WIKIBOOKS

Gambas/Printable version

Gambas

Contents

Preface

Back to Gambas

> “ When I first started to program on a computer I used the Basic interpreter on the C64. Then I moved to MS Dos and GWBasic/Qbasic. The next step was Visual Basic 3. At the same time the Internet came into use. So I put all my mini VB programs on the Internet. This was very helpful for me and others.
>
> After lots of bluescreens the divorce from MS was done in 2002. Linux and KDE were my new favourites and they still are.
>
> But what I always missed was VB for Linux and KDE. My old brain could not get used to C++ or Java. At Christmas time in 2003 I discovered Gambas built by Mr. Minisini and friends. It was a hit from the beginning. So I started converting all my mini VB programs to Gambas. (So called Minisinis)
>
> I think Gambas will be very successful. To promote it, I will put all my mini programs here under GNU as a wikibook.
>
> Everyone is invited to improve my bad English and to improve this Wikibook.
>
> rho in January 2005 de:Benutzer:Rho ”

Introduction

Back to Gambas

What is Gambas ?

Gambas is a programming language for Linux which attempts to mimic the ease of use of Visual Basic while improving on its functionality. Although Gambas is not source code compatible with Visual Basic, it is a BASIC Interpreter with object-oriented extensions. This makes Gambas a good choice for Linux users who want to use their VB knowledge on a GNU basis.

Gambas makes it very easy to build Linux GUI programs (the Gambas IDE is written in Gambas itself.) using the Qt toolkit. There is a Windows version of Gambas which barely runs under the Cygwin environment. Graphical applications do not yet work under the Windows version.

Developed in Paris by Benoît Minisini since 1999, Gambas is released under the GNU General Public Licence, the current version (July 29, 2009) being 2.15.2 Gambas is included in the Linux Professional Distribution from Novell.

'Gambas' is a backronym for **GAMBAS Almost Means BASic** (*Gambas* is also the Spanish word for shrimp.)

Why Gambas ?

With Gambas you can easily:

- Build solutions for any environment using "graphical application" when you start a new project. This mode takes care of which Desktop environment the end user is going to use, and the program will automatically adapt to Gnome, KDE, etc. depending on the **Current running Desktop environment**.
- Build KDE applications with DCOP
- Translate your programs into other languages
- Offer Visual Basic programs under Linux
- Build network solutions
- Quickly design your program GUI with QT or GTK+
- Use databases such as MySQL, PostgreSQL, Firebird, ODBC and SQLite
- Create network applications easily
- Make 3D OpenGL applications
- Make CGI web applications, and so on...

Citations

- "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."
- "If you pass the responsibilty for the pc operating system only to one company , then this is just the same, as if you would sell the abc or the integers to this company."
- "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 http://gambas.sourceforge.net/en/main.html. Gambas is translated into many languages. To run Gambas applications you need the Gambas interpreter component installed on the user machine."
- Slashdot: Gambas 1.0 Release Candidate Available raindog2 writes "After two and a half years of development, Gambas has become the first Visual Basic-style environment for Linux to enter release candidate status. Anyone who has been frustrated by a lack of production-quality free RAD environments should give it a try."
- You can support Gambas by making a small donation with PayPal Link: https://www.paypal.com/xclick/business=gambas%40users.sourceforge.net&no_note=1&tax=0&currency_code=EUR

Installation

Back to Gambas

Download

- See http://gambas.sourceforge.net/en/main.html
- Download
- Download Gambas (latest version)

How to compile, make and install Gambas?

Short version

```
tar xzvf gambas-x.xx.tar.gz
cd gambas-x.xx
./configure # or ./configure—disable-db-component
make
```

```
make install
```

Long version

See also http://gambaswiki.org/wiki/install

First

Back to Gambas

For the first steps see:

- http://gambaswiki.org/wiki/howto/getstarted

Creating your first gambas application

See the Hello World tutorial for information on how to create your first Gambas application.

Helpful

Back to Gambas

Nothing for now.

Comments

Back to Gambas

A good program has comments. They are very helpful to others trying to understand the program. When the program is compiled, all the comments are skipped. You can make comments as long as you want.

The difference between Gambas comments and VB comments is that Gambas has no such command as REM. **Use the apostrophe** instead.

'Apostrophe works. Rem does not.

Comments begin with an apostrophe and continue to the end of the line. Multi-line comments are not allowed.

Example

```
'This is a comment and this is skipped during compilation.
```

You can comment or uncomment when you are testing a program. In the code window you can also mark some code and make a comment out of it with the shortcut **Ctrl + K**. You can uncomment with **Ctrl + U**.

Assignments

Equal or not equal

In gambas the = sign is used differently than it is in mathematics. It is used to assign a value to a variable.

When the equal sign shows up in the code, you should always think of it as:

- *It is assigned*
- *A place in computer memory is reserved for the variable*
- *Fill the memory with this value*

Then you will not run into illogical conclusions.

The following code is correct in Gambas. Try it out. What is the result?

```
a = 5
a = a * 4
```

Translated to natural language this says:

```
Assign 5 to the variable a
Multiply 4 with the old variable a and assign it to the new variable a.
```

Maybe the first person who started assignment in computers came from Arabia. Perhaps this is why we have to read the line from the right to left.

By the way: the result is 20.

How does this work in a little program: You need one command button to get the program going. The result is shown in the terminal window.

```
PUBLIC SUB Button1_Click()
DIM a AS Integer
a = 5
a = a * 4
PRINT "a = ";a
END
```

A math teacher would not like these equations, but in Gambas (or in basic in general) it is correct. Of course, you could make things clearer by using two different variables.

```
PUBLIC SUB Button1_Click()
DIM a AS Integer
DIM b AS integer
a = 5
b = a * 4
PRINT b
END
```

The line

```
DIM a AS Integer
```

is not an assignment. It is a declaration of a variable as a datatype. In this case a is of the datatype *integer*.

Theory of the assignment

The following code shows you in general the method of assignment:

```
Variable = Expression
```

Assigns the value of an expression to one of the following elements. :

- A local variable.
- A function parameter.
- A global (class) variable.
- An array slot.
- An object public variable.
- An object property.

