ar AS String[]  
'Der array wird als String definiert  
PUBLIC SUB Button1_Click()  
' Der Array wird ausgegeben  
  i AS Integer  
  txt AS String  
  ar = NEW String[]  
  FOR i = 0 TO 3  
    ar.Add(Str(i))  
  NEXT  
  FOR i = 0 TO 3  
    txt = txt & ar[i]  
  NEXT  
  TextLabel1.Text = txt
```



```

END
PUBLIC SUB Button2_Click()
'Der array wird neu gefüllt
a AS Integer
i AS Integer
txt AS String
ar = NEW String[]
FOR a = 0 TO 3
    ar.Add(Str(a*a))
NEXT
FOR i = 0 TO 3
    txt = txt & ar[i]
NEXT
TextLabel1.Text = txt
END

```

Auch diese Programm können Sie mit dem Arraybefehl verkürzen:

```

ar AS Integer[]
'Der array wir als Zahlenliste definiert
PUBLIC SUB Button1_Click()
'Der Array wird gefüllt und ausgegeben
i AS Integer
txt AS String
ar = Array(0,1,2,3)
FOR i = 0 TO 3
    txt = txt & ar[i]
NEXT
TextLabel1.Text = txt
END
PUBLIC SUB Button2_Click()
'Der array wird mit den Quadratzahlen der ersten Füllung erneut gefüllt
a AS Integer
i AS Integer
txt AS String
ar = Array(ar[0]^2,ar[1]^2,ar[2]^2,ar[3]^2)
FOR i = 0 TO 3
    txt = txt & ar[i]
NEXT
TextLabel1.Text = txt
END

```

**Vorsicht!** Diese Programm funktioniert nur einmal richtig. Beim zweiten Programmablauf erfolgt eine Fehlermeldung.

Type mismatch: Wanted Integer[], got float[] instead.

Die Schachtelung des Arrays ist also fehlerträchtig:

```
ar = Array(ar[0]^2,ar[1]^2,ar[2]^2,ar[3]^2)
```

Als Übung können Sie ein Array aller kleinen Buchstaben des Alphabets aufstellen:

**Beispiel 3:** Ein 3-dimensionales Array

Um das Programm in Gang zu bringen brauchen Sie eine Form und einen Befehlsbutton.

```
PUBLIC SUB Button1_Click()
```

```
DIM i AS Integer
DIM ii AS Integer
DIM iii AS Integer
DIM Matrix[3, 3, 3] AS Integer
FOR i = 0 TO 2
  FOR ii = 0 TO 2
    FOR iii = 0 TO 2
      PRINT i, ii, iii
      Matrix[i, ii, iii] = i*9 + ii*3 + iii
      PRINT Matrix[i, ii, iii]
    NEXT
  NEXT
NEXT
END
```

### 5.30 Split » Zerlegt einen Text in ein Stringarray mit lauter einzelnen Elementen

Der Befehl *Split* zerlegt einen Textstring in kleine Textteile, die durch ein bestimmtes Trennzeichen abgegrenzt sind.

Die Syntax des Befehls lautet:

```
Array = Split ( Text [ , Trennzeichen , Nichttrennen ] )
```

Benutzt man den Befehl ganz einfach nur mit der Angabe für Text, dann wird als Trennzeichen das Komma verwendet.

```
Array = Split(Text)
```

Will man einen Satz beispielsweise in seine Wörter trennen, dann benutzt man folgende Variante:

```
Array = Split(Satz, " ")
```

Wenn man mehrere verschiedene Trennzeichen verwenden will ( zb Komma und Punkt) dann muss man diese als zweiten Parameter übergeben zusammengefügt zu einem einzigen String.

Will man bestimmte Bereiche nicht trennen , dann kann man ein Nichttrennzeichen angeben. Alles was zwischen zwei Nichttrennzeichen steht ( zb in Anführungsstrichen ) wird nicht zerlegt.

Für das Nichttrennzeichen gibt es keine Voreinstellung. Man muß es immer explizit eingeben.

Die Splitfunktion liefert einen Stringarray (= eine Wörterliste) zurück, der jeweils mit den einzelnen zerlegten Elementen gefüllt wurde.

Beispiel: Sie brauchen einen Befehlsknopf, um das Beispiel in Gang zu bringen.

```

PUBLIC SUB Button1_Click()
e AS String[]
s AS String
e = Split("Gambas Almost Means BASIC !", " ")
FOR EACH s IN e
    PRINT s
NEXT
End

```

### Ausgabe:

```

Gambas
Almost
Means
BASIC
!

```

Versuchen Sie einmal dasselbe Programm mit dem Trennzeichen a. Als Ausgabe erhalten Sie dann:

```
G mb s Almost Me ns BASIC !
```

Das Trennzeichen wird also nicht in die Elemente des Arrays mit aufgenommen.

Der Splitbefehl ist sehr praktisch und kann in einer Vielzahl von Programmen gut verwendet werden. Beispiel finden sich in [Gambas: Statistik](#)

## 5.31 Join Der Befehl zum zusammenfügen

Der Befehl klebt eine Liste wieder zusammen. Er ist das Gegenteil des Splitbefehls.

## 5.32 For each

Mit For each werden alle Einträge eines Arrays einer anderen Variable zugewiesen, die man dann ausgeben kann.

Beispiel: Sehr beliebt ist Addy der Summierer. Auch er nutzt einen Stringarray.

Sie brauchen eine Textarea, eine Textbox und einen Commandbutton um das Programm zu starten.

```

PUBLIC SUB Button1_Click()
DIM text AS String
DIM summe AS Float
DIM liste AS String[]
DIM posten AS String
text = textareal.Text

```

```
liste = Split(text,Chr(10))
FOR EACH posten IN liste
  summe = summe + Val(posten)
NEXT
textbox1.Text = summe
END
```

Geben Sie in die Textarea ein paar Zahlen und dazwischen Return ein. Dann drücken Sie auf die Befehlstaste. Die Zahlen werden zusammengezählt. Die Liste in der Textarea kann beliebig korrigiert werden. Auch negative Zahlen werden verarbeitet.

Vorsicht: Geben Sie keine Buchstaben oder Sonderzeichen ein. So robust ist Addy noch nicht.

## **5.33 Arrayroutinen Add,Remove,Resize,Revert,Clear, Sort**

Gambas bringt für alle seine Arrayarten eine Reihe von sehr nützlichen Routinen mit.

### **5.33.1 Arrayroutinen**

- Add
  - ein Element am Ende hinzufügen oder an einer Position einfügen
- Remove
  - ein Element am Ende wegnehmen oder an einer definierten Position wegnehmen
- Revert
  - Reihenfolge umdrehen
- Resize
  - Dem Array eine neue Größe geben
- Clear
  - Alle Elemente weglöschen
- Sort

- In aufsteigender oder absteigender Reihenfolge sortieren

Wie die Arrayroutinen funktionieren, zeigt das folgende Programm.

### **5.33.2 Sie brauchen**

- 11 Buttons
- 4 Textboxen
- 1 Textarea

um das Programm in Gang zu bringen.

### 5.33.3 Layout

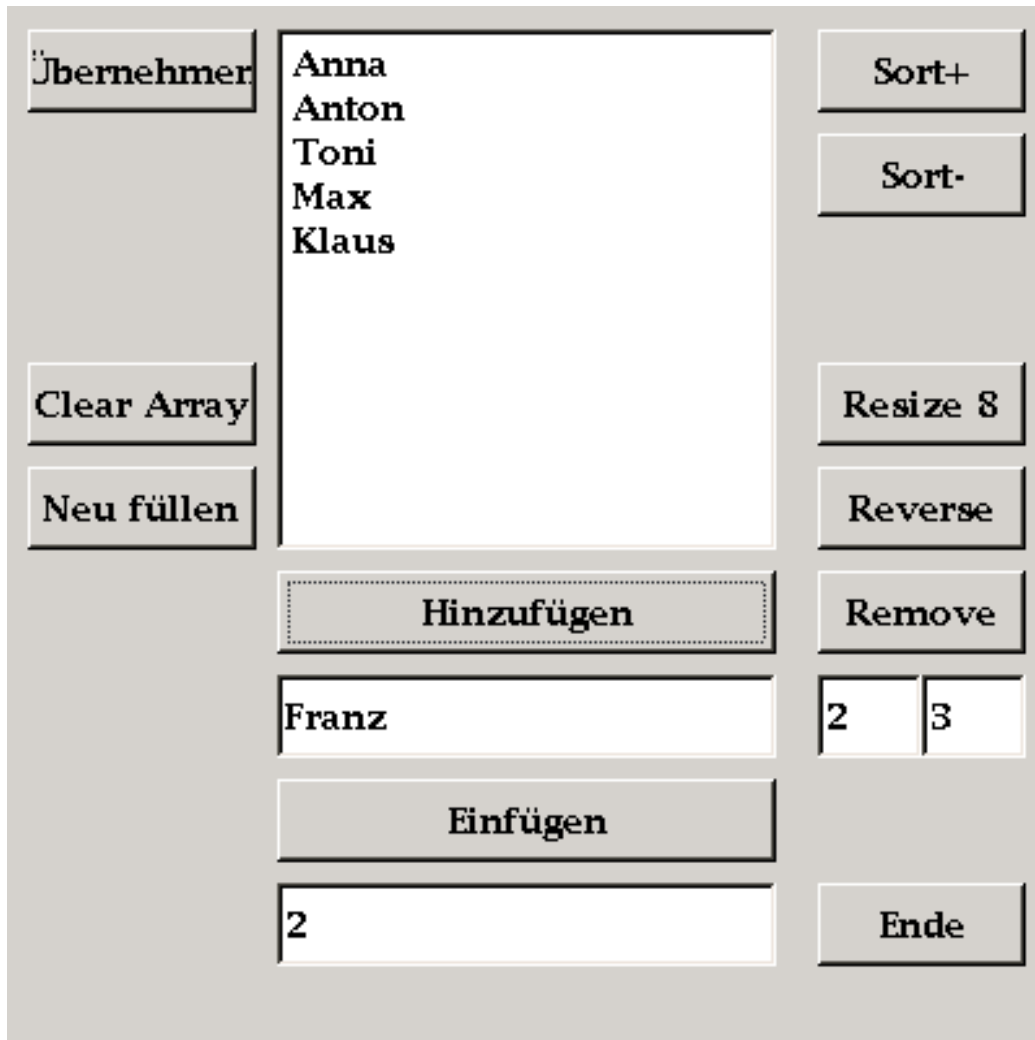


Abbildung 91: Bild:Gambasarrayroutinen.png

### 5.33.4 Code

```

PUBLIC a AS String[]
PUBLIC SUB Form_Open()
x AS Integer
a = Array("Anna", "Anton", "Toni", "Max", "Klaus")
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
    Textareal.Text = Textareal.Text & a[x] & Chr(10)

```

```
NEXT
Textbox1.Text = "Franz"
Button1.Text = "Hinzufügen"
Button2.Text = "Einfügen"
Textbox2.Text = "2"
Textbox3.Text = "2"
Textbox4.Text = "3"
ME.Text = "Arraybefehle"
END

PUBLIC SUB Button1_Click()
x AS Integer
a.Add(Textbox1.Text)
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
'Die Länge des Array wird von 1 an gezählt, die Position von 0 an
Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
END

PUBLIC SUB Button2_Click()
x AS Integer
a.Add(Textbox1.Text,Val(Textbox2.Text))
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
END

PUBLIC SUB Button3_Click()
x AS Integer
a.Sort(0)
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
END

PUBLIC SUB Button4_Click()
x AS Integer
a.Sort(16)
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
END

PUBLIC SUB Button5_Click()
x AS Integer
a.clear
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
END

PUBLIC SUB Button6_Click()
x AS Integer
a.resize(8)
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
```

```
END

PUBLIC SUB Button7_Click()
x AS Integer
a.reverse
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
    Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
END

PUBLIC SUB Button8_Click()
x AS Integer
a.remove(Val(textbox3.text),Val(textbox4.text))
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
    Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
END

PUBLIC SUB Button9_Click()
    ME.Close
END

PUBLIC SUB Button10_Click()
x AS Integer
a = Array("Anna","Anton","Toni","Max","Klaus")
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
    Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
END

PUBLIC SUB Button11_Click()
text AS String
posten AS String
text = textareal.Text
a = Split(text,Chr(10))
FOR EACH posten IN a
NEXT
END
```

Man kann dieses Programm vereinfachen, wenn man die Arrayausgabe in der Textarea in eine Subroutine packt, denn sie kommt immer wieder vor.

```
x as integer
Textareal.Text = ""
FOR x = 0 TO (a.Length - 1)
    Textareal.Text = Textareal.Text & a[x] & Chr(10)
NEXT
```

Übrigens: Für das Array Object[] funktioniert das Sortieren noch nicht.

## 5.34 Array mit Zufallszahlen füllen

Man kann ein Array auch mit Zufallszahlen füllen , um beispielsweise Algorithmen zum Sortieren zu testen.



Wie das geht zeigt folgendes Miniprogramm. Sie brauchen dazu eine Form und einen Befehlsbutton. Das Array wird bereits beim Laden der Form erzeugt und gefüllt. Drückt man auf den Befehlsknopf, wird das Array im Direktfenster ausgegeben.

Es werden 6 Zufallszahlen zwischen 1 und 10 erzeugt und in den Array eingelesen.

Einige Ungereimtheiten ergeben sich hier noch bei der Benennung der Arrays (siehe Programmkommentare) Außerdem erkennt man jetzt auch den Vorteil, daß es 2 Methoden der Array Füllung gibt.

```
' Gambas class file
ar AS Float[]
'warum gibt es eine Fehlermeldung ,
'wenn man das Array mit Integer[] festlegt.
PUBLIC SUB Form_Open()
  Randomize()
  ar = Array(Int(Rnd(1,10)) , Int(Rnd(1,10)) , Int(Rnd(1,10)) , Int(Rnd(1,10)) , Int(Rnd(1,10)) ,
Int(Rnd(1,10)))
END
PUBLIC SUB Button1_Click()
element AS Integer
  FOR EACH element IN ar
    PRINT element
  NEXT
END
```

### Alternative:

```
ar AS Integer[]
'Obwohl der Array als Integerarray definiert ist,
'funktioniert er weiter unten als Stringarray Wieso ?
PUBLIC SUB Form_Open()
x AS Integer
ar = NEW Integer[]
FOR x = 0 TO 10
  ar.Add(Str(x))
NEXT
END
PUBLIC SUB Button1_Click()
x AS Integer
Randomize()
FOR x = 0 TO 10
  ar[x] = Int(Rnd(1,10))
NEXT
END
PUBLIC SUB Button2_Click()
element AS Integer
  FOR EACH element IN ar
    PRINT element;
  NEXT
  PRINT
END
```

Wieviele Elemente hat der Array ? Warum so viele ? Probieren Sie das Programm einmal ohne den Randomize Befehl, dann erkennen Sie seine Notwendigkeit.

## 5.35 Lottozahlen ( 7 aus 49 )

Das folgende Programm erzeugt sieben Lottozahlen , wirft Doppelgänger heraus und gibt die Zahlen sortiert aus: Genutzt wird dabei ein Array mit 49 Elementen.

Zunächst wird das Programm in seine Einzelteile zerlegt und auf 4 Comanddbut-  
tons gelegt.

- Commandbutton1 » Array mit 7 Zufallszahlen
- Commandbutton2 » Kompletten Array über Print ausgeben
- Commandbutton3 » Nur die Elemente ausgeben, die ungleich Null sind
- Commandbutton4 » Kompletten Array wieder mit Nullen füllen.