Example

```
iVal = 1972
Name = "Gambas"
hObject.Property = iVal
cCollection[sKey] = Name
```

See Also

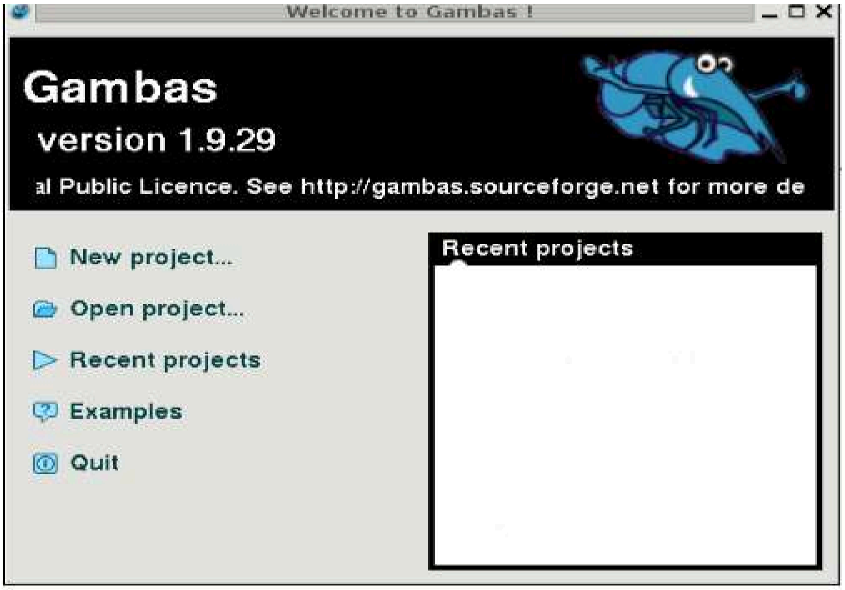
- Variable declarations
- Combination Assignment Operators

Hello World

Every programming book should start out with a "Hello World" program, just to get you started. This book's author sees no reason not to follow this "standard" so we will do just that.

Create a new project

Start gambas You will be greeted with this screen:



After that follow the steps as described below(no screen-shots)
Click "New Project":
Click "Next":
Select "Create a graphical project":
Click "Next":
In the "Select the name of the project" insert Hello_World
In the "Select the title of the project" insert Hello World
Click "Ok" to create the project

Create the Form(Window)

Ok, now we should have gambas started with a fresh, new project. You are now looking at the Gambas GUI, which is where we will do all our Gambas work. When you look at the interface, it seems pretty useless; No windows, No buttons or what have you. Let's change that right away.

On your left side, you'll see a tree view, right click on the Forms folder, and choose "New --> Form"(the --> indicates that you choose New, and follow the submenu to Form).

In the Create form dialog, just click "Ok".

As you noticed, quite a few things happened. A new Form (Window) was created, and some other new things showed up. **NOTE:** In gambas a *Window* is called a *Form*. I will use the term *Form(s)* in the rest of the book.

The grey Form that just got created looks a bit ugly(tiny dots all over the place) but don't worry about that. The dots are there for your aid in placing your objects in the Form.

To your right you should see the "ToolBox". That's where you can choose what objects you want in your form. In our "Hello World" program, we'll need one label(marked as "A" in the toolbox) and one button(marked as "Ok" in the toolbox). Let's start with the label. Click on the label in the toolbox, and move the new form. In the form, hold down the left mouse button, and "draw" the label on there. Once done, you will have a label in your form with the text "Label1" in it. Now you can change this text with the "Hello Word" phrase.

The next step is to put some code in the Ok Button (you must change the text to "OK" of this button as you changed in the Label). After changing the text please double click the button so we can enter the exit code in the event click of this button. Type QUIT in the click event and that's all you've created your Hello World Application

index

Maths

Back to Gambas

The Cross Sum Program

This miniprogram calculates the cross sum of an integer. You need 2 textboxes and 1 commandbutton to get the program going.

Please beware of the signs / and \ . In this program the sign \ is used for a division without a remainder.

The Code

```
PUBLIC SUB Button1_Click()

    DIM z AS Integer
    DIM crosssum AS Integer

    z = Val(TextBox1.Text)

    DO WHILE z <> 0
      crosssum = crosssum + z MOD 10
      z= z \ 10
    LOOP

    TextBox2.Text = Str$(crosssum)
END
```

Addy The Sum Program

This miniprogram calculates the sum of a row of integers. You need 1 textarea, 1 textbox and 1 commandbutton to get the program going. You can also add negative integers. Very helpful is the Split command and the String array String[]

The Code

```
PUBLIC SUB Button1_Click()
  DIM text AS String
  DIM summe AS Float
  DIM Elt AS String[]
  DIM Sb AS String

  text = textarea1.Text
  Elt = Split(text,Chr(10))

  FOR EACH Sb IN Elt
    summe = summe + Val(sb)
  NEXT

  textbox1.Text = summe
END
```

The Average Program

This miniprogram calculates the average of a row of integers. You need 1.textarea, 2 textboxes and 1 commandbutton to get the program going.

The Code

```
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 = textarea1.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
```

The Median

The Median of a list of values lies between two equal parts of the list. 50% of the values ar greater or equal to the median. 50% are smaller or equal to the median. In contrary to the average the median is not influenced that much by extreme values.

If you want to program the median, you have to

- 1.sort your list of values
- 2.check if the list has an even or an odd number of values.
- 3.choose the right formula for the median.
  - for an odd number of values

```
xmedian = x[(n+1)/2]
```

- - for an even number of values

```
xmedian = 1/2*(x[n/2] + x[n/2+1])
```

example 1:

sorted list of values:

```
11
12
13
14
15
```

The number of values is odd. n = 5

```
(n+1)/2 =  6/2 = 3
```

xmedian = x[3] = 13

example 2.

sorted list of values:

```
11
12
13
14
15
16
```

The number of values is even , n = 6

```
n/2 =  6/2 = 3
x[3] = 13
```

```
x[4] = 14
xmedian = 1/2*(13+14) = 13,5
```

You can take the list from above and copy it into the Gambas median program. Use the clipboard of your computer for that. Ctrl+C = Copy , Ctrl + V = Paste

example program

You need 2 textareas , 3 commandbuttons and 1 textbox, to get the program going:

```
' Gambas CLASS file
 PUBLIC liste AS String[]
 PUBLIC SUB Form_Open()
   ME.Text = "Computate the Median "
 END
 PUBLIC SUB Button1_Click()
 'sort
 Dim c AS Integer
 Dim j AS Integer
 Dim n AS Integer
 Dim y AS Variant
 Dim liste AS String[]
 Dim element AS String
 Dim txt AS String
 Dim text AS String
 text = Textarea1.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)
 'the last CR has to be killed chr(10)
 textarea2.Text = txt
 END
 PUBLIC SUB Button2_Click()
 'computate median, but sort first !!
  Dim text AS String
  Dim median AS Float
  Dim liste AS String[]
  Dim posten AS String
  text = Textarea2.Text
  liste = Split(text,Chr(10))
  'count the length of your list and choose the right formula
     IF liste.Length MOD 2 THEN
   'PRINT liste.Length MOD 2 & " odd"
   'PRINT (liste.length + 1)/2
   'PRINT liste[(liste.length + 1)/2 - 1]
   median = Val(liste[(liste.length + 1)/2 - 1])
   'the array starts with the element 0 not with the element 1 !
   ELSE
   'PRINT liste.Length MOD 2 & " even"
   median = (Val(liste[liste.length/2 - 1]) + Val(liste[liste.length/2]))/2
   ENDIF
  textbox1.Text = Str(median)
  END
  PUBLIC SUB Button3_Click()
  'list of values as an example
  textarea1.Text = "114,3"
  textarea1.Text = textarea1.Text & Chr(10) & "135,7"
  textarea1.Text = textarea1.Text & Chr(10) & "104,8"
  textarea1.Text = textarea1.Text & Chr(10) & "118,5"
  textarea1.Text = textarea1.Text & Chr(10) & "125,7"
  textarea1.Text = textarea1.Text & Chr(10) & "121,4"
  textarea1.Text = textarea1.Text & Chr(10) & "122,4"
  textarea1.Text = textarea1.Text & Chr(10) & "96,8"
  textarea1.Text = textarea1.Text & Chr(10) & "118,9"
  textarea1.Text = textarea1.Text & Chr(10) & "120"
  textarea1.Text = textarea1.Text & Chr(10) & "112,2"
  textarea1.Text = textarea1.Text & Chr(10) & "127,9"
  textarea1.Text = textarea1.Text & Chr(10) & "122,8"
  textarea1.Text = textarea1.Text & Chr(10) & "128,9"
  textarea1.Text = textarea1.Text & Chr(10) & "120,3"
```

```
'median = 120,3
END
```

Graphics

Back to Gambas

If you want to produce some graphics in gambas you should get used to the drawingarea control.

Draw a Line

This little program will draw a line. You have to start a new graphics project. You take a form as your start form. You need a drawingarea and a commandbutton on the form to get it going.

```
PUBLIC SUB Button1_Click()
Draw.Begin(DrawingArea1)
Draw.Line(1, 130, 500, 400)
Draw.End
END
```

When you want to draw some more lines, then try this example:

```
PUBLIC SUB Form_Open()
DIM B AS Integer
Draw.Begin(DrawingArea1)
FOR B = 1 TO 200 STEP 10
 Draw.Line(1, B, 500, B)
NEXT
Draw.End
END
```

Some other lines are drawn here:

```
PUBLIC SUB Button1_Click()
DIM B AS Integer
Draw.Begin(DrawingArea1)
' Draws a line horizontally across the centre of the form
Draw.Line (0, ME.Height / 2, ME.Width, ME.Height / 2)
' Draws a line vertically down the centre of the form
Draw.Line (ME.Width / 2, 0,ME.Width / 2, ME.Height)
' Draws a line from the top left, to the bottom right
Draw.Line (0, 0,ME.Width, ME.Height)
' Draws a line from the top right, to the bottom left
Draw.Line (ME.Width, 0,0, ME.Height)
Draw.End
END
```

When you want to manipulate the **width of your line** , then try this example:

```
PUBLIC SUB Form_Open()
DIM B AS Integer
Draw.Begin(DrawingArea1)
Draw.Line(10,100, 20, 100)
     FOR B = 1 TO 100 STEP 10
      Draw.LineWidth=B
      Draw.Line(10+B,100, 20+B, 100)
     NEXT
Draw.End
END
```

If you want to change **the color of your line** to white, then try this example:

```
Draw.Begin(DrawingArea1)
Draw.ForeColor = &HFFFFFF
Draw.Line(1, 130, 500, 400)
Draw.End
```

The next small example will draw a box and **fill it** with white color.

```
PUBLIC SUB Button1_Click()
Draw.Begin(DrawingArea1)
 Draw.FillColor = &HFFFFFF
 draw.FillStyle = 1
 'draw.ForeColor = &HFFFFFF the outline will be white also
 Draw.Rect (100, 100,200,200)
Draw.End
END
```

The Spiral Program

This program shows you a nice spiral. There is no scale command in Gambas (at least, not yet). Therefore the x and y coordinates are transformed in the program. The coordinate system ranges from xmin = - 2 to xmax = 2 and ymin = - 2 to ymax = 2.

You need a Drawingarea and a Commandbutton to get the example going.

How does this look ? See [1] (http://www.madeasy.de/7/prgspir.htm)

The code:

```
PUBLIC SUB Button1_Click()
DIM dymax AS Integer
DIM dymin AS Integer
DIM ymax AS Integer
DIM ymin AS Integer
DIM y AS Float
DIM dy AS Float
DIM dyi AS Integer
DIM dxmax AS Integer
DIM dxmin AS Integer
DIM xmax AS Integer
DIM xmin AS Integer
DIM x AS Float
DIM dx AS Float
DIM dxi AS Integer
DIM k AS Float
DIM a AS Float
DIM w AS Float
dymax = DrawingArea1.Height
dymin = 0
ymax = 2
ymin = -2
dxmax = DrawingArea1.Width
dxmin = 0
xmax = 2
xmin = -2
'x-axis is drawn
FOR x = xmin TO xmax STEP 0.005
y = 0
dy = CFloat(y - ymin) / (ymax - ymin ) * (dymax - dymin) + dymin
dx = CFloat(x - xmin) / (xmax - xmin ) * (dxmax - dxmin) + dxmin
dyi = Fix(dy)
dxi = Fix(dx)
Draw.Begin(DrawingArea1)
Draw.Point(dxi,DrawingArea1.Height- dyi)
Draw.End
NEXT
'y - Axis is drawn
FOR y = ymin TO ymax STEP 0.005
x = 0
dy = CFloat(y - ymin) / (ymax - ymin ) * (dymax - dymin) + dymin
dx = CFloat(x - xmin) / (xmax - xmin ) * (dxmax - dxmin) + dxmin
dyi = Fix(dy)
dxi = Fix(dx)
Draw.Begin(DrawingArea1)
Draw.Point(dxi,DrawingArea1.Height- dyi)
Draw.End
NEXT
'Here the spiral starts
FOR k = -100 TO 150 STEP 0.5
a = 0.97
'Distance from 0,0 to the point
w = 0.15
'Angle under whiche the point is seen from the origin
a = a ^k
w = w * k
x = a * Cos(w)
'x coordinate of each point
y = a * Sin(w)
'y coordinate of each point
dy = CFloat(y - ymin) / (ymax - ymin ) * (dymax - dymin) + dymin
dx = CFloat(x - xmin) / (xmax - xmin ) * (dxmax - dxmin) + dxmin
dyi = Fix(dy)
dxi = Fix(dx)
Draw.Begin(DrawingArea1)
Draw.Point(dxi,DrawingArea1.Height- dyi)
Draw.End
NEXT
END
```

Colours

Back to Gambas

Basic colour constants that may be used to specify colour

Hexadecimal Value	Colour
&H0	Black
&HFF	Red
&HFF00	Green
&HFFFF	Yellow
&HFF0000	Blue
&HFF00FF	Magenta
&HFFFF00	Cyan
&HFFFFFF	White

The RGB Colours Program

This program shows you the RGB colours. You can change the values of each colour from 0 - 255 and you will see the special mixed colour, when you hit the commandbutton.