```

ar AS Integer[]
PUBLIC SUB Form_Open()
'schon beim Öffnen der Form wird der Array mit lauter Nullen gefüllt.
x AS Integer
ar = NEW Integer[]
FOR x = 0 TO 48
  ar.Add(Str(0))
NEXT
END
PUBLIC SUB Button1_Click()
'dann wird der Array mit 7 Zufallszahlen an den Stellen gefüllt, die mit der Zufallszahl
übereinstimmen
x AS Integer
r AS Integer
Randomize()
FOR x = 0 TO 6
  r = Int(Rnd(1,49))
  IF r = ar[r] THEN x = x - 1
  'Falls ein Doppelgänger auftritt gehe eins zurück
  ar[r] = r
NEXT
END
PUBLIC SUB Button2_Click()
'zu Testzwecken wird das ganze Array ausgegeben
element AS Integer
FOR EACH element IN ar
  PRINT element
NEXT
END
PUBLIC SUB Button3_Click()
'jetzt werden nur noch die Elemente ausgegeben , die ungleich 0 sind
element AS Integer
FOR EACH element IN ar
  IF element <> 0 THEN PRINT element
NEXT
END
PUBLIC SUB Button4_Click()
'der Array wird wieder komplett auf Null gestellt
x AS Integer
FOR x = 0 TO 48
  ar[x] = 0
NEXT

```

END

Nachdem man das Programm in einzelne Schritte zerlegt hat, kann man es jetzt auf einen einzigen Button zusammenfassen. Und die Zusatzzahl fehlt natürlich noch.

```
' Gambas class file Lottozahlen
ar AS Integer[]
PUBLIC SUB Form_Open()
'schon beim Öffnen der Form wird der Array mit lauter Nullen gefüllt.
x AS Integer
ar = NEW Integer[]
FOR x = 0 TO 48
  ar.Add(Str(0))
NEXT
END
PUBLIC SUB Button1_Click()
x AS Integer
r AS Integer
element AS Integer
'der Array wird komplett auf Null gestellt
FOR x = 0 TO 48
  ar[x] = 0
NEXT
'dann wird der Array mit 7 Zufallszahlen an den Stellen gefüllt, die mit der Zufallszahl
übereinstimmen
Randomize()
FOR x = 0 TO 6
  r = Int(Rnd(1,49))
  IF r = ar[r] THEN x = x - 1
  'Falls ein Doppelgänger auftritt gehe eins zurück
  ar[r] = r
NEXT
'nur die Zahlen <> 0 werden ausgegeben
FOR EACH element IN ar
  IF element <> 0 THEN PRINT element
NEXT
END
```

Überlegen Sie einmal warum die Elemente des Arrays zweimal auf Null gestellt wurden. Wenn Sie es nicht wissen, dann werfen Sie doch die eine Schleife heraus und probieren Sie das Programm ein paar Mal aus. Dann erkennen Sie die Notwendigkeit, wenn man mehrfach den Befehlsbutton drückt.

## 5.36 Doppelgänger aus dem Array herauswerfen

Idee dazu: Erst sortieren, dann vergleichen, dann gleiche Elemente löschen.

## 5.37 Theorie

.Array Diese virtuelle Klasse ist die Eltern Klasse jedes anderen Gambas arrays.

Diese Klasse ist virtuell. Man kann sie nicht als Datentyp benutzen. Sie ist nicht erzeugbar (creatable. ).

Eigenschaften:

- Count (identisch mit Length)
- Length

Methoden

- Add
- Clear
- Remove
- Resize
- Reverse
- Sort

## 5.38 Arten von Arrays

Arrays sind in Gambas meist an den eckigen Klammern zu erkennen: []

Mit der Schnellmethode über den Arraybefehl können Sie aber auch mit runden Klammern () definiert werden.

Wenn Sie nicht wissen, was für eine Art von Array sie definiert haben, dann lassen Sie sich die Art des Arrays im Direktfenster ausgeben. Wenn man sein Array mit dem Arraybefehl gefüllt hat, dann wird der Typ des Arrays durch das erste Element definiert. Die anderen Elemente werden automatisch zum selben Typ umgewandelt.

Mit Join kann man die Elemente des Arrays bei der Ausgabe trennen

Beispiel:

```
PUBLIC SUB Form_Open()  
  ar AS Integer[]  
  ar = Array(3,3,3)  
  PRINT Object.Type(ar)  
END
```

Ausgabe: Integer[]

```
PRINT Object.Type(Array("2.4", 3, 3.2))
```

Ausgabe: String[]

```
PRINT [ "A", "B", "C" ].Join("/")
```

Ausgabe A/B/C

## 5.39 Zweidimensionales Array anlegen

Siehe [Gambas: Tabelle](#)

## 5.40 Array Routinen

- Zweidimensionale Felder anlegen
- Ein Boolean-Array
- Anzahl der Dimensionen eines Arrays bestimmen
- Arrays dynamisch vergrößern
- FAQ - Oft gestellte Fragen über Arrays
- Element in Array suchen
- Schnelles Einfügen und Löschen
- Arrayinhalte schnell mischen
- Schnelle Sortierung mit QuickSort und MinSort

## 5.41 Stoffsammlung und Fragen zu Arrays

Will man die einzelnen Einträge durch ein Zeichen trennen, geht das normalerweise mit der Join Methode.

Hinzufügen von Einträgen

Wie kann man Einträge in ein Array hinzufügen ?

Zählen der Einträge

Wie erfolgt das Zählen der Einträge ? ( Length Eigenschaft )

```
print a.Lenght funktioniert aber leider nicht
```

### Einträge durchsuchen

Wenn man ein Array nach einem Eintrag durchsuchen will, geht das so:

### Sortieren der Einträge

Wie kann man Arrays sortieren ? Wie kann man nach den Namen der Werte sortieren lassen ? Wie kann man die Elemente in Ihrer Reihenfolge umdrehen ?

Lassen Sie sich am Anfang nicht von den vielen verschiedenen Befehlen verwirren. Suchen Sie sich am Anfang nur die wichtigsten heraus und arbeiten Sie mit Ihnen.

AS, Print, =, &, FOR TO, NEXT, ME.Close, IF THEN, TRUE, FALSE

Der Rahmen für ein Unterprogramm wird mit PUBLIC SUB und END am Ende von Gambas automatisch beim Klick z.B. auf einen Befehlsbutton vorgegeben.

Wenn Ihnen ein neuer Befehl über den Weg läuft, dann schauen Sie einfach mit F1 in der Hilfe nach, was er bedeutet.

## 5.42 Alphabetischer Befehlsindex

Auf [gambasdoc.org](http://gambasdoc.org) finden Sie eine stets aktuelle Übersicht zu allen Gambas-Befehlen.

### Mathematische Zeichen in Gambas

Zahl + Zahl	Addiert zwei Zahlen, zählt zusammen
- Zahl	Rechnet die negative Zahl einer Zahl aus. Die negative Zahl von Null ist Null.N = 5R = -8PRINT N ; " " ; R ; " , " ; -N ; " " ; -R==> 5 -8 , -5 8
Zahl - Zahl	Abziehen, zieht eine Zahl von einer anderen ab
Zahl * Zahl	Multipliziert zwei Zahlen, malnehmen

Zahl / Zahl	Bildet einen Bruch. Eine Zahl wird durch eine andere geteilt. Ein <i>division by zero</i> (= Division durch Null) Fehler erscheint, wenn der Teiler gleich Null ist. Das Ergebnis ist nicht immer eine ganze Zahl. Deswegen sollte das Ergebnis als eine float Variable deklariert werden.
Zahl ^ Potenz	Eine Zahl wird potenziert. PRINT 4^3==> 64
Zahl \ Zahl	Berechnet den Quotienten von 2 Zahlen. Ein <i>division by zero</i> (= Division durch Null) Fehler tritt auf, wenn die Zahl rechts vom Backslash (Schrägstrich) gleich Null ist. A \ B ist dasselbe wie INT(A/B). PRINT 9\4 ; " , " ; 9 MOD 4==> 2 , 1
Zahl MOD Zahl	Berechnet den Rest eines Bruches zweier Zahlen. Ein <i>division by zero</i> (= Division durch Null) Fehler tritt auf, wenn die Zahl rechts vom MOD Befehl gleich Null ist. PRINT 9\4 ; " , " ; 9 MOD 4==> 2 , 1
Zahl AND Zahl	Berechnet das mathematische AND der Binärwerte zweier Zahlen. PRINT 5 AND 3==> 1
Zahl OR Zahl	Berechnet das mathematische OR (oder) der Binärwerte zweier Zahlen. PRINT 5 OR 3==> 7
Zahl XOR Zahl	Berechnet das mathematische ExOR (entweder oder) der Binärwerte zweier Zahlen. PRINT 5 XOR 3==> 6

### 5.43 AND (Und)

Normalerweise wird der AND Befehl nicht sehr oft benutzt.

Er hat 2 Funktionen:

1. Logische Verknüpfung von zwei Aussagen und Überprüfung Ihres Wahrheitswertes
2. Vergleich von 2 Zahlen im binären Zahlensystem und Ausgabe von Übereinstimmungen

Beispiel für die Erste Funktion:

```
PRINT TRUE AND FALSE
```

Ausgabe: False

```
PRINT TRUE AND TRUE
```

Ausgabe: True

Eine Zusammenfassung steht in folgendem Programm:

```
PUBLIC SUB Form_Open()
  DIM a AS Boolean
  DIM b AS Boolean
  FOR a = 0 TO 1
    FOR b = 0 TO 1
      PRINT Str$(a) & " AND " & Str$(b) & " ==> " & Str$(a AND b)
    NEXT
  NEXT
END
```

Wahrheitstabelle:

A	B	A AND B
False	False	False
False	True	False
True	False	False
True	True	True

Elektrische Schaltung: Die AND Verknüpfung entspricht einer elektrischen Serien- bzw. Reihenschaltung.

## 5.44 OR (Oder)

Der Logikbefehl OR verknüpft zwei Aussagen und gibt für die Verknüpfung einen neuen Wahrheitswert aus.

OR = Oder = Das Eine oder das Andere oder Beide.

Es reicht aus, wenn eine der beiden Aussagen wahr ist. Auch dann ist die Oder Verknüpfung bereits wahr.

Eine Zusammenfassung steht in folgendem Programm:



```

PUBLIC SUB Form_Open()
  DIM a AS Boolean
  DIM b AS Boolean
  FOR a = 0 TO 1
    FOR b = 0 TO 1
      PRINT Str$(a) & " OR " & Str$(b) & " ==> " & Str$(a OR b)
    NEXT
  NEXT
END

```

Wahrheitstabelle:

A	B	A OR B
False	False	False
False	True	True
True	False	True
True	True	True

Elektrische Schaltung: Die OR Verknüpfung entspricht einer elektrischen Parallelschaltung.

## 5.45 XOR

XOR = Exclusive OR = Entweder das Eine oder das Andere

Die Funktion zeigt folgendes Programm

```

STATIC PUBLIC SUB Main()
  DIM a AS Boolean
  DIM b AS Boolean
  FOR a = 0 TO 1
    FOR b = 0 TO 1
      PRINT Str$(a) & " XOR " & Str$(b) & " ==> " & Str$(a XOR b)
    NEXT
  NEXT
END

```

Wahrheitstabelle:

A	B	A XOR B
False	False	False
False	True	True
True	False	True
True	True	False

## 5.46 NOT

NOT kehrt den Wahrheitswert um.

Die Funktion zeigt folgendes Programm

```

STATIC PUBLIC SUB Main()
  DIM a AS Boolean
  FOR a = 0 TO 1
    PRINT Str$(a) & " ==> " & Str$(NOT a)
  NEXT
END

```

Wahrheitstabelle:

A	NOT A
True	False
False	True

## 5.47 Bitweise Operatoren

### 5.47.1 Bitweises AND

Betrachten Sie dazu folgendes Programm:

```

PUBLIC SUB Form_Open()
  DIM x AS Integer
  DIM y AS Integer
  FOR x = 1 TO 15
    FOR y = 1 TO 15
      PRINT Bin$( x, 4 ), Bin$( y, 4 ), Bin$( x AND y, 4 )
    NEXT
  NEXT
END

```

### 5.47.2 Bitweises OR

Betrachten Sie dazu folgendes Programm:

```

PUBLIC SUB Form_Open()
  DIM x AS Integer
  DIM y AS Integer
  FOR x = 1 TO 15
    FOR y = 1 TO 15
      PRINT Bin$( x, 4 ), Bin$( y, 4 ), Bin$( x OR y, 4 )
    NEXT
  NEXT
END

```

### 5.47.3 Bitweises XOR

Betrachten Sie dazu folgendes Programm:

```
PUBLIC SUB Form_Open()  
DIM x AS Integer  
DIM y AS Integer  
FOR x = 1 TO 15  
  FOR y = 1 TO 15  
    PRINT Bin$( x, 4 ), Bin$( y, 4 ), Bin$( x XOR y, 4 )  
  NEXT  
NEXT  
END
```

## 5.48 Übersicht der Strukturen von Gambas

Wie die meisten Programmiersprachen gibt es in Gambas verschiedene Möglichkeiten zur Ablaufsteuerung:

- Reihenfolge
- Auswahl
- Wiederholung
- Unterprogramme

### 5.48.1 Reihenfolge

Normalerweise wird in einem Programm eine Befehlszeile nach der anderen abgearbeitet. Ausnahmen zu diesem Prinzip gibt es allerdings auch: Die Definition von Variablen mit dem DIM-Befehl erfolgt immer am Programmanfang oder am Anfang eines Unterprogrammes. Außerdem bietet der Befehl GOTO Möglichkeiten, die Reihenfolge des Programmes zu überspringen. Bei Unterprogrammen spielt die Reihenfolge im Codefenster keine Rolle, da sie über ihren Namen angesprungen werden. Trotzdem sollte man sich eine gewisse Reihenfolge angewöhnen. Beispielsweise gehört das Unterprogramm *PUBLIC SUB Form\_Open()* an den Anfang des eigenen Codes.

```
<Anweisung>  
<Anweisung>  
<Anweisung>  
<...>
```

## 5.48.2 Auswahl

Programmverzweigungen erfordern Bedingungen und Entscheidungen. Um dies zu ermöglichen gibt es verschiedene Befehle. Falls eine Bedingung nicht zutreffen sollte, wird die Zeile einfach übersprungen und das Programm macht bei der Zeile nach der Bedingungsabfrage weiter.

### Einseitige Auswahl

```
IF <Bedingung> THEN <Anweisung>
```

### Zweiseitige Auswahl

```
IF <Bedingung> THEN  
  <Anweisung>  
ELSE  
  <Anweisung>  
ENDIF
```

### Fallunterscheidung

```
SELECT CASE TRUE  
  CASE <Bedingung1>  
    <Anweisung>  
  CASE <Bedingung2>  
    <Anweisung>  
  CASE <Bedingung3>  
    <Anweisung>  
  CASE ELSE <Anweisung>  
END SELECT
```

## 5.48.3 Schleifen

In Schleifen wird ein Befehl oder eine Befehlsfolge mehrfach wiederholt.

### Zählschleife

```
FOR <Laufvariable> = <Startwert> TO <Endwert>  
  <Anweisung>  
NEXT
```

### Abweisende Schleife

```
WHILE <Wiederhol-Bedingung>  
  DO <Anweisung>
```

### Nicht abweisende Schleife

```
REPEAT  
  <Anweisung>  
UNTIL <Abbruch-Bedingung>
```

### Zählschleife für Arrays

```
FOR EACH <Element> IN <Liste>  
  <Anweisung>  
NEXT
```

Die aufgezählten Strukturen kann man ineinander verschachteln. Dies führt aber oft zu unübersichtlichem Code und sollte vielleicht eher durch Funktionen in Unterprogrammen ersetzt werden.

## 5.49 Verzweigungen

### 5.49.1 if - then

Die einfachste Verzweigung eines Programmes erfolgt mit dem if - then Befehl

- If heißt auf deutsch **FALLS**
- then heißt auf deutsch **DANN**

Der If Then Befehl macht es möglich Programmverzweigungen zu programmieren, wenn eine bestimmte Bedingung erfüllt ist. Auf deutsch heißt der Befehl Falls die Bedingung wahr ist, dann mache folgendes.

Am besten versteht man dies an Hand eines Beispiels: Es gibt den Wochentag des heutigen Tages aus. Das Programm funktioniert im Terminalmodus:

```

STATIC PUBLIC SUB Main()
PRINT WeekDay(Now)
PRINT "Heute ist ";
IF WeekDay(Now) = 1 THEN PRINT "Montag"
IF WeekDay(Now) = 2 THEN PRINT "Dienstag"
IF WeekDay(Now) = 3 THEN PRINT "Mittwoch"
IF WeekDay(Now) = 4 THEN PRINT "Donnerstag"
IF WeekDay(Now) = 5 THEN PRINT "Freitag"
IF WeekDay(Now) = 6 THEN PRINT "Samstag"
IF WeekDay(Now) = 7 THEN PRINT "Sonntag"
END

```

Das Programm nutzt den Befehl Weekday und den Befehl Now von Gambas

Zwischen If und Then kann man auch mehrere Bedingungen auflisten und auch bei den Anweisungen nach dem *Then* Befehl kann mehrere Anweisungen aufreihen, die man dann mit Endif abschließt.