Out of 3 decimal numbers, one hexadecimal number is build as a string using the following code:

```
sHEX = Hex$(r,2) & Hex$(g,2) & Hex$(b,2)
sHEX = "&H" & sHEX & "&"
```

To get the program going , you will need 1 commandbutton, 1 drawingarea, 3 textboxes and 3 textlabels.

What does it look like? See it here (in German) http://www.madeasy.de/7/prgfarbergb.htm

The Code:

```
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
```

The Sunset Program

When you need a background for a program, you can use this little color sunset program.

You will need a drawing area on your form to get the program going. You can choose different colours when you use the following scheme:

Blue RGB(0, 0, i) Red RGB(i, 0, 0) Green RGB(0, i, 0) Yellow RGB(i, i, 0) Violett RGB(i, 0, i) Türkis RGB(0, i, i) Gray RGB(i, i, i)

What does it look like ?

The Code

```
PUBLIC SUB Form_Open()
DIM r AS Integer
DIM g AS Integer
DIM b AS Integer
DIM h AS Integer
DIM he AS Integer
DIM sHex AS String
r = 0
g = 0
FOR b = 0 TO 255
sHEX = Hex$(r,2) & Hex$(g,2) & Hex$(b,2)
sHEX = "&H" & sHEX & "&"
Draw.begin(DrawingArea1)
Draw.Forecolor= Val(sHEX)
Draw.LineWidth=5
he = DrawingArea1.Height
h = he * b \ 255
Draw.Line(0, h, DrawingArea1.Width, h)
Draw.End
NEXT
END
```

You can make the program better, when you add the following code and change the border of the form to Resizeable.

```
PUBLIC SUB Form_Resize()
DrawingArea1.Move(0, 0, ME.ClientWidth, ME.ClientHeight)
END
```

The Spectrum program

The natural colour spectrum is shown in the following program using a slidebar.

You need a drawing area and a slidebar as controls.

```
' Gambas class file
PUBLIC SUB Form_Open()
drawingarea1.BackColor = &HC2020C&
scrollbar1.minvalue = 1
scrollbar1.maxvalue = 1120
END
PUBLIC SUB ScrollBar1_Change()
drawingarea1.BackColor = Val(spektrum(scrollbar1.value))
END
PUBLIC FUNCTION spektrum(ss AS Integer) AS String
x AS Integer
sHEX AS String
s AS Integer
r AS Integer
g AS Integer
b AS Integer
s = 0
FOR x = 1  TO 255
'red to yellow
r = 255
b = 0
sHEX = Hex$(r,2) & Hex$(x,2) & Hex$(b,2)
sHEX = "&H" & sHEX & "&"
s = s + 1
'PRINT s
IF s = ss THEN RETURN sHEX
NEXT
's = 255
FOR x = 255  TO 1 STEP -1
'from yellow to green
g = 255
b = 0
sHEX = Hex$(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
'from green to 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
'from cyan to blue
r = 0
b = 255
sHEX = Hex$(r,2) & Hex$(x,2) & Hex$(b,2)
sHEX = "&H" & sHEX & "&"
s = s + 1
'PRINT s
IF s = ss THEN RETURN sHEX
NEXT
's = 1020
b = 255
FOR x = 1  TO 100
'from blue to violett
g = 0
sHEX = Hex$(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
```

Maybe you can change the program, so that the colour value is shown in a textbox. 1. Get a textbox out of the toolbox. 2. Add the following code

```
textbox1.text = spektrum(scrollbar1.value)
```

in the part PUBLIC SUB ScrollBar1_Change()

Mouse

The Mouse Coordinate Program

This program will show the coordinates of the mousepointer continuously in two textboxes, when the left mousebutton is pressed. You need a form with a drawingarea and two textboxes to get the program going. See how it looks like here: http://www.madeasy.de/7/prgmausxy1.htm

The upper left corner of the drawing area has the coordinates (0,0). The right lower corner has the coordinates (drawingArea1.width, drawingarea1.height)

The Code

```
PUBLIC SUB DrawingArea1_MouseMove()
Textbox1.text = Mouse.X
Textbox2.text = Mouse.Y
END
```

You can simplify the program, when you work without the drawingarea. The code should look like this.

```
PUBLIC SUB Form1_MouseMove()
Textbox1.text = Mouse.X
Textbox2.text = Mouse.Y
END
```

Try it out !

We can even get rid of the textboxes using the print command instead. Then the coordinates appear in the terminal window.

The code should look like this.

```
PUBLIC SUB Form1_MouseMove()
PRINT mouse.X
PRINT Mouse.Y
END
```

Try it out !

Be aware that the DrawingArea is the only control in Gambas (at current version 2.n) that tracks mouse movement. Other controls will only provide Mouse coordinates on the Mouse down event. Additionally they have ENTER and LEAVE functions which provide the 'Mouse Over' functionality.

Output

Back to Gambas

Simple output

The interaction between the user and the computer consists of the input and output of data. You wouldn't know what the computer is doing without seeing or hearing anything. That's the importance of output.

Printing

Note: The word "printing" here means using the Print statement, it's not about using the printer or printing files. Printing is a fairly simply part of gambas, but also essential. Most often printing is used to output information to the user, and can also prove a valuable troubleshooting tool. Whenever printing, you need an object, followed by of course, something to print. Printing may be used with various objects, however, the most common in the terminal window.

Example: You need a command button on your form:

```
PUBLIC SUB Button1_Click()
    Print "Hello world!!!"
END
```

This example prints a message in the terminal window, not on the current form like in VB

Output in Labels, Textlabels, Textboxes oder Textareas

You can also aim your output to different kind of text controls.

Example: You need a command button and a textbox on your form:

```
PUBLIC SUB Button1_Click()
    Textbox1.text = "Hello world!!!"
END
```

Message boxes

One of the easiest way of an output is the message box. This is what the code of a normal message box should look like.

```
 Message.Info("Hallo")
```

Example:

```
PUBLIC SUB Button1_Click()
Message.Info("Hallo, this is your message", "OK")
END
```

Spacing

There is various way to alter how text is spaced when printing. The most common is the comma. A comma will go to the next print zone. Print zones are 15 characters long. You can think of it like pressing the tab key when typing something out. Remember that print zones are fixed, so if you've typed 1 letter, and then used a comma, then it will be a big space. If you type 13 characters and use a comma, it will not be a large space. For example:

```
      Private Sub Form_Click()
             Me.Print "Hello", "Next Zone"
      End Sub
```

Several new things are introduced in this example. The subroutine(which is a function that does not return a value; you'll learn more about them later) Form_Click is called when the user clicks on the current Form(Form1). 'Me' is the same as the current form (Form1). Don't be afraid to experiment. No matter what you do in VB, its always reversible. Now, the comma isn't all that versatile. Another feature is tab. Tab will move so many spaces from the BEGINNING of the line. Followed by tab in parentheses is the amount of characters spaces. For example:

```
            Form1.Print "Hello"; Tab(10); "Yay"
```