```

If
  Bedingung1
  Bedingung2
Then
  Anweisung1
  Anweisung2
Endif

```

## 5.49.2 if then else

Der *If then Else* erlaubt die Programmierung einer Alternative zur If Bedingung. Das wird am besten an einem Beispiel klar

```
dim name as string
if name = männlich then
  Print "An Herrn " & name
else
  Print "An Frau " & name
endif
```

Hier muß allerdings noch die Funktion *männlich* definiert werden.

Ein weiteres Beispiel: Sie brauchen eine leere Form , um das Programm in Gang zu bringen. Es erzeugt 100 Zufallszahlen zwischen 1 und 99 und überprüft ihre Teilbarkeit durch 2.

```
PUBLIC SUB Form_Open()
a AS Integer
x AS Integer
Randomize()
FOR x = 1 TO 100
  a = Int(Rnd(1,100))
  'Zufallszahl zwischen 1 und 99 erzeugen
  IF a MOD 2 = 0 THEN
    'teilbar durch 2 ??
    PRINT a & " ist teilbar durch 2"
  ELSE
    PRINT a & " ist nicht teilbar durch 2"
  ENDIF
NEXT
END
```

## 5.49.3 GOTO Sprungadresse

Auch das alte GOTO gibt es noch in Gambas. Man kann es allerdings nur in einem Unterprogramm verwenden. Wie das geht zeigt folgendes Beispiel:

Man braucht eine Textarea und einen Befehlsbutton um das Programm in Gang zu bringen.

```
PUBLIC SUB Form_Open()
ME.Width = 400
ME.Height = 300
ME.Text = "GOTO Befehl mit Labels als Sprungadresse"
Button1.Text = "Zufallszahl einordnen"
Button1.Width = 200
Textareal.Text = ""
Textareal.Width = 300
Textareal.Height = 160
END
PUBLIC SUB Button1_Click()
z AS Integer
Randomize()
z = Int(Rnd(1,100))
```

```

IF z < 10 THEN GOTO zehn
IF z < 20 THEN GOTO z2
IF z < 50 THEN GOTO z5
'PRINT " z > 49 , da z = ";z
textareal.Text = textareal.Text & " z > 49 , da z = " & z & Chr(10)
RETURN

zehn:
textareal.Text = textareal.Text & " z < 10 , da z = " & z & Chr(10)
RETURN

z2:
textareal.Text = textareal.Text & " z < 20 , da z = " & z & Chr(10)
RETURN

z5:
textareal.Text = textareal.Text & " z < 50 , da z = " & z & Chr(10)
RETURN
END

```

## 5.49.4 Select Case

Der Select Case Befehl erlaubt die Auswahl aus mehreren Alternativen.

Wie er funktioniert zeigt folgendes Beispiel. Es überprüft die Randomfunktion eines Würfels 1000 mal und gibt die Häufigkeit, wie oft eine Zahl des Würfels erhalten wurde im Direktfenster aus. Sie brauchen nur eine leere Form um das Programm in gang zu bringen.

```

PUBLIC SUB Form_Open()
x AS Integer
w AS Integer
a AS Integer
b AS Integer
c AS Integer
d AS Integer
e AS Integer
f AS Integer
Randomize()
FOR x = 1 TO 1000
w = Int(Rnd(6) + 1)
SELECT CASE TRUE
CASE w = 1
a = a + 1
CASE w = 2
b = b + 1
CASE w = 3
c = c + 1
CASE w = 4
d = d + 1
CASE w = 5
e = e + 1
CASE w = 6
f = f + 1
END SELECT
NEXT
PRINT a,b,c,d,e,f
END

```

Das Programm kann man etwas abkürzen:

```
PUBLIC SUB Form_Open()  
x AS Integer  
w AS Integer  
a AS Integer  
b AS Integer  
c AS Integer  
d AS Integer  
e AS Integer  
f AS Integer  
FOR x = 1 TO 1000  
    w = Int(Rnd(6) + 1)  
    SELECT CASE w  
        CASE 1  
            a = a + 1  
        CASE 2  
            b = b + 1  
        CASE 3  
            c = c + 1  
        CASE 4  
            d = d + 1  
        CASE 5  
            e = e + 1  
        CASE 6  
            f = f + 1  
    END SELECT  
NEXT  
PRINT a, b, c, d, e, f  
END
```

Für eine genaue Syntaxbeschreibung siehe:

- <http://www.gambasdoc.org/help/lang/select?de>

## 5.50 Schleifen

Schleifen erlauben es, einen bestimmten Programmteil mehrfach zu wiederholen. Wenn man nicht weiß, wie oft man die Anweisung in der Schleife wiederholen muß, dann sind Do-Schleifen eine gute Wahl. Wenn man genau weiß, wie oft die Schleife wiederholt werden soll, dann sollte man den For...Next-Befehl wählen. Die For-Next Schleife verwendet eine Zähler (counter), dessen Wert während jeder Wiederholung der Schleife erhöht oder verringert wird.

### 5.50.1 For next Schleife

Schleifen erlauben es, einen bestimmten Programmteil mehrfach mit einer sich verändernden Zahl durchlaufen zu lassen. Deswegen wird die FOR...NEXT Schleife auch als Zählerschleife bezeichnet.



Das Prinzip der Schleife schaut so aus:

```
FOR i = 1 to 9 STEP 1
  Print i 'Programmcode
NEXT
```

Die Variable *i* besitzt am Anfang den Wert 1. Jedesmal wenn das Programm das Ende der Schleife bei dem Befehl NEXT erreicht, springt das Programm an den Anfang der Schleife bei FOR zurück. Für jeden neuen Schleifendurchgang wird *i* um 1 (STEP 1) erhöht. Erreicht *i* den Endwert der Schleife, den *to Wert* (im Beispiel 9) so bricht die Schleife ab, der Programmablauf wird nach der Schleife fortgesetzt.

Das folgende Programm zeigt den Gebrauch einer For-Next Schleife. Sie brauchen dazu nur eine leere Form. Klicken Sie auf die Form im Entwurfsmodus und geben Sie den Programmcode weiter unten ein. Dann starten Sie das Programm mit F5. Jedesmal wenn sie mit der Maus auf die Form kommen, wird Hallo Welt 10 mal ausgegeben.

```
PUBLIC SUB Form_Enter()
DIM i AS Integer
FOR i = 1 TO 10
  PRINT "Hallo Welt"
NEXT
END
```

Probieren Sie eine Variante die noch einfacher ist:

```
PUBLIC SUB Form_Enter()
DIM i AS Integer
FOR i = 1 TO 10
  PRINT i
NEXT
END
```

Probieren Sie noch folgende Variante:

```
PUBLIC SUB Form_Enter()
DIM i AS Integer
FOR i = 1 TO 10
  PRINT i,i*i,i*i*i
NEXT
END
```

Sie können die 3 Beispiele auch mit dem Befehl

```
PUBLIC SUB Form_Open()
```

automatisch starten.

Um Schleifen lesbarer zu machen, rückt man den inneren Befehlsteil der Schleife ein. Er wird mehrfach durchlaufen. Im Gegensatz zu VB verzichtet Gambas auf die Nennung des Zählers am Ende der Schleife nach dem Next Befehl.

```
For I = 1 to 10
  print I
' Next I    in VB möglich
' In Gambas als überflüssig herausgeworfen.
Next
```

Wenn Sie auf die Grafik verzichten und den Terminalmodus nutzen, dann lautet das letzte Programm folgendermaßen:

```
STATIC PUBLIC SUB Main()  
DIM i AS Integer  
FOR i = 1 TO 10  
    PRINT i,i*i,i*i*i,i^4,i^5  
NEXT  
END
```

Das Ergebnis schaut so aus:

```
1 1 1 1 1  
2 4 8 16 32  
3 9 27 81 243  
4 16 64 256 1024  
5 25 125 625 3125  
6 36 216 1296 7776  
7 49 343 2401 16807  
8 64 512 4096 32768  
9 81 729 6561 59049  
10 100 1000 10000 100000
```

Eine Schleife mit dem Step Befehl zeigt folgendes Beispiel: Erstaunlicherweise funktioniert hier das Komma in der Printausgabe nicht richtig . Probieren Sie einmal das Semikolon stattdessen aus.

```
PUBLIC SUB Form_Open()  
x AS Integer  
FOR x = 0 TO 100 STEP 5  
    PRINT x,  
NEXT  
END
```

Das Ergebnis sollte lauten:

```
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
```

Das tut es aber nicht ! Noch ein paar komische Beispiele:

```
PUBLIC SUB Form_Open()  
DIM i AS Integer  
FOR i = 1 TO 100 STEP i  
    PRINT i  
NEXT  
END
```

```
PUBLIC SUB Form_Open()  
DIM i AS Integer  
DIM x AS Integer  
FOR x = 1 TO 10  
    FOR i = 1 TO 100 STEP x  
        PRINT i & " "  
    NEXT  
NEXT  
END
```

## 5.50.2 Do Schleifen

Eine DO LOOP Schleife ist im Prinzip eine Endlosschleife und muß durch irgendeine Bedingung in der Schleife unterbrochen werden.

**Beispiel 1: Endlosschleife**

```
PUBLIC SUB Form_Open()
x AS Integer
x = 1
DO
  THEN PRINT "x = " & x
  x = x + 1
LOOP
END
```

**Beispiel 2: Endlosschleife , scheinbar zu Ende.**

```
PUBLIC SUB Form_Open()
a AS Integer
x AS Integer
Randomize()
x = 1
DO
  a = Int(Rnd(1,100))
  IF x < a THEN PRINT "x = " & x & " a = " & a
  x = x + 1
LOOP
END
```

Warum werden nicht alle Zahlen von 1 bis 99 im Direktfenster ausgegeben ? Wann und warum hört die Schleife eigentlich auf ?

## 5.50.3 For Each Schleife

Kommt bei Arrays zur Anwendung Siehe [Gambas: Arrays](#)

## 5.50.4 Schleife und Verzweigung kombiniert

Der Programmcode schaut so aus. Sie brauchen nur einen Befehlsknopf und starten mit F5.

```
PUBLIC SUB Button1_Click()
DIM k AS Integer
FOR k = 1 TO 10000
  IF k = 5000 THEN PRINT "Die Zahl 5000 wurde erreicht !"
NEXT
END
```

## 5.51 Sortierungen

Siehe [Gambas: Sortieren](#)

## 5.52 Allgemeines

Funktionen sind **Unterprogramme, die einen Wert** ( Zahl, Text etc) an das Hauptprogramm **zurückgeben**.

Umgekehrt wird oft ein Wert vom Hauptprogramm an die Funktion übergeben, dort wird der Wert verarbeitet und das Ergebnis wiederzurückgegeben.

Funktionen sind am Anfang gewöhnungsbedürftig. Beim effektiven Gebrauch einer Programmiersprache sind sie aber irgendwann unverzichtbar, da sie die Programmerstellung erleichtern und die Programme verkürzen, wenn ein Programmteil öfter durchlaufen werden muss. Fast alle abgeleiteten Befehle einer Programmiersprache haben sich aus Funktionen entwickelt. Prozeduren sind mit den Funktionen verwandt, nur geben Sie keinen Wert zurück.

## 5.53 Ein ganz einfaches Beispiel

Sie definieren eine Funktion **Add** die 2 Zahlen zusammenzählt.

Die 2 Zahlen , die addiert werden sollen, werden dabei einfach in der Klammer hinter dem Funktionsnamen angegeben.

Sie brauchen für das Beispiel:

- Eine Form
- Einen Commandbutton

Der Code lautet:

```
'Hauptprogramm
PUBLIC SUB Button1_Click()
  Message.Info(Add(6,10))
  'Funktionsaufruf und Rückgabe des Wertes
END
'Funktion als Unterprogramm
PUBLIC FUNCTION Add(a AS Integer, b AS Integer) AS Integer
  RETURN a + b
END
```

Diese Funktion ist von den Werten a und b abhängig. Die Werte a und b können beliebig gewählt werden.

## 5.54 Beispiele von Funktionen:

- Time ( Die Systemzeit wird zurückgegeben)
- Date ( Das Datum wird zurückgegeben)
- Quadratwurzel ( Eine Zahl wird eingegeben , die Quadratwurzel kommt zurück)  $a^*a$  wird zu  $\text{sqr}(a)$
- VGA Farbe in RGB umwandeln ( Eine der 16 VGA Farben wird eingegeben, der RGB-Wert kommt zurück)

### 5.54.1 Beispiel TIME: ( in die Funktion geht kein Parameter ein , es kommt trotzdem ein Wert zurück)

Ein gutes Beispiel für eine einfache Funktion ist der **Befehl Time** (= Zeit). Diese Funktion liefert die aktuelle Systemzeit. Sie können diese Funktion mit folgendem Befehl testen:

```
Print Time
```

Das folgende kleine Programm ohne Grafik gibt die Zeit im Direktfenster aus:

```
STATIC PUBLIC SUB Main()
  PRINT Time
END
```

Wenn Sie die Zeit immer wieder aktualisiert haben wollen, müssen Sie eine Endlosschleife einbauen. Das Programm schaut dann so aus:

```
STATIC PUBLIC SUB Main()
  DIM i AS Integer
  i = 1
  DO WHILE i <= 10
    PRINT Time
  LOOP
END
```

Ein ähnliches Beispiel für die Grafikausgabe aktualisiert die Zeitausgabe immer wieder. Dazu nutzt man den Timer, das **Zeitgeber-Steurelement**. Dieser hat zunächst mit der aktuellen Zeit nichts zu tun, sondern löst ein Ereignis nach einer bestimmten Zeit aus.

Das folgende kleine Beispielprogramm kombiniert den Timer und die Timefunktion zu einer ständig aktualisierten Zeitangabe, wie sie diese auch rechts unten an ihrem PC finden.

Starten Sie ein neues Projekt. Nennen Sie es Zeitausgabe. Platzieren Sie anschließend eine Textbox auf Ihrer Form. Außerdem brauchen Sie das Zeitgebersteuer-element mit dem Uhrensymbol. Holen Sie sich beides mit F6 und einem Doppelclick auf das Symbol aus der Werkzeugkiste. Achten Sie darauf, dass der Zeitgeber mit der **Enabled Eigenschaft** erst angeschaltet werden muß. Dies können Sie bei den Eigenschaften des Timers zur Entwurfszeit einstellen oder Sie können es auch im Programmcode mit der Zeile

```
timer1.Enabled = TRUE
```

erreichen.

```
PUBLIC SUB Timer1_Timer()  
timer1.Enabled = TRUE  
textbox1.Text = Time  
END
```

Das Programm zeigt in der Textbox die aktuelle Zeit an. Diese wird jede Sekunde (= 1000 millisekunden) aktualisiert.

### 5.54.2 Beispiel UCASE : ( in die Funktion geht ein Parameter ein und es kommt ein umgewandelter Wert zurück)

Die Funktion **UCase** (= **Großschreibung**) ist eine einfache Funktion, die nur die Übergabe eines Strings benötigt. Um so etwas ( Variablen, Konstanten oder Strings ) zu übergeben, müssen diese in Klammern eingeschlossen werden und nach dem Funktionsnamen stehen.

Erstellen Sie eine Form mit einer Textbox und einem Commandbutton. Schreiben Sie für den Commandbutton folgenden Code:

```
PUBLIC SUB Button1_Click()  
Textbox1.Text = UCase(Textbox1.text)  
END
```

Starten Sie das Programm, tippen Sie beliebigen Text in das Textfeld und klicken Sie auf den Befehlsbutton. Als Ergebnis steht der Text in Großbuchstaben in der Textbox.

Folgender Code funktioniert erstaunlicherweise nicht:

```
PUBLIC SUB TextBox1_KeyRelease()  
IF Key.return THEN UCase(Textbox1.text)  
END
```

Der Text wird in der Version 0.99 nicht groß geschrieben.

Obwohl folgender Code für die KeyReleasemethode funktioniert. Meines Erachtens liegt hier noch ein Programmfehler vor.

```
PUBLIC SUB TextBox1_KeyRelease()
  IF Key.return THEN textbox1.visible = FALSE
END
```

Die Textbox wird unsichtbar.

Wenn Sie die Funktionen **UCase** und **LCase** (= **Kleinschreibung**) noch einmal zusammen in einem einfachen Programm anschauen wollen, dann probieren Sie folgenden Code ohne Grafik:

```
STATIC PUBLIC SUB Main()
  DIM a AS String
  a = "Test"
  PRINT a
  PRINT LCase(a)
  PRINT UCase(a)
END
```

( Siehe dazu im Abschnitt *Erste Schritte , Drittes Programm: Programme im Terminalfenster ohne Grafik* )

### 5.54.3 Beispiel 3: (mehrere Parameter)

Werden mehrere Werte übergeben , dann müssen sie in der Klammer nach dem Funktionsnamen durch Kommata voneinander getrennt werden.

In Beispiel 2 gab die Funktion unterschiedlichen Text zurück, abhängig von der Zeichenkette, die ihr übergeben wurde. Andere Funktionen benötigen zusätzliche Parameter, um ihr zu sagen, wie sie diese Parameter handhaben soll. Zum Beispiel übernimmt die Funktion **Mid\$** (= **Text aus der Mitte herauschneiden**)

- einen Parameter für den benutzen Text,
- einen Parameter um den Startpunkt zu fixieren (gemessen in Zeichenzahl),
- und dann einen dritten und optionellen Parameter um festzulegen, wieviele Zeichen zurückgegeben werden sollen.

Im folgenden Beispiel werden alle Möglichkeiten der Stringbearbeitung = Zeichenfolgenmanipulation die Gambas kennt, zusammengefasst. Auch die Funktion **Mid\$** ist dabei. Sie brauchen nur eine leer Form um das Programm in Gang zu bringen.

```
PUBLIC SUB Form_Open()
  DIM Wort$ AS String
  Wort$ = "Kleinstaatengrenze"
  PRINT Len(Wort$)
  PRINT Left$(Wort$,7)
```

```
PRINT Right$(Wort$,6)
PRINT Mid$(Wort$,3,5)
PRINT Mid$(Wort$,6)
PRINT Instr(Wort$,"eng")
PRINT Instr(Wort$,"Tomate")
PRINT Instr(Wort$,"eng", 12)
PRINT Instr(Wort$,"e",11)
END
```

### Ergebnis

- 18 ( Länge des Wortes )
- Kleinst ( Die 7 ersten Buchstaben )
- grenze ( Die 6 letzten Buchstaben )
- einst ( 5 mittlere Buchstaben beginnend beim 3.)
- staategrenze ( Buchstaben beginnend ab dem 6. Buchstaben )
- 11 ( "eng" steht an der 11.Position im Wort )
- 0 ( Tomate findet sich nicht )
- 0 ( eng findet er nicht, wenn erst ab der 12.Position gesucht werden soll )
- 11 ( "e" findet er an 11.Position , wenn erst ab der 11.Position gesucht werden soll.)

**ACHTUNG:** In VB kann man den Befehl Mid\$() nutzen um eine Buchstabenfolge aus einem Text herauszuschneiden und dafür einen anderen Text einzusetzen. Das geht in Gambas nicht.

### Beispiel:

```
rem VB:
MyString = "Der Hund springt"
Mid$(MyString, 5, 4) = "Ball"
```

Ergebnis MyString = "The fox jumps".

In Gambas geht das nicht, Dafür kann man folgendes machen:

```
MyString = "Der Hund springt"
MyString = Left$(MyString, 4) & "Ball" & Mid$(MyString, 9)
```

Ein komplettes Programm dazu ( ohne Grafik) schaut so aus:

```
STATIC PUBLIC SUB Main()
DIM MyString AS String
MyString = "Der Hund springt"
PRINT MyString
'in VB moeglich: Mid$(MyString, 5, 4) = "Ball" in Gambas nicht
MyString = Left$(MyString, 4) & "Ball" & Mid$(MyString, 9)
PRINT MyString
END
```



### 5.54.4 Befehle zur Stringbearbeitung = Zeichenfolgenmanipulation

&	Zusammenkleben von Strings
InStr()	Man bekommt die Position eines Zeichens oder Substrings zurück
left()	Substrings von Links beginnend abtrennen
like	Vergleich von 2 Strings oder Substrings
LTrim()	Wegschneiden von Leerzeichen am Beginn eines Strings
mid()	Schneidet einen Substring aus der Mitte heraus
right()	Substrings von Rechts beginnend abtrennen
RTrim()	Wegschneiden von Leerzeichen am Ende eines Strings
split()	Aufteilung eines Strings in einzelne Wörter
Trim()	Wegschneiden von Leerzeichen am Beginn und am Ende eines Strings
CStr()	Einen Ausdruck in einen String umwandeln.
Str()	Einen Ausdruck in einen String umwandeln. (Umkehrfunktion oder Gegenteil von val())
val()	Verwandelt einen String in eine Zahl, eine Flie&szlig;kommazahl oder sonst etwas.

## 5.55 Funktionen selber programmieren

Richtig versteht man Funktionen erst, wenn man sie selbst programmiert hat. Dann erkennt man auch Ihre praktische Bedeutung. Man kann Sie nämlich meist völlig unverändert in andere Programme übernehmen.

### 5.55.1 Hoch3

Im Folgenden soll eine Funktion programmiert werden, die eine Zahl hoch 3 nimmt. Dazu starten Sie ein neues grafisches Gambasprojekt: Auf der Form platzieren Sie 2 Textboxen und einen Befehlsbutton. Die Textboxen befreien Sie mittels Eigenschaftsfenster F4 von ihrem Text. Beim Commandbutton geben Sie folgenden Code ein:

```
PUBLIC SUB Button1_Click()
DIM b AS Integer
b = Val(Textbox1.text)
textbox2.Text = Cstr(Hoch(b))
END
'-Hier fängt die Funktion an -----
PUBLIC FUNCTION Hoch(a AS Integer ) AS Integer
a = a^3
RETURN a
END
```

Vergessen Sie nicht den Return Befehl in Ihrer Funktion. Sonst bekommen Sie nichts zurück geliefert.

Die Programmierung von Funktionen in Gambas unterscheidet sich etwas von der Programmierung in VB. In Gambas kann man den Funktionsnamen in der Funktion nicht aufrufen, in VB ist das möglich.

### 5.55.2 Dritte Wurzel ( hoch 1/3 )

Im Folgenden soll eine Funktion programmiert werden, die aus einer Zahl die dritte Wurzel zieht . Dazu starten Sie ein neues grafisches Gambasprojekt: Auf der Form platzieren Sie 2 Textboxen und einen Befehlsbutton. Die Textboxen befreien Sie mittels Eigenschaftsfenster F4 von ihrem Text. Beim Commandbutton geben Sie folgenden Code ein:

```
PUBLIC SUB Button1_Click()
DIM b AS Integer
b = Val(Textbox1.text)
textbox2.Text = Cstr(wurzel(b))
END
'-Hier fängt die Funktion an -
PUBLIC FUNCTION wurzel(a AS Float ) AS Float
a = a^(1/3)
RETURN a
END
```

Wenn Sie jetzt in die Textbox eine positive Zahl eingeben und auf den Befehlsbutton klicken, wird aus ihr die dritte Wurzel errechnet.

Warum wurde hier die Funktionsvariable als float-Zahl definiert. Probieren Sie dasselbe mit einer Integervariablen aus. Vergessen Sie nicht den Return Befehl in Ihrer Funktion. Sonst bekommen Sie nichts zurück geliefert.

### 5.55.3 Texte Zusammenkleben mit einer Funktion

Das nächste Beispiel Programm zeigt Ihnen den Aufruf einer Funktion zum Kleben zwei Textteile.

Sie brauchen eine Textbox und 2 Befehlsbuttons, um das Programm in Gang zu bekommen.

```
' Gambas class file
PUBLIC SUB Button1_Click()
  DIM b AS String
  Textbox1.Text = wort(b) & "Franz"
END
'-----
PUBLIC SUB Button2_Click()
  Textbox1.Text = wort & "Fritz"
  'Dieser Aufruf der Funktion ergibt eine Fehlermeldung, da der erwartete Parameter nicht
  mitübergeben wurde.
END
'-----
PUBLIC FUNCTION wort ( a AS String) AS String
  a = "Hallo "
  RETURN a
END
```

Wenn Sie auf den Button 1 Klicken wird die Funktion korrekt aufgerufen. Wenn Sie auf den Button 2 Klicken ergibt sich eine Fehlermeldung. Verstehen Sie auch warum ?. Wie könnte man den Code der Funktion umändern , so dass der Code unter dem Button 2 auch korrekt abläuft ?

Lösung:

```
' Gambas class file
PUBLIC SUB Button1_Click()
  DIM b AS String
  Textbox1.Text = wort(b) & "Franz"
END
PUBLIC SUB Button2_Click()
  Textbox1.Text = worte() & "Fritz"
END
PUBLIC FUNCTION wort ( a AS String) AS String
  a = "Hallo "
  RETURN a
END
PUBLIC FUNCTION worte() AS String
  a AS String
  a = "Hallo "
  RETURN a
END
```

### 5.55.4 Fakultät

Im folgenden Beispiel soll die Fakultät einer Zahl berechnet werden.

Die Fakultät zu einer Zahl (mathematische Darstellung: "n!", wird gesprochen: "n Fakultät") ist das Produkt der natürlichen, d.h. ganzzahlig, positiven Zahlen von 1 bis n.

Symbol: ! = Ausrufezeichen

Es gilt:

- $0! = 1$
- $1! = 1$
- $2! = 1 * 2 = 2$
- $3! = 1 * 2 * 3 = 6$
- $4! = 1 * 2 * 3 * 4 = 24$
- $5! = 1 * 2 * 3 * 4 * 5 = 120$
- $6! = 1 * 2 * 3 * 4 * 5 * 6 = 720$
- .....
- $n! = n * (n - 1)!$

Die höchste Zahl, deren Fakultät sich in Gambas Basic ohne Kunstgriffe zur Emulation eines größeren Wertebereichs berechnen lässt, ist 170. Bei dem Programm unten ist 13 die Obergrenze Sie brauchen für das Programm einen Befehlsknopf = Commandbutton und zwei Textboxen.

```
' Gambas class file
PUBLIC SUB Button1_Click()
  DIM a AS Integer
  a = Val(textbox1.text)
  ' textbox2.Text = Fakultaet(Val(textbox1.text))
  textbox2.Text = Fakultaet(a)
END
'---Hier beginnt die Funktion -----
FUNCTION Fakultaet(F AS Integer) AS Integer
  DIM k AS Integer
  DIM z AS Integer
  IF F = 0 THEN F = 1
  z = F - 1
  FOR k = 1 TO z
    F = F*k
  NEXT
  RETURN F
END
```

## 5.55.5 Binominalkoeffizient

Will man eine Summe potenzieren, dann kann man dazu gut den Binominalkoeffizienten gebrauchen .

Beispiel:

$$\begin{aligned}(a+b)^2 &= 1*a^2 + 2*a*b + 1*b^2 \\ (a+b)^3 &= 1*a^3 + 3*a^2*b + 3*a*b^2 + 1*b^3 \\ (a+b)^4 &= 1*a^4 + 4*a^3*b + 6*a^2*b^2 + 4*a*b^3 + 1*b^4\end{aligned}$$

Die Zahlen vor den verschiedenen a-und-b Gliedern werden Koeffizienten genannt und gehorchen den Regeln des Pascalschen Dreiecks. Siehe <http://www.madeasy.de/2/pascal.htm>

Die Regel des Dreiecks ist einfach: jede Zahl (außer der 1 am Rand) erhält man durch Addition der beiden Zahlen, die über ihr liegen (siehe Beispiel).

**| Bild**

Man kann die Zahlen aber auch mit der Formel für den Binominalkoeffizienten ausrechnen:

**| Bild**

Auch in der Statistik und Kombinatorik findet man immer wieder diese Formel, deswegen soll sie im folgenden programmiert werden.

Schreiben wir die Formel zunächst von der obigen grafischen Darstellung in eine Textdarstellung um:

$$\begin{aligned}N\text{-ueber-}k &= N! / k! / (N-k)! \\ \text{wobei } N! &= 1*2*3*4*....*(N-1)*N = N \text{ Fakultät}\end{aligned}$$

Dabei muß immer gelten  $N > 0$  und  $k > 0$  und  $N > k$  oder  $N = k$  .

Zu Eingabe und Ausgabe der Werte brauchen Sie 3 Textboxen und zum Starten der Berchnung einen Befehlsbutton. Das ganze schaut dann so aus:

Will man die Formel programmieren, kann man das ohne Funktionsaufruf tun, dies zeigt der folgende Programmcode:

```
' Gambas class file
PUBLIC SUB Button1_Click()
DIM N AS Integer
DIM k AS Integer
DIM F AS Integer
DIM x AS Integer
DIM nf AS Integer ' N Fakultät
DIM nmk AS Integer ' n minus k Fakultät
DIM kf AS Integer ' k Fakultät
DIM nuk AS Integer
N = Val(textbox1.text)
k = Val(textbox2.text)
```

```

F = 1
FOR x = 1 TO N
  F = F * x
NEXT
nf = F
F = 1
FOR x = 1 TO k
  F = F * x
NEXT
kf = F
F = 1
FOR x = 1 TO (n- k)
  F = F * x
NEXT
nmk = F
nuk = nf / nmk / kf
textbox3.Text = nuk
END

```

Probieren Sie jetzt einmal die Werte für  $(a+b)^4$  aus. Dabei ist  $N = 4$  und  $k$  steigt von 0 bis 4 an. Setzen Sie diese Werte 4 und 0, 4 und 1, 4 und 2, 4 und 3, 4 und 4 in Ihr Programm ein und Sie erhalten die Koeffizienten der Binomischen Gleichung und der entsprechenden Zeile im Pascalschen Dreieck.

Eleganter funktioniert die Programmierung, wenn man die Fakultätsberechnung einer Funktion übergibt.

### 5.55.6 Pascalsche Dreieck

Wenn man den Binominalkoeffizienten  $n$  über  $k$  auch einer Funktion anvertraut, kann man in einer Doppelschleife alle Werte des Pascalschen Dreiecks beispielsweise von Zeile 1 bis Zeile 20 berechnen.

Da die Ausgabe in einer grafischen Form schwierig ist, kann man es im Terminalmodus programmieren:

Das Programm schaut so aus

```

For Zeile = 1 to Zeile = 12
  Berechne N ueber K
    wobei N = Zeile und k von 0 bis N wächst
    Ausgabe als Print zeilenweise getrennt durch Kommas
Next Zeile

```

Verfeinern wird das Programm in der inneren Schleife

```

For Zeile = 1 to Zeile = 20
  For K = 0 To N = zeile
    Berechne N ueber K
      Print N ueber k;
  Next k
Print ' fuegt einen Zeilenvorschub ein
Next Zeile

```

Das komplette Programm schaut dann so aus :

```
' Gambas class file
STATIC PUBLIC SUB Main()
  DIM Zeile AS Integer
  DIM N AS Integer
  DIM k AS Integer
  DIM F AS Integer
  DIM x AS Integer
  DIM nf AS Integer ' N Fakultät
  DIM nmk AS Integer ' n minus k Fakultät
  DIM kf AS Integer ' k Fakultät
  DIM nuk AS Integer
  FOR Zeile = 1 TO 12
    N = Zeile
    FOR k = 0 TO N
      F = 1
      FOR x = 1 TO N
        F = F * x
      NEXT
      nf = F
      F = 1
      FOR x = 1 TO k
        F = F * x
      NEXT
      kf = F
      F = 1
      FOR x = 1 TO (n- k)
        F = F * x
      NEXT
      nmk = F
      nuk = nf / nmk / kf
      PRINT nuk & " ";
    NEXT
  PRINT
NEXT
END
```

Das Ergebnis im Direktfenster schaut so aus.