This will NOT print "yay" 10 spaces after the O of "Hello". Rather it will print 10 spaces from the beginning of the line. You may use as many tabs as you want in the same print command. Although tab is useful, sometimes it is better to spaces things in relation to what has already been printed. This is where the space function comes in. The syntax of space is identical to that of tab. Space will move the next printed text so many spaces over from its CURRENT location. For example:

```
     Pic.print "Hello"; Space(10); "Yay"
```

This will print the first Y of "Yay" 10 spaces to the right of the O in "Hello". It is important to note, if you write:

```
    Pic.Print "Hello"
    Pic.Print "Hello"
```

They will appear on separate lines as:

```
    Hello
    Hello
```

This can be easily dealt with in the need of having separate print statements print on the same line. You merely have to change the code to:

```
   Pic.Print "Hello";
   Pic.Print "Hello"
```

This will appear as:

```
   HelloHello
```

If you want to make a blank line in between the two hellos, then you may simply have a blank print statement WITHOUT a semicolon. For example:

```
   Pic.Print "Hello"
   Pic.Print
   Pic.Print "Hello"
```

This will print as:

```
   Hello

   Hello
```

It is important to remember that even if the first print has a semicolon at the end, often referred to as a trailing semicolon, the empty print will only reverse it, and print the second Hello on the next line, and no blank line will appear.

Text

Back to Gambas

The Key Code Program

With this simple miniprogram you can check out the keycode of a button you have pressed. You only need a Form to get the program going. Once you started the program and you use a button on your keyboard, the keycode is printed in the terminal window.

The Code

```
PUBLIC SUB Form_KeyRelease()
PRINT key.code
END
```

The Key Release Program

With this miniprogram you can check the arrow keys. When they are used and released a new information is shown.

You need a textbox to get the program going.

Once you started the program and you use an arrow key in the textbox, the content of the textbox will change.

The Code

```
PUBLIC SUB TextBox1_KeyRelease()
SELECT Key.code
 CASE Key.left
  Textbox1.Text ="Left"
 CASE Key.right
  Textbox1.Text ="Right"
 CASE Key.up
  Textbox1.Text ="Up"
 CASE Key.down
  Textbox1.Text ="Down"
END SELECT
END
```

Input Restrictions

If you want a textbox to accept only digits you should use the command **STOP EVENT**.

Example

You need a textbox on your form to get it going.

```
PUBLIC SUB MyTextBox_KeyPress()
  IF Instr("0123456789", Key.Text) = 0 THEN
   STOP EVENT
 ENDIF
END SUB
```

Example2

You can reach nearly the same with the following code:

```
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
PUBLIC SUB Form_Open()
 ME.Text = "Only digits accepted !"
END
```

Clipboard

Back to Gambas

The Clipboard Program

This program shows you the use of the clipboard in Gambas. You need 2 commandbuttons and 2 textareas to get the program going.

The Code:

```
' Gambas class file
PUBLIC SUB Button1_Click()
Clipboard.Copy (Textarea1.Text)
'Copy Text into the Clipboard
END
PUBLIC SUB Button2_Click()
Textarea2.Text  = Clipboard.Paste ()
'get the text out of the clipboard
END
```

Time

Back to Gambas

Usar la Propiedad Time

Time returns the current time. Example:

You need 1 button:

```
Private Sub Button1_click()
   message(time)
end
```

Te daria algo asi: 22:13:00.998 (En ubuntu me da asi :P)

Por ejemplo si queremos hacer que a determinada hora nos avise, seria algo asi:

Necesitas 2 Textbox, 1 Boton, 1 Timer y un modulo..

En textbox1 escribimos la Hora que queremos que nos avise. En textbox2 escribimos el minuto que queremos que nos avise.

Within the module write:

```
PUBLIC Hora as string[]
PUBLIC Final as string[]
```

Within form Fmain, Button1 write:

```
Private Sub Button1_click()
dim tiempo as String
tiempo=Time
  Module1.Hora=Split(tiempo,":")
    Timer1.enabled = true

end
```

```
Private Sub Timer1_()
dim tiempo as string
Timer1.delay = 500
  tiempo=Time
    Module1.Final=split(tiempo,":")
    if Module1.hora[0] =  Module1.Final[0] and Module1.hora[1] = Module1.Final[1] then
            message.info("Ya es la hora que ingresaste antes")
            Timer1.enabled = false
    end if
end
```

Wait Command

The Syntax is

```
WAIT [ Delay ]
```

This command calls the event loop. If Delay is specified, it does not return until Delay seconds elapse. Delay is a floating point number. So, if you want to wait 100 ms, just do:

```
 WAIT 0.1
```

During the wait, no keyboard or mouse events are processed. Only drawing, timer and file descriptors events, like Process_Write are still running.

Example

You need a commandbutton and a textbox to get it going.

```
PUBLIC SUB Button1_Click()
 WAIT 10
 'waits 10 seconds
 WAIT 0.1
 'waits 100 milliseconds
 textbox1.Text = ""
END
```

Sound

Back to Gambas

Sound

There is no native sound class. You have to activate a component.

In the project properties menu, check a component like `gb.sdl.sound` or `gb.sdl2.audio` for using it.

After that, use the static class Music! Example:

```
PUBLIC SUB Button1_Click()
 Music.Load("/path/to/your/file.mp3")
 Music.Play()
END
```

Toolbox

Back to Gambas

List

The gambas toolbox contains all the controls you can drop on a Gambas form. http://commons.wikimedia.org/wiki/File:Gambas_toolbox3.png

In the beginning you should get used to some of the tools.

- Try the textbox and textarea.
- Try the commandbutton.
- Try the drawingArea
- Try the timer.

They are important and the other ones are a lot easier, when you have learned to use these few in the beginning.

- Label
- Image
- TextLabel
- ProgressBar
- Button
- CheckBox
- RadioButton
- ToggleButton
- TextBox
- ComboBox
- TextArea
- ListBox
- ListView
- TreeView
- IconView
- GridView
- ColumnView
- Frame
- Panel
- TabStrip
- ScrollView
- DrawingArea
- Timer
- GambasEditor
- LCDNumber
- Dial
- SpinBox
- ScrollBar
- Slider
- TableView
- Splitter
- Workspace

The first one in the row, the Selection Tool, is rather useless. It isn't actually a control. The selection tool simply gives you the plain pointer back, with which you can select and operate on forms and their controls.

Properties

Methods

Events

Grid

Back to Gambas

Introduction

A grid has rows and columns. They have width and height. You can fill a grid with text, numbers or even a picture.