```
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
```

Ein schiefes , aber doch erkennbares Pascalsches Dreieck.

Man kann das Programm deutlich vereinfachen, wenn man sich  $n$  über  $k$  und die Fakultät als Funktion ausrechnen laesst.

## 5.56 Vordefinierte Konstanten

gb.Null	Null value
gb.Boolean	Boolean value
gb.Byte	Byte integer number
gb.Short	Short integer number
gb.Integer	Integer number
gb.Float	Floating point number
gb.Date	Date and time value
gb.String	Character string
gb.Variant	Variant
gb.Object	Object reference

### 5.56.1 Datei Typen

gb.File	Regular file
gb.Directory	Directory
gb.Device	Special file for a device
gb.Pipe	Named pipe
gb.Socket	Special file for a socket
gb.Link	Symbolic link

### 5.56.2 String Konstanten

gb.NewLine	Newline character. Equivalent to Chr\$(10)
gb.Tab	Tab character. Equivalent to Chr\$(9)

### 5.56.3 Sortierungen

gb.Binary	Binary sort
gb.Case	Case insensitive sort
gb.Lang	Language based sort



### 5.56.4 Wochentage

gb.Monday	Monday
gb.Tuesday	Tuesday
gb.Wednesday	Wednesday
gb.Thursday	Thursday
gb.Friday	Friday
gb.Saturday	Saturday
gb.Sunday	Sunday

### 5.56.5 Vorgegebene Zahlenformate

gb.GeneralNumber	Write a number with twelve decimal digits. Use scientific format if its absolute value is lower than $10^{-4}$ or greater than 107.
gb.Fixed	Equivalent to "0.00"
gb.Percent	Equivalent to "###%"
gb.Scientific	Write a number with its exponent and eighteen decimal digits.

### 5.56.6 Vorgegebene Datums- und Zeitformate

gb.GeneralDate	Write a date only if the date and time value has a date part, and write a time only if it has a date part.
gb.LongDate	Long date format.
gb.MediumDate	Medium date format.
gb.ShortDate	Short date format.
gb.LongTime	Long time format.
gb.MediumTime	Medium time format.
gb.ShortTime	Short time format.

### 5.56.7 Verschiedene Formate

---

gb.Standard	Use gb.GeneralNumber for formatting numbers and gb.GeneralDate for formatting dates and times.
-------------	--

Sortierverfahren werden bei Datenbanken, Tabellen, Arrays und vielem mehr gebraucht. Einige einfache Sortierverfahren werden im Folgenden beschrieben.

## 5.57 Ripplesort

Sie haben eine Zahlenfolge 13,12,15,11,14 . Diese Zahlen wollen Sie in aufsteigender Reihenfolge ordnen, so daß die kleinste Zahl am Anfang und die größte am Ende steht.

Wie das geht zeigt folgendes Programm. Es nutzt ein Array ( siehe [Gambas: Arrays](#)) und mehrere Hilfsvariablen . Dieses Array wird mehrfach durchsortiert, bis die Reihenfolge stimmt.

Sie brauchen eine Form und einen Commandbutton, um das Programm in Gang zu bringen.

```
' Gambas class file
ar AS Integer[]
PUBLIC SUB Form_Open()
'schon beim Öffnen der Form wird der Array mit lauter Nullen gefüllt.
x AS Integer
ar = NEW Integer[]
ME.Text = "Sortieren"
FOR x = 0 TO 4
  ar.Add(Str(0))
NEXT
END
PUBLIC SUB Button1_Click()
c AS Integer
j AS Integer
n AS Integer
y AS Integer
element AS Integer
y = 0
n = ar.count
ar[0] = 13
ar[1] = 12
ar[2] = 15
ar[3] = 11
ar[4] = 14
REPEAT
c = 0
FOR j = 0 TO n - 2
  PRINT j,y,ar[0],ar[1],ar[2],ar[3],ar[4]
  IF ar[j] > ar[j + 1] THEN
    y = ar[j]
```

```

        ar[j] = ar[j + 1]
        ar[j + 1] = y
        c = 1
    ENDIF
NEXT
UNTIL c = 0
END

```

Die Ausgabe im Direktfenster schaut dann so aus:

```

PRINT j,y,ar[0],ar[1],ar[2],ar[3],ar[4]
0      0      1      2      3      4      5
0      0      13     12     15     11     14
1 13 12 13 15 11 14
2 13 12 13 15 11 14
3 15 12 13 11 15 14
0 15 12 13 11 14 15
1 15 12 13 11 14 15
2 13 12 11 13 14 15
3 13 12 11 13 14 15
0 13 12 11 13 14 15
1 12 11 12 13 14 15
2 12 11 12 13 14 15
3 12 11 12 13 14 15
0 12 11 12 13 14 15
1 12 11 12 13 14 15
2 12 11 12 13 14 15
3 12 11 12 13 14 15

```

Das Sortierverfahren steckt in der Repeat until Schleife . Darin steckt eine for next Schleife und darin steckt wieder eine Vertauschung mit if then . Das ist schon ein bißchen schwer zu kapieren. Mit der ausführlichen Ausgabe im Direktfenster kann man das Sortieren genau verfolgen. Man kann den Array ändern und andere Zahlen einfügen. Dann kann man wieder zuschauen, wie die Sortierung abläuft. Der Kern ist eine Vertauschung zweier benachbarter Zahlen, wenn die linke Zahl größer als die rechte ist. Der Rest der Zeile bleibt stehen. Dann erfolgt ein neuer Durchlauf der For Next Schleife , bei dem nichts passiert da in Position ar[1] des Arrays jetzt 13 steht und  $13 < 15$  ist. ( das Array beginnt mit der zahl 0 und hört mit n-1 auf, wenn es n Elemente hat. Beim nächsten Durchlauf der For Next Schleife wird die Position  $ar[2] = 15$  mit  $ar[3] = 11$  verglichen . Da  $15 > 11$  wird getauscht. Dann ist die For next Schleife das erste Mal zu Ende, da  $j = 3 = n-2$  und die Repeatschleife wird ein zweites mal durchlaufen an, denn der Hilfsparameter c zeigt einen stattgefundenen Tausch an. Und so geht das immer weiter bis  $c = 0$  , d.h. keine Vertauschung mehr nötig war.

### 5.57.1 Ripplesort mit Zufallszahlen

Das folgende Programm ist vom obigen abgeleitet. Erst werden per Zufall 5 Zahlen zwischen 1 und 5 erzeugt. Diese werden dann sortiert. Sie brauchen eine Textarea und 2 Commandbuttons, um das Programm in Gang zu bringen.

```
' Gambas class file
ar AS Integer[]
PUBLIC SUB Form_Open()
'schon beim Öffnen der Form wird der Array mit lauter Nullen gefüllt.
x AS Integer
ar = NEW Integer[]
ME.Text = "Sortieren"
FOR x = 0 TO 4
  ar.Add(Str(0))
NEXT
END
PUBLIC SUB Button1_Click()
x AS Integer
r AS Integer
element AS Integer
txt AS String
z AS Integer
txt = ""
'der Array wird komplett auf Null gestellt
FOR x = 0 TO 4
  ar[x] = 0
NEXT
Randomize()
FOR x = 0 TO 4
  r = Int(Rnd(1,6))
ar[x] = r
NEXT
x = 0
FOR EACH element IN ar
PRINT x,element
txt = txt & x & " " & element & Chr(10)
  x = x + 1
NEXT
textareal.Text = txt
END
PUBLIC SUB Button2_Click()
c AS Integer
j AS Integer
n AS Integer
y AS Integer
element AS Integer
txt AS String
y = 0
n = ar.count
REPEAT
c = 0
FOR j = 0 TO n - 2
  PRINT j,y,ar[0],ar[1],ar[2],ar[3],ar[4]
  IF ar[j] > ar[j + 1] THEN
    y = ar[j]
    ar[j] = ar[j + 1]
    ar[j + 1] = y
    c = 1
  ENDIF
NEXT j
y = ar[c]
ar[c] = ar[n - 1]
ar[n - 1] = y
END REPEAT
END
```

```

NEXT
UNTIL c = 0
FOR EACH element IN ar
txt = txt & element & Chr(10)
NEXT
textareal.Text = textareal.Text & txt
END

```

## 5.57.2 Zahlenliste beliebiger Länge durchsortieren

Im folgenden Beispiel wird der Ripplesort algorithmus zum sortieren einer beliebig langen Werteliste benutzt. Sie brauchen 2 Textareas und 2 Commandbuttons um das Programm in Gang zu bringen.

Code:

```

' Gambas class file
PUBLIC SUB Button1_Click()
c AS Integer
j AS Integer
n AS Integer
y AS Variant
DIM Liste AS String[]
element AS String
txt AS String
text AS String
text = Textareal.Text
liste = Split(text,Chr(10))
y = 0
n = liste.length
REPEAT
c = 0
FOR j = 0 TO n - 2
'PRINT j,y,liste[0],ar[1],ar[2],ar[3],ar[4]
IF Val(liste[j]) > Val(liste[j + 1]) THEN
y = Val(liste[j])
liste[j] = liste[j + 1]
liste[j + 1] = Str(y)
c = 1
ENDIF
NEXT
UNTIL c = 0
FOR EACH element IN liste
txt = txt & Str(element) & Chr(10)
NEXT
'PRINT txt
textarea2.Text = ""
textarea2.Text = txt
END
PUBLIC SUB Button2_Click()
'Beispielliste
textareal.Text = "114,3"
textareal.Text = textareal.Text & Chr(10) & "135,7"
textareal.Text = textareal.Text & Chr(10) & "104,8"
textareal.Text = textareal.Text & Chr(10) & "118,5"
textareal.Text = textareal.Text & Chr(10) & "125,7"
textareal.Text = textareal.Text & Chr(10) & "121,4"

```

```
textareal.Text = textareal.Text & Chr(10) & "122,4"  
textareal.Text = textareal.Text & Chr(10) & "96,8"  
textareal.Text = textareal.Text & Chr(10) & "118,9"  
textareal.Text = textareal.Text & Chr(10) & "120"  
textareal.Text = textareal.Text & Chr(10) & "112,2"  
textareal.Text = textareal.Text & Chr(10) & "127,9"  
textareal.Text = textareal.Text & Chr(10) & "122,8"  
textareal.Text = textareal.Text & Chr(10) & "128,9"  
textareal.Text = textareal.Text & Chr(10) & "120,3"  
END
```

## 5.58 Ereignisse der Steuerelemente

Jedes Steuerelement hat verschiedene Ereignisse, die mit ihm verknüpft sind. Das häufigste Ereignis und meistens auch die Voreinstellung ist das Click Ereignis. Es wird benutzt, wenn etwas passieren soll, wenn man das Steuerelement einmal anklickt. Relativ häufig wird auch das Enter oder das Keypressereignis benutzt. Viele andere Ereignisse dagegen wird man eher selten nutzen. Was der genaue Unterschied ist zwischen den Ereignissen und den Methoden eines Steuerelementes, das ist wohl nicht so ganz klar definiert.

Typisches Beispiel mit einem Befehlsknopf auf der Form:

```
PUBLIC SUB Button1_Click()  
    Message.Info("Hallo Welt")  
END
```

Das Ereignis ist der einfache Click. Die ausgelöste Folge ist die Ausgabe einer Meldung. (Ein 'Click' mit der Maus besteht aus dem Herunterdrücken UND Loslassen der Maustaste, erst mit dem Loslassen ist das Ereignis komplett, wünscht man eine Reaktion schon beim Herunterdrücken, so wähle man das Ereignis 'MouseDown'. Ein 'Click' mit der Tastatur wird ausgelöst, wenn der Knopf den Fokus hat und die Leertaste gedrückt und losgelassen wird.)

Ein weiteres 'Click'-Ereignis mit der Tastatur ist der Zugriff auf Befehlsknöpfe mit einer Tastenkombination ('Alt' und ein Zeichen des Knopfzeichners. Dazu im Eigenschaftfenster vor den Buchstaben, welcher mit der 'Alt'-Taste zum Auslösen des 'Click'-Ereignisses dienen soll, ein '&' stellen. Daraufhin wird bei laufendem Programm dieser Buchstabe unterstrichen. Zur Auslösung des Ereignisses muß der entsprechende Knopf nicht den Fokus haben.

Die weitere Bedienung des Befehlsknopfes mit der Tastatur:

```
PUBLIC SUB Button1_KeyPress()  
    IF key.Code = key.Space OR key.Code = key.Return OR key.Code=key.Enter THEN  
        Message.Info("Hallo Welt")  
    ENDIF  
END
```

Die Meldung wird ausgegeben, wenn der Befehlsknopf den Fokus hat, und eine der Tasten 'Leerzeichen', 'Return' oder 'Enter' gedrückt wurde ('Return' ist die Taste beim alphanumerischen Block, 'Enter' befindet sich beim Ziffernblock der Tastatur).

Wenn bei dem Knopf gleichzeitig das 'Click'-Ereignis abgefragt wird, muß das Code-Fragment 'key.Code = key.Space OR ' entfallen, da sonst 2 Subroutinen aufgerufen werden.

Alle Ereignismöglichkeiten, die mit einem Steuerelement ausgelöst werden können, finden Sie heraus, wenn Sie mit der rechten Maustaste auf das Steuerelement klicken und die Zeile *Ereignis* wählen.

Beim Steuerelement Button ( Befehlsknopf) finden sich beispielsweise folgende programmierbaren Ereignisse:

```
Click DblClick Drag DragMove Drop Enter GotFocus KeyPress KeyRelease
Leave LostFocus Menu MouseDown MouseMove MouseUp MouseWheel
```

Probieren Sie einmal folgendes Ereignis aus:

```
PUBLIC SUB Button1_Enter()
    Message.Info("Funktioniert schon wenn die Maus auf den Button kommt")
END
```

Oder folgende Variante:

```
PUBLIC SUB Button1_Leave()
    Message.Info("Funktioniert erst wenn die Maus den Button verläßt")
END
```

Wenn man die Ereignisse kombiniert, kommen sich die Ereignisse manchmal in die Quere:

Probieren Sie zb folgendes Programm:

```
PUBLIC SUB Button1_Leave()
    ME.Text = "Funktioniert erst wenn die Maus den Button verlaesst."
END
PUBLIC SUB Button1_Click()
    Message.Info("Funktioniert schon wenn geklickt wird")
END
```

Wenn man hier den Befehlsknopf anklickt und verläßt ihn nicht, wird trotzdem schon die Überschrift der Form geändert.

## 5.59 Ereignisse programmieren mit EVENT, STOP EVENT, RAISE

Ereignisse selbst programmieren macht nur selten Sinn. Man kann beispielsweise die Tasteneingabe einer Textbox überprüfen, wenn man verhindern will, daß

etwas anderes als Zahlen eingegeben wird. Es handelt sich hier um eine Art Fehlermanagement.

**Beispiel:** Sie brauchen eine Textbox auf Ihrer Form.

```
'Meine Textbox verarbeitet nur Ziffern.  
PUBLIC SUB TextBox1_KeyPress()  
  IF Instr("0123456789", Key.Text) = 0 THEN  
    STOP EVENT  
  ENDF  
END SUB
```

Wenn Sie etwas anderes eingaben als Ziffern, verweigert die Textbox die Annahme. Ziffern hingegen werden akzeptiert und in der Textbox dargestellt.

Siehe auch [Gambas:\\_Textbearbeitung#Texteingabe](#)

Beim folgenden Beispiel wird ein Ereignis nach dem 4.Mal Klicken auf die Taste ausgelöst.

Sie brauchen eine Form mit einem Befehlsbutton darauf. Außerdem brauchen Sie eine neue Klasse *Class1*. Diese erhalten Sie, wenn Sie im Projektmenü auf die Zeile Klasse mit der rechten Maustaste klicken und Neue Klasse wählen. Den vorgeschlagenen Namen Class1 sollten sie einfachheitshalber beibehalten:

Folgender Code für die Form wurde verwendet:

```
PUBLIC SUB Form_Open()  
  MyClass=NEW Class1 AS "Beobachte"  
END  
PRIVATE MyClass AS Class1  
PUBLIC SUB Button1_Click()  
  MyClass.testen ()  
END  
PUBLIC SUB Beobachte_ereignis(Data AS String)  
  Message.Info (Data)  
END
```

Folgender Code für die Klasse wurde verwendet:

```
PRIVATE zaehler AS Integer  
EVENT ereignis (Data AS String)  
PUBLIC SUB testen()  
  zaehler=zaehler + 1  
  IF zaehler>3 THEN  
    RAISE ereignis("Taste > 3 mal gedrückt")  
    zaehler=0  
  END IF  
END
```



## 5.60 Fehler abfangen mit Catch

Der Befehl *CATCH* markiert den Beginn einer Fehleroutine innerhalb einer Funktion oder einer Procedure.

Der Catch Teil wird immer dann ausgeführt, wenn ein Fehler bei der Ausführung der Funktion auftritt.

Der Fehler kann innerhalb der Prozedur selber auftreten oder aber in einer anderen Funktion, die aus der Prozedur heraus aufgerufen wurde. Dies gilt allerdings nur unter der Voraussetzung, dass die aufgerufene Funktion selber keinen CATCH Befehl enthält.

Je tiefer der Catch Teil zu finden ist , desto größere Priorität hat er.

Wenn ein Fehler im Catch Teil selber auftaucht, dann wird er nicht korrigiert. Der Catch Teil schützt sich also nicht selber.

Falls eine Finally Teil in der Funktion auftaucht, muss er dem Catch Teil vorausgehen.

Siehe FINALLY .

### 5.60.1 Beispiel 1

Das Programm speichert der Inhalt der Textarea als Datei. Die Fehleroutine fängt den Abbrechen Befehl im Dateiauswahldialog ab.

Sie brauchen eine Textarea und einen Befehlsbutton auf Ihrer Form, um das Programm in Gang zu bringen.

### 5.60.2 Der Code:

```
PUBLIC SUB Form_Open()  
    Textarea1.Text = "Dies ist ein Test"  
END  
PUBLIC SUB Button1_Click()  
    Dialog.SaveFile()  
    File.Save(Dialog.Path, TextArea1.Text)  
    CATCH  
    IF Error THEN RETURN  
END
```

Siehe auch [Gambas:\\_Dateien#Datei\\_speichern](#)

### 5.60.3 Beispiel 2

Sie haben ein Programm geschrieben, welches die Kreisfläche aus dem Radius berechnen soll. Sie wollen verhindern, dass in die Textbox mit der Radiuseingabe etwas anderes als eine Zahl eingegeben wird. Dazu ergänzen Sie Ihren Code durch eine Fehlerroutine mit *CATCH*.

```
CATCH
message.Info("Bitte eine Zahl eingeben")
END
```

Das komplette Programm schaut so aus:

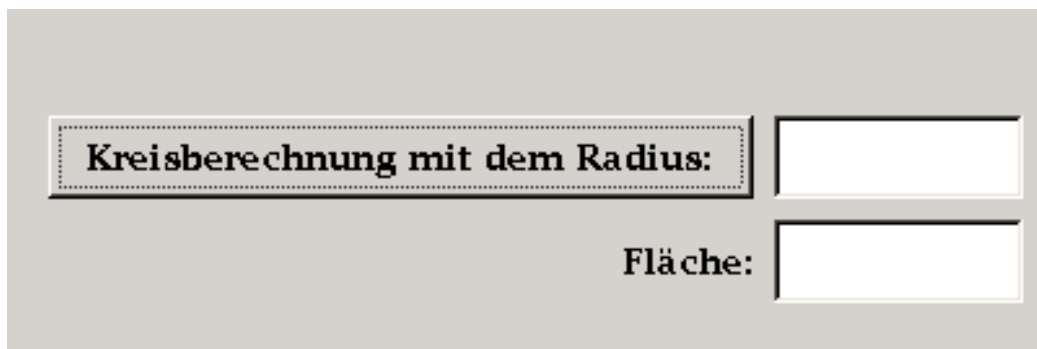


Abbildung 92: Bild:Gambaskreis.png

Sie brauchen:

- 2 Textboxen
- 1 Befehlsbutton
- 1 Labelfeld

und das Programm in Gang zu bringen.

#### Der Code

```
PUBLIC SUB Form_Open()
  Textbox1.Text = ""
  Textbox2.Text = ""
  Button1.Text = "Kreisberechnung mit dem Radius:"
  Label1.Text = "Fläche:"
  Label1.Alignment = 66
END
PUBLIC SUB Button1_Click()
  r AS Float
  F AS Float
  IF Textbox1.Text = "" THEN
```

```

r = 0
Textbox1.Text = Str(0)
ELSE
r = Val(textbox1.text)
ENDIF
F = Pi*r^2
Textbox2.Text = Str(Round(F,-2))
CATCH
message.Info("Bitte eine Zahl eingeben")
END

```

Probieren Sie ganz bewußt eine Fehleingabe aus. Geben Sie im Feld für den Radius einen Buchstaben ein. Was passiert dann ? Jetzt versuchen Sie dasselbe noch einmal und schalten die Fehleroutine CATCH nach Markierung mit der rechten Maustaste und *Block auskommentieren* ab.

### 5.60.4 Beispiel 3

Das folgende Beispiel zeigt einen etwas komplizierten Code aus der Gambashilfe.

**Beispiel:**

```

' Print a file to the screen
SUB PrintFile(FileName AS STRING)
DIM hFile AS File
DIM sLig AS STRING
OPEN FileName FOR READ AS #hFile
WHILE NOT EOF(hFile)
LINE INPUT #hFile, sLig
PRINT sLig
WEND
FINALLY ' Always executed, even if a error raised
CLOSE #hFile
CATCH ' Executed only if there is an error
PRINT "Cannot print file "; FileName
END

```

## 5.61 Fehler abfangen mit TRY

Try ist im Grunde genommen das selbe wie Catch nur das hierbei nicht die gesamte Funktion nach Fehlern geprüft wird sondern nur eine spezielle Anweisung.

### 5.61.1 Beispiel

In diesem Beispiel wird eine Datei in dem Homeverzeichnis des aktuellen Users erstellt. Wenn der Zugriff zu dem Zielort allerdings gesperrt ist (z.b. wenn die

Datei oder das Verzeichnis read-only ist), wirft die Try-Anweisung einen Error der durch das Schlüsselwort ERROR überprüft werden kann.

```
PUBLIC SUB Main()  
  TRY File.Save("/home/" & User.Name & "/file.txt", "Hallo " & User.Name)  
  IF ERROR THEN  
    PRINT "Folgender Fehler ist aufgetreten: " & Error.Text  
    PRINT "Und zwar genau hier: " & Error.Where  
  ELSE  
    PRINT "Ich habe erfolgreich eine Datei in deinem Homeverzeichnis erstellt!"  
  ENDIF  
END
```

# Kapitel 6

## Anhang

### 6.1 F11

Das Direktfenster wird mit **F11** aufgerufen.

### 6.2 Direkteingabe in das Direktfenster ohne Programm

Willkommen zur Gambas Konsole. Um einen Ausdruck auszuwerten, geben Sie '?' gefolgt vom Ausdruck ein und drücken Return.

Beispiel:

```
? 5 + 3
```

Ausgabe: 8

### 6.3 Ausgabe im Direktfenster mit dem PRINT Befehl

Die Ausgabe des Printbefehls erfolgt im Unterschied zu Visual Basic im Direktfenster.

Beispiel. Sie brauchen nur eine leer Form, um dieses Programm in Gang zu bringen. Start mit **F5**.

```
PUBLIC SUB Form_Open()  
  PRINT "Hallo Welt!"  
END
```



Abbildung 93: Bild:HalloWeltDirektfenster.png

## 6.4 Programmierung ohne Grafik nur im Direktfenster

Sie können Gimp auch für Programme ohne die grafische Oberfläche nutzen. Diese Programme laufen im Direktfenster (= Terminalfenster) auf der Konsole ab. Mit Visual Basic gibt es diese Möglichkeit nicht mehr.

Starten Sie Gimp wie gewohnt.

Wählen Sie **Neues Projekt**.

Folgen Sie dem Wizard und drücken Sie **Nächster**.

Bei *Typ des Projektes auswählen* wählen Sie **Terminal-Projekt erzeugen**.

Folgen Sie dem Wizard und drücken Sie **Nächster**.

Geben Sie ihrem Projekt einen Namen z.B. **1mal1**.

Folgen Sie dem Wizard und drücken Sie **Nächster**.

Wählen Sie das Verzeichnis, in dem ihr Projekt als Unterverzeichnis abgespeichert werden soll. (Meist belässt man die Voreinstellung /gambas-0.99)

Folgen Sie dem Wizard und drücken Sie **Nächster**.

Die Daten Ihres Projektes werden zusammengefasst dargestellt.

Drücken Sie **OK**.

Jetzt erscheint das Projektfenster. Drücken Sie mit der rechten Maustaste auf das Verzeichnis **Klassen**.

Drücken Sie dann mit der linken Maustaste auf **Neu** und **Klasse**.

Lassen Sie den Namen bei **Class1** als Startklasse und Drücken Sie **OK**.

Geben Sie als Beispiel folgendes Programm ein:

```
STATIC PUBLIC SUB Main()  
DIM x AS Integer  
DIM y AS Integer  
  
FOR x = 1 TO 10  
  FOR y = 1 TO 10  
    PRINT x;" mal ";y; " = "; x*y  
  NEXT  
NEXT  
  