Example 1

You need a new form to get the program going. On this form you place a gridview , which you can find in your toolbox. There should be a picture x.png in your directory. Otherwise it wont be shown. There wont be an error message, when it is not found.

```
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
```

Create a Grid without the Toolbox

The grid ist creatable. How it is done shows the little example.

```
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
```

You just need a empty form to get the program going.

Properties of the grid

```
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
```

Methods

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

Events

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

Fill the grid with some values

You have a list of values and you want to have them in a grid. The example shows you a solution. Still not very nice and with some German words in it.

You need the following controls:

- 1 Textarea
- 1 Gridview
- 2 Commandbuttons

The code

```
PUBLIC SUB Button1_Click()
  textarea1.Text = "114"
  textarea1.Text = textarea1.Text & Chr(10) & "135"
  textarea1.Text = textarea1.Text & Chr(10) & "104"
  textarea1.Text = textarea1.Text & Chr(10) & "118"
  textarea1.Text = textarea1.Text & Chr(10) & "125"
  textarea1.Text = textarea1.Text & Chr(10) & "121"
  textarea1.Text = textarea1.Text & Chr(10) & "122"
  textarea1.Text = textarea1.Text & Chr(10) & "96"
  textarea1.Text = textarea1.Text & Chr(10) & "118"
  textarea1.Text = textarea1.Text & Chr(10) & "120"
  textarea1.Text = textarea1.Text & Chr(10) & "112"
  textarea1.Text = textarea1.Text & Chr(10) & "127"
  textarea1.Text = textarea1.Text & Chr(10) & "122"
  textarea1.Text = textarea1.Text & Chr(10) & "128"
  textarea1.Text = textarea1.Text & Chr(10) & "120"
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 = textarea1.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
```

When you push button1, then the textarea is filled with values. When you push button2, they will be placed in the grid.

You can improve this program if you want. It looks a little newbee like.

Dialogue

back to Gambas

Standard Dialogs

They make your coding easier.

In the following example you will use standard dialogues choosing Standarddialoge für die Wahl

- the font
- the colour
- the file name
- the file path

You need 1 textarea and 3 commandbuttons, to get the program going.

```
PUBLIC SUB Button1_Click()
 IF Dialog.SelectFont() THEN RETURN
 Textarea1.Font = Dialog.Font
END
PUBLIC SUB Button2_Click()
 IF Dialog.SelectColor() THEN RETURN
 Textarea1.Background = Dialog.Color
END
PUBLIC SUB Button3_Click()
Dialog.SaveFile()
File.Save(Dialog.Path, TextArea1.Text)
CATCH
IF ERROR THEN RETURN
END
```

Theory

Properties

```
Color  Filter  Font  Path  Title
```

Methods

```
OpenFile  SaveFile  SelectColor  SelectDirectory  SelectFont
```

Examples

Back to Gambas

Examples

Gambas is delivered with some examples. You have to search in the menus.

These demo applications are small and focused on one functionality.

For advanced skills, the best example is the IDE. You can find the code in the gambas .tar archive and open it with the IDE.

TextEditor

What you will need

- 1 Form.
- 1 TextArea.
- 2 Buttons.

The code.

Note: Ill finash this later.

Array

Back to Gambas

A simple array

An array is a **list, filled with variables of the same type**.

- You can fill an array with all the weekdays of a week.
- You can fill an array with the name of all the months of a year.
- You can fill it with the names of a class in your school.
- You can fill it with the numbers 0-9 and so on.

Arrays will shorten the length of your code, because all the members of the array can be reached with one loop. Every element of an array can be reached by its position in the array. Be careful: **The first variable has the position zero.**

Example:

The following list of names shall be stored in an array

Jack, John , Anne , Alice

You can use the array command for that. The output is done with a for-each loop.

The program: You need a form and a commandbutton to get it going.

```
ar AS String[]
' a string array is defined
PUBLIC SUB Form_Open()
 ar = Array("Jack","John","Anne","Alice")
 ' the array is filled
END
PUBLIC SUB Button1_Click()
 element AS String
 FOR EACH element IN ar
  PRINT element
 NEXT
' the array is printed into the terminal window
END
```

The Array command

The Array command is used with the following syntax:

```
array = Array ( Expression , ... )
```

It creates an array and returns it. The type of the array is the type of the first expression. The other expressions are automatically converted. You can use the square bracket syntax as an alternative to the Array() subroutine.

Examples

```
PRINT Object.Type(Array(2.4, 3, 3.2))
```

> Float[]

```
PRINT Object.Type(Array("2.4", 3, 3.2))
```

> String[]

```
PRINT [ "A", "B", "C" ].Join("/")
```

> A/B/C

The Addy program

In the following example an array is filled with integers. Then the elements are looked up in a for-each loop and added together. This miniprogram calculates the sum of a row of integers. You need 1 textarea, 1 textbox and 1 commandbutton to get the program going. You can also add negative integers. Very helpful is the Splitcommand and the Stringarray String[]

The Code:

```
PUBLIC SUB Button1_Click()
text AS String
summe AS Float
elt AS String[]
Sb AS String
text = textarea1.Text
elt = Split(text,Chr(10))
FOR EACH Sb IN elt
 summe = summe + Val(sb)
NEXT
textbox1.Text = summe
END
```

When you start the programm, you can fill any number into the textarea. When you want to add another number just push the RETURN button.

The Split command

```
Array = Split ( String [ , Separators , Escape ] )
```

This command 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.

Example: you need a command button on your form to get it going.

```
PUBLIC SUB Button1_Click()
DIM Elt AS String[]
DIM Sb AS String
Elt = Split("Gambas Almost Means BASIC ! 'agree ?'", " ", "'")
FOR EACH Sb IN Elt
 PRINT Sb
NEXT
End
```

Output in the terminal window:

```
Gambas
Almost
Means
BASIC
!
agree ?
```

Arithmetic

Basic Arithmetic Operations

It's similar to other programming languages.

Use the `Float` datatype when you calculate values!

```
Dim x As Float
Dim y As Float

x = 0.32
y = x - (0.32 - 0.32 ^ 3 / (2 * 3))
```

Branch

Back to Gambas

What are branches in a program?

A branch is a point at which your program must make a choice. With these structures, it is possible to make programs that can have multiple outcomes. There are mainly two branching methods:

- if then and ist variations
- select case

If...Then statement

The if then branch works quite the same as it is used in natural language. If something is true , then make something. If it is not true then just continue.