END
```

Drücken Sie **F5** um es zu starten.

Schauen Sie sich im Direktfenster das Ergebnis an. Eventuell mit **Alt + Tab** zwischen den Fenster hin und herschalten.

Beenden Sie das Programm mit einem Klick auf das **rote Quadrat** im Projektfenster. Oder mit **ALT + F4**.

Beenden Sie Gambas mit **Datei Beenden** oder mit **Strg + Q** oder programmieren Sie noch etwas weiter.

## 6.5 Frage:

Wie kann man ein nicht grafisches Programm als ausführbare Datei auf dem Desktop zugänglich machen?

Wieso funktionieren Arrays im Terminalmodus nicht?

Wie kann man das Direktausgabefenster programmtechnisch verkleinern?

## 6.6 Datentypen

gb.Null	Nullwert
gb.Boolean	Wahrheitswert
gb.Byte	Byte Ganzzahl
gb.Short	Kurze Ganzzahl
gb.Integer	Ganzzahl
gb.Float	Fließkommazahl
gb.Date	Datum und Uhrzeit
gb.String	String (Text)
gb.Variant	Variant
gb.Object	Objekt Referenz

## 6.7 Dateitypen

gb.File	Reguläre Datei
gb.Directory	Verzeichnis
gb.Device	Special file for a device
gb.Pipe	Named pipe
gb.Socket	Special file for a socket
gb.Link	Symbolischer Link

## 6.8 String (Text) Konstanten

gb.NewLine	Zeilenumbruch. Entspricht Chr\$(10)
gb.Tab	Tabulator. Entspricht Chr\$(9)

## 6.9 Sortierarten

gb.Binary	Binaersortierung
gb.Case	Groß- und Kleinschreibung wird nicht berücksichtigt



gb.Lang	Sprachbasierte Sortierung
---------	---------------------------

## 6.10 Wochentage

gb.Monday	Montag
gb.Tuesday	Dienstag
gb.Wednesday	Mittwoch
gb.Thursday	Donnerstag
gb.Friday	Freitag
gb.Saturday	Samstag
gb.Sunday	Sonntag

## 6.11 Vordefinierte Zahlenformate

gb.GeneralNumber	Schreibt eine Zahl mit 12 dezimalen Ziffern. Benutzen sie die wissenschaftliche Notation , wenn der absolute Betrag kleiner als 1 / 10000 oder größer als 10 000 000.
gb.Fixed	entspricht "0.00"
gb.Percent	entspricht "###%"
gb.Scientific	Schreibt eine Zahl mit seinem Exponenten und 18 dezimal Ziffern.

## 6.12 Vordefinierte Datums und Zeitformate

gb.GeneralDate	Write a date only if the date and time value has a date part, and write a time only if it has a date part.
gb.LongDate	Long date format.
gb.MediumDate	Medium date format.
gb.ShortDate	Short date format.
gb.LongTime	Long time format.
gb.MediumTime	Medium time format.
gb.ShortTime	Short time format.

## 6.13 Verschiedene Formate

gb.Standard	Use gb.GeneralNumber for formatting numbers and gb.GeneralDate for formatting dates and times.
-------------	--

## 6.14 Weitere Konstanten

### 6.14.1 Farben

Black, Blue, Cyan, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, DarkYellow, Default, Gray, Green, LightGray, Magenta, Orange, Pink, Red, Transparent, Violet, White, Yellow

### 6.14.2 Tasten

BackSpace, BackTab, CapsLock, Delete, Down, End, Enter, Esc, Escape, F1, F10, F11, F12, F13, F14, F15, F16, F17, F18, F19, F2, F20, F21, F22, F23, F24, F3, F4, F5, F6, F7, F8, F9, Help, Home, Insert, Left, Menu, NumLock, PageDown, PageUp, Pause, Print, Return, Right, ScrollLock, Space, SysReq, Tab, Up

### 6.14.3 Rahmen

Etched, None, Plain, Raised, Sunken

## 6.15 Konstanten

Konstanten	Beispiele
Der true Wert.	TRUE
Der false Wert.	FALSE
Ganze Zahlen.	0, 123, -32769
Hexadezimale Kurzzahlen.	&H1F5, &HFFFF, &FFFF
Hexadezimal gekennzeichnete Zahlen.	&H10BF332E, &10BF332E

Hexadezimal unsigned Zahlen.	&H8000&, &HFFFF&
Binärzahlen.	&X1010010101, %101001011
Fließkommazahlen.	1.0, -5.345219E+45
Textkonstanten.	"Hallo Welt !"
String constants to be translated.	("This software is cool")
Null Konstante / Leerer Text.	NULL

## 6.16 Escape Sequenzen

Textkonstanten können folgende Escape Sequenzen enthalten:

Escape Sequenz	ASCII Entsprechung	Entsprechung
\n	CHR\$(13)	Zeilenumbruch
\r	CHR\$(10)	Wagenrücklauf
\t	CHR\$(9)	Tabulator
\"	CHR\$(34)	Doppelte Anführungszeichen
\\	CHR\$(92)	Backslash
\xx	CHR\$( &Hxx)	Hexadezimalcode

Sie können Textkonstanten in einzelnen Teilen oder zusammen schreiben. Z.B. "Gambas" " ist " "großartig" ergibt dasselbe wie "Gambas ist großartig"

## 6.17 GB Komponenten

Komponenten sind Gambas plug-ins die neue Klassen zum Interpreter hinzufügen. Jedes Projekt hat eine Liste der Komponenten, die beim Starten geladen werden müssen. Man findet diese Liste unter dem Menüpunkt *Projekt Eigenschaften*.

Ein Gambas Projekt ohne jede Komponente ist eine einfache Textanwendung. Wenn es eine echte grafische Anwendung werden soll, muß zumindest die **gb.qt** Komponente geladen sein. Um eine Datenbank zu nutzen braucht man die **gb.db** Komponente etc.

gb	Gambas interne Basis Klasse
gb.compress	Compression library

gb.corba	Corba Client
gb.crypt	Verschlüsselungskomponente
gb.db	Datenbank Zugang
gb.db.form	Daten und Datenbank Darstellung mit DataControl und DataSource
gb.debug	Gambas Debugger Hilfe
gb.eval	Gambas Auswertung mathematischer Ausdrücke
gb.form	weitere Steuerelemente für grafische Anwendungen
gb.gtk	Graphical GTK+ toolkit component
gb.image	Bildverarbeitung
gb.info	Information über die installierten Komponenten
gb.ldap	Bibliothek von OpenLdap Code
gb.net	Netzwerk Komponente
gb.net.curl	Network high-level protocols management
gb.opengl	OpenGL Bibliothek
gb.pcre	Perl-compatible Regular Expression Matching
gb.pdf	PDF komponente basierend auf der Poppler library
<b>gb.qt</b>	<b>Graphisches QT toolkit</b>
gb.qt.editor	Gambas editor with syntax highlighting
gb.qt.ext	Graphische QT toolkit Erweiterung
gb.qt.kde	KDE integration and scripting
gb.qt.kde.html	KDE web browser
gb.qt.opengl	Graphical QT toolkit opengl component
gb.sdl	Library based on SDL
gb.sdl.image	2D Library based on SDL
gb.sdl.opengl	OpenGL library based on SDL
gb.sdl.sound	Sound library based on SDL
gb.settings	Management von Voreinstellungen und Konfigurationsdateien
gb.v4l	Video capture component
gb.vb	Visual Basic compatibility

gb.xml.libxml	XML tools based on libxml
gb.xml.libxml.rpc	XML-RPC client based on libxml and libcurl
gb.xml.libxml.xslt	XSLT tools based on libxslt

### 6.17.1 Zusätzliche Komponenten laden

Zusätzliche Gambas Komponenten kann man über den Menüpunkt Projekt – Eigenschaften des Gambashauptbildschirmes erreichen.

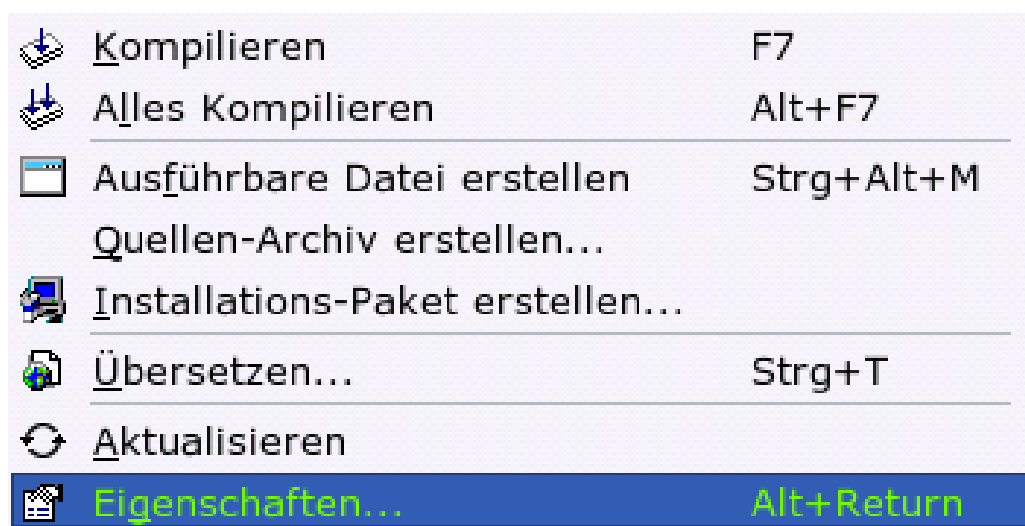


Abbildung 94: Bild:Menue Projekt.png

In der Komponentenanzeige wählt man die gewünschten Komponenten aus.

In der Werkzeugbox erscheint dann z.B. eine Schrift mit der Bezeichnung QT. Diese klickt man an und hat dann eine Reihe weiterer Steuerelemente zur Verfügung.

### 6.17.2 Liste der derzeitigen Gambaskomponenten

Komponente	Beschreibung	Kontrollen
gb	Gambas internal native classes	

gb.compress	Compression library	
gb.db	Database access component	
gb.debug	Gambas application debugger helper	
gb.eval	Gambas expression evaluator	
gb.net	Networking component	DnsClient, ServerSocket, SerialPort, Socket, UdpSocket
gb.qt	Graphical QT toolkit component	Label, TextLabel, PictureBox, ProgressBar, Button, CheckBox, RadioButton, ToggleButton, ToolButton, TextBox, ComboBox, TextArea, ListBox, ListView, TreeView, IconView, GridView, ColumnView, HBox, VBox, HPanel, VPanel, Frame, Panel, TabStrip, ScrollView, DrawingArea, Timer
gb.qt.editor	Gambas editor with syntax highlighting	GambasEditor
gb.qt.ext	Graphical QT toolkit extension component	LCDNumber, Dial, SpinBox, ScrollBar, Slider, MovieBox, TableView, HSplit, VSplit, Workspace, TextView
gb.qt.kde	KDE integration and scripting	URLLabel, ColorBox, DatePicker

### 6.17.3 Experimentelle Komponenten

WARNUNG! Folgende Komponenten befinden sich noch im BETA Stadium und können sich daher jederzeit ändern.

---

gb.qt.kde.html	KDE Webbrowser
gb.vb	Visual Basic Kompatibilität
gb.xml.libxml	XML Werkzeuge, basierend auf libxml
gb.xml.libxml.rpc	XML-RPC Klient, basierend auf libxml und libcurl
gb.xml.libxml.xslt	XSLT Werkzeuge, basierend auf libxslt

### 6.17.4 Programmdateien

Das Gambas-Programm befindet sich je nach Verwendeter Distribution bzw. Betriebssystem in verschiedenen Verzeichnissen. Unter Linux sollte ein

```
locate gambas
```

Auskunft darüber geben, wo Gambas selbst, die Beispiel-Programm usw. sich befinden.

### 6.17.5 Eigene Projekte

Die eigenen Projekte stehen unter meist unter /home/BENUTZER/gambasxx/

### 6.17.6 Konfigurationsdateien

Im versteckten Verzeichnis /home/BENUTZER/.gambas stehen Konfigurationsdateien

- file:/home/BENUTZER/.gambas/gambas.conf
- file:/home/BENUTZER/.gambas/gambas-database-manager.conf

### 6.17.7 Projektdateien

Jedes Projekt hat sein eigenes Unterverzeichnis. Beispielsweise:

/home/rho/gambas-1.0/addy	Projektverzeichnis
---------------------------	--------------------

/home/rho/gambas-1.0/addy/addy	Ausführbare Datei
/home/rho/gambas-1.0/addy/addy.png	Ein Icon für das Programm
/home/rho/gambas-1.0/addy/Form1.class	GambasCode im Klartext
/home/rho/gambas-1.0/addy/Form1.class~	Sicherungsdatei
/home/rho/gambas-1.0/addy/Form1.form	Formdatei im Klartext
/home/rho/gambas-1.0/addy/.project	Projektdatei im Klartext
/home/rho/gambas-1.0/addy/.gambas	Nochmal ein Verstecktes Unterverzeichnis. Wozu dient es?

### Beispiel einer Projektdatei

```
# Gambas Project File 1.0
Project=addy
Startup=Form1
TabSize=2
Version=0.0.2
Library=gb.qt
SnapToGrid=1
ShowGrid=1
Snap=8
Localize=0
KeepDebugInfo=0
ControlPublic=0
ExecPath=/home/name/gambas-1.0/addy/addy
Prefix=0
```

### Beispiel einer einfachen Formdatei

```
# Gambas Form File 1.0
{ Form1 Form
  Move(227,172,216,400)
  Text = ("")
  Border = Window.Fixed
  { TextArea1 TextArea
    Move(16,8,184,280)
    Text = ("")
  }
  { Button1 Button
    Move(16,296,184,32)
    Font = Font["14"]
    Text = ("Summe")
  }
  { TextBox1 TextBox
```



```

Move(16, 336, 184, 32)
Text = ("")
}
}

```

## 6.18 Wie arbeitet Gambas ?

Wenn Sie ein Programm in Gambas schreiben, dann erzeugen Sie

- eine oder mehrere **form** Dateien
  - Die form Datei enthält eine Beschreibung der gewünschten Programmoberfläche
- eine oder mehrere **class** Dateien
  - Die class datei enthält den Programmcode, den man selbst geschrieben hat.
- eine **.project** Datei
  - Die project Datei enthält Informationen über ihr Projekt wie zb den Namen und die Versionsnummer.

Diese Dateien werden durch den Compiler **gbc** zu binärem **pcode** übersetzt. Der Pcode wird im Verzeichnis `.gambas` abgespeichert.

Danach wertet der Interpreter **gbx** den pcode aus und holt sich die benötigten Qt-Komponenten dazu. Die Qt-Komponenten sind Teil von gambas. Siehe [Gambas: Komponenten](#)

Eine QT-Komponente ist ein C++/qt Programm, das die GUI Beschreibung vom pCode erhält und das die Qt-Gui erzeugt.

**gba** ist ein Archivierer. Er nimmt die `.project` Information und klebt sie mit dem p-Code zu einem ausführbaren Programm zusammen **executable file**. Zum Ablaufen lassen des Programmes braucht man noch **gbx**.

- Project – "compilieren (F7)" startet gbc.
- Project – "run (F5)" startet gbc und gbx.
- Project – "make executable (Ctrl-Alt-M)" startet gbc und gba

Wer hat Gambas entwickelt ?

Siehe auch: <http://gambas.sourceforge.net/fosdem2005.jpg> Benoit der Mann mit dem roten T-Shirt

- Benoît Minisini
  - France
    - \* Der Kopf des Ganzen
- Fabien Bodard
  - France
    - \* IDE file/project/picture selector and mascot redesign.
- Nigel Gerrard
  - United Kingdom
    - \* The MySQL database driver, the QT extended component, the SQLite database driver.
- Paul Gardner-Stephen
  - Australia
    - \* Porting Gambas on Solaris
- Daniel Campos
  - Spain
    - \* The network component, the compression component. Works on the GTK+ component too!
- Carlos F. A. Paniago
  - Brazil
    - \* Porting Gambas on FreeBSD.
- Rob Kudla
  - United States
    - \* GambasWiki webmaster and Mandrake RPM packages maintainer. The Perl Compatible Regular Expression component.
- Ronald Onstenk
  - Netherlands

- \* The find list in the Development Environnement.
- Brandon Bergren
  - Porting Gambas on CygWin.
- Laurent Carlier
  - France
    - \* The SDL component.
- José L. Redrejo Rodríguez
  - Spain
    - \* Let Gambas compile on all Debian architectures.

Gambas ist näher verwandt mit VB2005 (Vb.net) als zu VB3,4,5,6, da es echt objektorientiert ist. Die VB Varianten 3-6 werden von Microsoft nicht mehr unterstützt und werden über kurz oder lang auslaufen. So gibt es dann mit Gambas unter Linux eine echte objektorientierte Open Source Basic Programmierumgebung, wie es unter Windows die objektorientierte Programmierumgebung VB.Net gibt.

## 6.19 Unterschiede zwischen Visual Basic und Gambas

Siehe dazu folgende Seite: <http://www.gambasdoc.org/help/doc/diffvb?de>

Die Programmierung von Funktionen in Gambas unterscheidet sich etwas von der Programmierung in VB. In Gambas kann man den **Funktionsnamen in der Funktion** nicht aufrufen, in VB ist das möglich.

Der Befehl **Select Case of** in Gambas unterscheidet sich etwas von der Nutzung in VB.

Die **Timerfunktion** gibt die Zeit in Sekunden aus , die seit Programmbeginn verstrichen sind. In Visual Basic ist das anders. Hier wird mit demselben Programm die Zeit ausgegeben, die seit 0 Uhr verstrichen ist.

In VB wird der Programmcode des Formobjektes und der Formdefinition in ein und derselben Datei abgelegt. Gambas trennt beides: Es gibt eine .form und eine .class Datei.

## 6.20 Übersicht Dateierendungen im Vergleich:

Art der Datei:	VB	Gambas
Projekt Definition	.vbp	.project
Module	.bas	.module
Dateierendung Klasse	.cls	.class
Form Definitions Datei	.frm	.form
Binärdateien	.frx	beliebig

## 6.21 VB Form Import

Es gibt ein ImportVBProject von Rob Kudla, welches die VB Forms in Gambas Klassen umwandelt. Die automatische Umwandlung von VB Code in Gambas Code ist schwieriger. Dazu ist einige Handarbeit notwendig.

## 6.22 Globale Variablen in Gambas

In VB war es möglich zb aus der Form2 heraus eine Textbox in Form1 direkt anzusprechen. Der Befehl dazu lautet: `Form1!Textbox1.Text = "zb"`. Dies funktioniert in Gambas nicht. In Gambas gibt es keine projektübergreifenden globalen Variablen. Man kann sich behelfen, wenn man eine Klasse erzeugt, die man Global nennt. Die globalen Variablen deklariert man als statische public Variablen in dieser Klasse. Dann kann man auf sie mit `Global.variablename` im ganzen Projekt zugreifen.

*Minsini: It's still poor programming practice but at least they'll be identified as global variables whenever you use them ;)*

Siehe [Gambas:\\_Form#Globale\\_Variablen\\_in\\_Gambas](#)

## 6.23 IsNumber

Frage zu Isnumber

```
TextArea1.Text = "349"  
PRINT IsNumber(TextArea1.Text)
```

ergibt FALSE. Warum ?

IsNumber() überprüft nur den Typ eines Wertes beginnend von boolean bis float. Es verhält sich anders als IsNumber() in VB.

Wenn man wissen will ob ein Textstring eine Zahl ist, dann muß man Val()/CInt()/CFloat() nutzen und überprüfen ob man eine Fehlermeldung bekommt.

## 6.24 Unload Befehl unter Gambas nicht verfügbar

Unter VB gibt es die Methode *Unload*. Diese ist unter Gambas nicht verfügbar. Als Alternative kann man die Methode Hide oder Delete verwenden. Bei einer Form gibt es noch die Close Methode.

Beispiel: Sie brauchen eine Form und einen Commandbutton für dieses Beispiel.