Try this little example: You just need a command button on your form to get it going. The output is shown in the terminal window.

```
PUBLIC SUB Button1_Click()
k AS Integer
FOR k = 1 TO 10000
 IF k = 5000 THEN PRINT "5000 has been reached !"
NEXT
END
```

Try to alter the program so that the output is shown in a textbox. You have to add a textbox to your form. You will find it with F6 in your toolbox. Then you have to change the code, so that the print command is exchanged.

```
PUBLIC SUB Button1_Click()
k AS Integer
FOR k = 1 TO 10000
 IF k = 5000 THEN textbox1.text = "5000 has been reached !"
NEXT
END
```

If...Then...Else...End if

You can modify the if-then construction by adding some other commands like else and end if. Here is another example with an **else** and **end if** command in it. You just need a command button on your form to get it going.

```
 PUBLIC SUB Button1_Click()
 k AS Integer
 FOR k = 1 TO 6000
  IF k < 5000 THEN
     PRINT k
   ELSE
     PRINT "5000 has been reached!"
   END IF
 NEXT
END
```

The program counts from 1 to 6000 . When the number 5000 is reached , you get a message in the terminal window.

You have to be aware of the right outline , otherwise it wont work.

Here is another example using the toggle button: It shows a message without a messagebox and the message can be copied into the clipboard. You need a toggle button and a textarea to get it going. You should change the property of the textarea to

- Visible = false

When you click the button then the text will appear. When you click once again, the text will disappear.

```
PUBLIC SUB ToggleButton1_Click()
DIM Help AS String
 IF TextArea1.Visible = FALSE THEN
   ToggleButton1.Text = "Close Info "
   Help = "Hello, this is a info" & Chr(13) & Chr(10)
   Help = Help & "www.madeveryeasy.de" & Chr(13) & Chr(10)
   Help = Help & "rho55@gmx.de"
   TextArea1.Visible = TRUE
   TextArea1.text = Help
 ELSE
  TextArea1.Visible = FALSE
   ToggleButton1.Text = "Show Info"
 ENDIF
END
```

Select Case

If you want to branch and you have more than 2 possible answers, then you can use the select-case command. Try this little example to get used to it. You just need an empty form to get it going. The output will be done in the terminal window.

```
PUBLIC SUB Form_Open()
 PRINT Now
 PRINT WeekDay(Now)
  SELECT CASE WeekDay(Now)
   CASE 1
     PRINT "Monday"
   CASE 2
     PRINT "Tuesday"
   CASE 3
     PRINT "Wednesday"
   CASE 4
     PRINT "Thursday"
   CASE 5
     PRINT "Friday"
   CASE 6
     PRINT "Saturday"
   CASE 7
     PRINT "Sunday"
 END SELECT
END
```

Constant

Back to Gambas

Constants

Always use constants when a value doesn't change!

```
Const const_ratio As Float = 0.5
```

Loop

Back to Gambas

Loops

Introduction to loops

Loops are powerful control structures used to repeat a given section of code a certain number of times or until a particular condition is met.

Gambas has three main types of loops: **for / next** loops , **do** loops and **for / each** loops.

For / Next Loops

The syntax of a **for / next** loop has three components:

- a counter,
- a range, and
- a step.

A basic **for / next** loop appears as follows:

```
    for X = 1 to 100 step 1
       print X
    next
```

In this example, X is the counter, "1 to 100" is the range, and "1" is the step.

When you want to make a complete example out of that , you need a form and a command button and this following code.

```
PUBLIC SUB Button1_Click()
X AS Integer
 for X = 1 to 100 step 1
       print X
 next
END
```

The variable X has to be declared as an integer.

When a **for / next** loop is initialized, the counter is set to the first number in the range—in this case, X is set to 1. The program then executes any code between the **for** and **next** statements normally. Upon reaching the **next** statement, the program returns to the **for** statement and increases the value of the counter by the step. In this instance, X will be increased to 2 on the second iteration, 3 on the third, etc.

To change the amount by which the counter variable increases on each iteration, simply change the value of the step. For example, if you use a step 2, X will increase from 1 to 3, then to 5, 7, 9, and so on. When the step is not explicitly stated, step 1 is used by default. (Note that the step can be a negative value. For instance, "for X = 100 to 1 step -1" would decrease the value of X from 100 to 99 to 98, etc.)

When X reaches the maximum value in the range (100 in the example above), the loop will cease to execute, and the program will continue to the code beyond the **next** statement.

It is possible to edit the value of the counter variable within a **for / next** loop, although this is generally considered bad programming practice:

```
    for X = 1 to 100 step 1
       print X
       X = 7
    next
```

While you may on rare occasions find good reasons to edit the counter in this manner, the example above illustrates one potential pitfall: Because X is set to 7 at the end of every iteration, this code creates an infinite loop. To avoid this and other unexpected behavior, use extreme caution when editing the counter variable!

Try some more examples to get used to loops:

```
PUBLIC SUB Button1_Click()
i AS Integer
FOR i = 1 TO 10
 PRINT "Hallo Welt"
NEXT
END
```

```
PUBLIC SUB Button1_Click()
DIM i AS Integer
FOR i = 1 TO 10
  PRINT i,i*i,i*i*i
NEXT
END
```

To make loops better readable, you should move the inner lines of code to the right. They will be repeated in the loop. In contrary to VB you should not mention the counter at the end of the loop behind the next command.

```
For I = 1 to 10
 print I
' Next I    VB code
' skipped in Gambas because it is redundant
Next
```

Do Loops

Do loops are a bit more flexible than For loops, but should generally only be used when necessary. Do loops come in the following formats:
-Do while
-Do until
-Loop while
-Loop until
While loops (both do while and loop while) will continue to execute as long as a certain conditional is true. An Until loop will loop as long as a certain condition is false, on the other hand. The only difference between putting either While or Until in the Do section or the Loop section, is that Do checks when the loop starts, and Loop checks when the loop ends. An example of a basic loop is as follows:

```
    Do
       print "hello"
       x=x+1
    Loop until x=10
```

This loop will print hello several times (depending on the initial value of x). As you may have noticed, Do loops have no built in counters. However, they may be made manually as shown above. In this case, I chose x as my counter variable, and every time the loop executes, 'x' increases itself by one. When 'x' reaches 10, the loop will cease to execute. The advantage of Do loops is that you may exit at any time whenever any certain conditional is met. You may have it loop as long as a certain variable is false, or true, or as long as a variable remains in a certain range.

Nested Loops

A nested loop is any type of loop inside an already existed loop. They can involve any type of loop. For this, we will use For loops. It is important to remember that the inside loop will execute its normal amount multiplied by how many times the exterior loop runs. For example:

```
    For i = 1 to 10
       For j = 1 to 10
           print i*j,
       Next
      print
    Next
```

This will print the times table. Upon the first pass of the i loop, it will run the j loop ten times. Then on the second pass of the i loop, it will run the j loop another ten times, and so on. Why is there another print command between the two next commands. Try it out, what you will get, when you delete it.