```
PUBLIC SUB Button1_Click()  
    Button1.delete  
END
```

Oder

```
PUBLIC SUB Button1_Click()  
    ME.delete  
    'Form1.close  
END
```

### Schwierigkeiten mit Gambas

## 6.25 Die Inflation der Befehle

Wer den Übergang von Basic am C64, zu GWBasic und QBasic unter Dos erlebt hat, dem fiel es relativ leicht auf Visual Basic umzusteigen. Die Version VB 3.0 war ( und ist) sehr stabil. Sie ist kompakt und hat ein sehr gut funktionierendes Hilfesystem mit Beispielprogrammen für jeden einzelnen Befehl. Die Zahl der Basicbefehle blieb in diesen Sprachen ziemlich konstant, nur die Art der Programmierung änderte sich und wurde komfortabler und übersichtlicher.

Schon VB 6 und um so mehr VB.Net sind für Anfänger weitaus schwieriger zu erlernen. Woran liegt das und was kann man als Gambaseinsteiger für Schlussfolgerungen daraus ziehen?

Meines Erachtens liegt es an der Inflation der vielen Möglichkeiten und Befehle, die einen Anfänger überfordern.

Ein Anfänger beim Programmieren braucht schnelle Erfolgserlebnisse. Mit relativ kleinen, leicht verstehbaren Programmbeispielen sollte er schon ganz gut brauchbare Ergebnisse erreichen. Dabei muß sein Wortschatz an Programmbefehlen erst wachsen. Er muß lernen, mit den neuen Befehlen umzugehen und sie sicher zu gebrauchen. Was sind grundlegende, sehr häufig gebrauchte Befehle und auf welche Befehle kann man am Anfang verzichten? Hier muß die Programmiersprache Gambas noch verbessert werden. Es sollte eine Statistik über die Häufigkeit der Befehle geben. Hilfreich wäre auch eine Einschätzung von erfahrenen Programmierern, welche Befehle von Ihnen häufig genutzt werden und welche anfangs verzichtbar sind.

Meines Erachtens sind folgende Befehle sehr wichtig:

```
DIM , AS , Print , = , & , FOR TO, NEXT, ME.Close. IF THEN, TRUE, FALSE
```

Fangen Sie mit diesen an und versuchen Sie sie gut zu verstehen.

## 6.26 Funktionen und Prozeduren

Dem Anfänger bieten Prozeduren und Funktionen erhebliche Probleme, Dabei sind Prozeduren noch einfach zu verstehen, denn es werden keine Werte übergeben. **Jedes Unterprogramm ist eine Prozedur.** Der Rahmen eines Unterprogrammes wird in Gambas wie auch in VB automatisch erstellt wenn man in der Formularansicht auf das grafische Objekt klickt. In diesen Rahmen kann man dann seinen Code für das Unterprogramm eingeben.

Funktionen sind etwas schwieriger zu verstehen. Ist man den alten Spagetticode von Uraltbasic mit Gotos und Rücksprüngen gewohnt, dann fällt es einem anfangs schwer eine Aufgabe an ein Unterprogramm zu übergeben und dazu dessen Namen zu nutzen. Im Unterschied zu einer Prozedur wird in einer Funktion aber noch mindestens ein Wert mitübergeben. Man muß also wissen wie die Prozedur heißt, wie man sie aufruft, wie der Wert heißt den man beim Aufruf mitübergibt und wie der Wert heißt, der aus der Funktion wieder herauskommt. Als Anfänger ist man versucht ohne Funktionsaufrufe auszukommen und das Problem ohne Funktion zu codieren. Das erscheint am Anfang verständlicher. Es ist aber in jedem längeren Programm unökonomisch. Wohl oder übel muß man sich an Funktionen gewöhnen, denn sie erleichtern das Programmierleben ungemein.

Siehe auch <http://www.madeasy.de/7/prgmini.htm#funk>

## 6.27 Die Objektorientierte Programmierung ( OOP )

Der Nutzen der objektorientierten Programmierung bleibt dem Anfänger leider oft zu lange verborgen. OOP macht das Programmieren anfangs unnötig kompliziert, bis man endlich begreift, dass die OOP vor allem dazu dient nicht immer wieder dasselbe neu programmieren zu müssen, wenn es schon einmal erarbeitet wurde.

Die Grafische Bedienoberfläche erleichtert es einem am Anfang auf die OOP Methode zu verzichten, obwohl eigentlich die vielen grafischen Werkzeuge die idealen Beispiele für Objekte und Klassen sind.

Denken Sie daran: Jede Programmiersprache hat ein paar grundlegende (**”primitive”**) **Datentypen** , wie z.B. Zahlen, Buchstaben und Logische Datentypen.

z.B.

- boolean
- byte
- short
- int
- long
- float
- double
- char
- string

Diese einfachen Datentypen können nun miteinander kombiniert werden. So entstehen **zusammengesetzte Datentypen** wie Arrays, Strukturen oder auch Klassen.

Auch für diese komplexen Datentypen können nunmehr Variablen deklariert werden, und im Fall von Klassen heißen die Variablen Objekte.

**Merke:**

- **Klassen sind zusammengesetzte Datentypen.**
- **Objekte sind die Daten, die man in diese zusammengesetzten Datentypen hineinschreibt.**

**Beispiel:**

- Der Befehlsbutton in der Werkzeugkiste ist eine Klasse.
- Setzt man einen Commandbutton auf eine Form dann ist dieser spezielle Befehlknopf zusammen mit seinen Eigenschaften ein Objekt.

## 6.28 Public, Private, Static

Gegenüber den alten Basicversionen erscheinen in Gambas ( wie auch in VB6 oder VB.Net) die Befehle Public, Private, Static etc.. Es sind sogenannte Zugriffsbeschränkungen bei Zugriffen auf Klassen. Der Sinn dieser Zugriffsmodifizierer bleibt einem am Anfang verborgen.

## 6.29 Es gibt (keine) globale Variablen wie unter VB

Unter VB kann man sich einfach eine globale Variable zb im einem Modul global.bas definieren und dann zb Text von einer Form zur anderen kopieren, wenn man diese globale Variable nutzt. Das geht unter Gambas nicht so einfach. Benoit Minisini ist da ziemlich streng und will uns zu einem sauberen Programmierstil erziehen.

*Zitat: There is no such thing as a project-wide global variable in Gambas. As a workaround, consider making a class called Global and declaring your global variables as static public variables in that class, and then referring to them as Global.variablename in your project. It's still poor programming practice but at least they'll be identified as global variables whenever you use them ;)*

Benoit Minisini mag sie nicht. Aber sie sind möglich und m.E. auch ganz praktisch. Siehe [Gambas:\\_Form#Globale\\_Variablen\\_in\\_Gambas](#)

## 6.30 Wünschenswert wäre ein interaktives Lernprogramm

Vielleicht ist es auch wünschenswert , für die einzelnen Befehle kleine Lektionen zu entwickeln, die zunächst den Befehl erklären und dann in einem Übungsteil,



die gelernten Erkenntnisse abfragen. Diese Lektionen kann man dann zur Wiederholung und bei Bedarf immer wieder einmal durcharbeiten. Legendär und didaktisch hervorragend war das Lernprogramm, welches bei den ersten Dosversionen der Textverarbeitung Word mitgeliefert wurde. So ein schönes interaktives Lernprogramm in Gambas, am besten selbst in Gambas programmiert und Internet basiert wäre ideal. Die Lektionen sollten, dabei nicht länger als 3 - 5 Minuten zur Bearbeitung dauern.

Ein Beispiel für ein schönes interaktives Lernprogramm der Sprache Java steht hier: <http://www.gailer-net.de/tutorials/java/java-toc.html>

Da Gambas auch mit Java verwandt ist, kann man es durchaus auch nutzen um Gambas zu verstehen.

## **6.31 Gambas light**

Vielleicht sollte es für Anfänger eine Lightversion von Gambas geben. Mit einigen wenigen Basisbefehlen und Basiswerkzeugen.

## **6.32 Nachteile von Gambas**

Gambas ist noch nicht so weit verbreitet.

Cplusplus und Java sind die meist genutzten erwachsenen Programmiersprachen. PHP, Perl und andere Scriptsprachen werden auch viel genutzt. Gambas bietet sich für die große Zahl der Visual Basic Programmierer an, die Open Source programmieren wollen und Linux und KDE nutzen.

Einen weiteren Nachteil sollte man nicht verschweigen. Bis jetzt gibt es keine portierbaren EXE Files. Gambasprogramme laufen zwar als compilierte Programme sehr gut. Aber auf dem Rechner muß eine Gambasumgebung aufgespielt sein. Eine einfache VBRUN.DLL, wie unter Windows, existiert noch nicht. Das wird sich aber sicher bald ändern.



# Kapitel 7

## Autoren

<b>Edits</b>	<b>User</b>
1	<a href="#">Kristjan</a>
3	<a href="#">Wikimurmeltier</a>
114	<a href="#">BK</a>
3	<a href="#">Berni</a>
2	<a href="#">Sundance Raphael</a>
1	<a href="#">Tschäfer</a>
1	<a href="#">Timo Müller</a>
1	<a href="#">Daniel B</a>
10	<a href="#">InselAX</a>
49	<a href="#">Harald Weber</a>
1	<a href="#">FeG</a>
2	<a href="#">Klartext</a>
3	<a href="#">Merkel</a>
2	<a href="#">Jorunn</a>
1	<a href="#">Klaus Eifert</a>
1	<a href="#">Gronau</a>
1	<a href="#">Martin Fuchs</a>
154	<a href="#">Rho</a>
1	<a href="#">Matthias M.</a>
1	<a href="#">Chirak</a>
28	<a href="#">Momomu</a>
8	<a href="#">ThePacker</a>
6	<a href="#">Rudolf73</a>
1	<a href="#">Planktom</a>
3	<a href="#">YellowZeta</a>
1	<a href="#">Tomodachi</a>

1	<a href="#">MichaelFrey</a>
1	<a href="#">Delphimann</a>
7	<a href="#">Dieter</a>
5	<a href="#">Robbo21</a>
1	<a href="#">Blak</a>
6	<a href="#">Rathjen</a>
5	<a href="#">Progman</a>
4	<a href="#">Held03</a>
3	<a href="#">Admiral Snyder</a>
117	<a href="#">Dirk Huenniger</a>

# Kapitel 8

## GNU Free Documentation License

### 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter

or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

### 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also

clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.
- H. Include an unaltered copy of this License.
- Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If the-

re is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option de-

signate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author

or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the

electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

#### 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License

or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

#### 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

#### 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.



# Kapitel 9

## Bildnachweis

In der nachfolgenden Tabelle sind alle Bilder mit ihren Autoren und Lizenzen aufgelistet.

Für die Namen der Lizenzen wurden folgende Abkürzungen verwendet:

- GFDL: Gnu Free Documentation License. Der Text dieser Lizenz ist in einem Kapitel diese Buches vollständig angegeben.
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. Der Text dieser Lizenz kann auf der Webseite <http://creativecommons.org/licenses/by-sa/3.0/> nachgelesen werden.
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. Der Text dieser Lizenz kann auf der Webseite <http://creativecommons.org/licenses/by-sa/2.5/> nachgelesen werden.
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. Der Text der englischen Version dieser Lizenz kann auf der Webseite <http://creativecommons.org/licenses/by-sa/2.0/> nachgelesen werden. Mit dieser Abkürzung sind jedoch auch die Versionen dieser Lizenz für andere Sprachen bezeichnet. Den an diesen Details interessierten Leser verweisen wir auf die Onlineversion dieses Buches.
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. Der Text dieser Lizenz kann auf der Webseite <http://creativecommons.org/licenses/by-sa/1.0/> nachgelesen werden.
- cc-by-2.0: Creative Commons Attribution 2.0 License. Der Text der englischen Version dieser Lizenz kann auf der Webseite <http://creativecommons.org/licenses/by/2.0/> nachgelesen werden. Mit dieser Abkürzung sind jedoch auch die Versionen dieser Lizenz für andere Sprachen bezeichnet. Den an diesen Details interessierten Leser verweisen wir auf die Onlineversion dieses Buches.
- GPL: GNU General Public License Version 2. Der Text dieser Lizenz kann auf der Webseite <http://www.gnu.org/licenses/gpl-2.0.txt> nachgelesen werden.
- PD: This image is in the public domain. Dieses Bild ist gemeinfrei.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.

Bild	Autor	Lizenz
1	Benoît Minisini	GPL
2	Harald Weber	GFDL
3	<a href="#">User:Rho</a>	GFDL
4		PD
5		PD
6		PD
7		PD
8		PD
9		PD
10	Benoît Minisini	GPL
11	Benoît Minisini	GPL
12	Benoît Minisini	GPL
13		PD
14		PD
15	Benoît Minisini	GPL
16	Benoît Minisini	GPL
17	Benoît Minisini	GPL
18	Benoît Minisini	GPL
19	Benoît Minisini	GPL
20	Benoît Minisini	GPL
21	Benoît Minisini	GPL
22	Benoît Minisini	GPL
23	Benoît Minisini	GPL
24	Benoît Minisini	GPL
25	Benoît Minisini	GPL
26	Benoît Minisini	GPL
27	Benoît Minisini	GPL
28	Benoît Minisini	GPL
29	Benoît Minisini	GPL
30		PD
31		PD
32		PD
33	<a href="#">Benutzer:Rho</a>	GFDL
34		PD
35		PD
36		PD
37		PD
38	<a href="#">Benutzer:Rho</a>	GFDL
39		PD
40		PD

---

41	Benutzer:Rho	GFDL
42		PD
43	Benutzer:Rho	GFDL
44	Benutzer:Rho	GFDL
45	Benutzer:Rho	GFDL
46		PD
47		PD
48		PD
49	Benutzer:Rho	GFDL
50	Benutzer:Rho	GFDL
51	Benutzer:Rho	GFDL
52	Benoît Minisini	GFDL
53	Dr.Reiner Hoffmann	GFDL
54	Dr.Reiner Hoffmann	GFDL
55	Dr.Reiner Hoffmann	GFDL
56		PD
57		PD
58		PD
59		PD
60		PD
61		PD
62	Benoît Minisini	GFDL
63	Benoît Minisini	GFDL
64	Dr.Reiner Hoffmann	GFDL
65		PD
66		PD
67	Dr.Reiner Hoffmann	GFDL
68	Benutzer:Rho	GFDL
69		PD
70		PD
71		PD
72		PD
73	Harald Weber	GFDL
74	Benoît Minisini	GPL
75		PD
76		PD
77		PD
78		PD
79		PD
80		PD
81		PD

82		PD
83		PD
84		PD
85	Harald Weber	GFDL
86		PD
87		PD
88		PD
89		PD
90		PD
91		PD
92		PD
93	Harald Weber	GFDL
94	Benoît Minisini	GFDL