Subroutine

Subroutine in Gambas is a piece of code maintained as a block usable in many parts of your program. You need not write it twice. Routines can return a value depending on their return type. For instance we can return a String or Integer. Let's check the example below:

```
Public Sub Multiplying() As String
  Dim n, v1, v2 As Integer
  Dim res As String
  n = 4 'multiplied by 9
  v1 = (n - 1) '3
  v2 = (10 - n) '6
  res = v1 & v2 'result 36
  Print res '4*9=36
  Return "9*" & n & "=" & res
End
```

Later on we can use this subroutine in main program as:

```
Public Sub Form_Open()
  Message.Info(Multiplying())
End
```

We declare return type of subroutine after AS written in close relation to the name of subroutine as we can see in examples above. There are some built in subroutines connected with events of windowed program. Such a type is Form_Open() fired when form opened. Special name for subroutines used in classes based on the object oriented principle is methods. To say it simple: Subroutine is reusable part of code. Another example in our code might be:

```
Public Sub sayHello()
  Message.Info("Hello Maria")
End
```

After creating of subroutine we can use it in similar way:

```
Public Sub Form_Open()
 sayHello()
 Message.Info(Multiplying())
End
```

String Manipulation

String Concatenation

The **concatenation operator** represented by an ampersand symbol is used to join two strings together:

```
DIM bestclub AS String
bestclub = "Liverpool" & " Football Club"    ' Liverpool Football Club
```

The addition operator cannot be used to concatenate strings

In gambas, unlike traditional basic, the plus symbol cannot be used as concatenation operator. If an attempt is made to use a plus symbol for this purpose, a type mismatch error will occur when an attempt is made to run the program:

```
' This does not work in gambas
PRINT "I have 3 cats" + " and 2 dogs"    ' You cannot concatenate strings by using a plus symbol
```

Concatenation combination assignment operator

The **concatenation combination assignment operator** represented by a andequals symbol, can also be used to concatenate strings:

```
DIM breakfast AS String
breakfast = "bacon"
breakfast &= " and eggs"    ' breakfast = breakfast & " and eggs" (using the andequals symbol)
```

Search and Replace

hello i have made a try to search in a text file -two function : 1- search by line and column 2-search by pos

-first function returns an array with results it will be for example like this ["1,12" , "2,1"] , the first element is the line the second is the column

-second function returns also an array but single one ex:[1,2,45,455] every element is the string pos

here you are ! :

```
AUTHOR: abom
MAIL: abom@linuxac.org

----

STATIC PUBLIC SUB searchfile(filename AS String, strser AS String) AS String[]

  DIM icounter AS Integer, fn AS stream, ln AS String, wpos AS Integer, rstr AS NEW String[]
  DIM spos AS Integer
  icounter = 0

  IF Exist(filename) THEN

     fn = OPEN filename FOR READ

      WHILE NOT Eof(fn)

        LINE INPUT #fn, ln
        icounter = icounter + 1
          IF strser = "" THEN RETURN
          spos = 0
          wpos = 0
          DO
           wpos = InStr(ln, strser, spos, gb.Text)

          IF wpos <> 0 THEN
             spos = wpos + Len(strser)
             rstr.add(icounter & "," & wpos)
          END IF
          LOOP UNTIL (wpos = 0)
      WEND

  END IF
  RETURN rstr

END
----
STATIC PUBLIC SUB searchfilebypos(filename AS String, strser AS String) AS String[]

  DIM icounter AS Integer, fn AS stream, txt AS String, wpos AS Integer, rstr AS NEW String[]
  DIM spos AS Integer
  icounter = 0

  IF Exist(filename) THEN
       wpos = 0
       spos = 0

       txt = file.Load(filename)

       IF strser = "" THEN RETURN
       DO

           wpos = InStr(txt, strser, spos, gb.Text)

          IF wpos <> 0 THEN
             spos = wpos + Len(strser)
             rstr.add(wpos)
          END IF

       LOOP UNTIL (wpos = 0)
  END IF
  RETURN rstr

END
```

Environment

Back to Gambas

Environment variables and command line parameters

Please refer to the official wiki:

- http://gambaswiki.org/wiki/comp/gb/args
- http://gambaswiki.org/wiki/comp/gb/env

A small example :

```
Dim arg as String
arg = Args[0] ' Get the first commandline parameter
Print arg
Print Env["USER"] ' case sensitive!
```

Files

Back to Gambas

File handling

Please refer to the official wiki: http://gambaswiki.org/wiki/lang/open

You can read a file with some text for example.

With Gambas 1

```
' Gambas 1
Dim OneLine As String
Dim FileName As String = "/home/moi/test.txt"
Dim hFile As File

OPEN FileName FOR READ AS #hFile
WHILE NOT Eof(hFile)
  LINE INPUT #hFile, OneLine
  PRINT OneLine
WEND
CLOSE #hFile
```

It's a bit easier with Gambas 3.

```
' Gambas 3
 Dim hFile As File
 Dim OneLine As String
 Dim FileName As String = "/home/moi/test.txt"

 hFile = Open FileName For Read
 While Not Eof(hFile)
   Line Input #hFile, OneLine
   Label1.Text = OneLine
 Wend
 Close #hFile
```

Exist

back to Gambas

Does a File or Directory exist ?

Example:

```
PUBLIC SUB Button1_Click()
   IF Exist("/bin/bash") = TRUE THEN
        Label1.Text="OK i found Bash"
   END IF
   IF Exist("/bin/bash") = FALSE THEN
        Label1.Text="Error Bash is missing"
   END IF
END
```

Get system's root / home folder ?

Example :

```
'Created : Cijo Jose
'E-mail  : info.cijo@gmail.com
'Job ID  : Junior Software Engineer

PUBLIC SUB Button1_Click()

       dir_name as string
       dir_name = system.root    ' Will return root folder
       print dir_name
       dir_name = system.home    ' Will return home folder
       print dir_name

END
```

When you click on your button your gambas program checks for Bash if Bash is found then Label1 displays "OK i found Bash" if Bash is not found then Label1 displays "Error Bash is missing"

Gambas.SteveStarr - 02 Aug 2005

Object Oriented Programming

Back to Gambas

Object Oriented Programming in Gambas

Gambas is not a fully Object Oriented Programming Language but it's tending to approach this.

Gambas lets you create objects with your own classes. You can use events, simple inheritance, polymorphism and so on.

Please read the official wiki! http://gambaswiki.org/wiki/doc/object-model

Operators

The Gambas programming language provides a set of **operators** that allow values and variables to be compared, evaluated or manipulated. These operators can combined together with values and variables to make expressions. The operators may take unary, dyadic or ternary form, depending on the number of operands that they utilize.

Assignment operator

In Gambas, as with most other programming languages, the **equals sign** acts as a dyadic **assignment operator** that assigns the value of the expression of the right hand operand to the variable named by the left operand.

Nudge operators

The nudge operators ++ and -- are not supported in Gambas. However the INC and DEC commands and combination assignment operators can be used to increment or decrement the values of variables.

Variable

Back to Gambas

Variable Declaration (Dimensioning)

In Gambas it is necessary to declare every variable. To dimension a variable, the code is:

```
Dim variablename as type
```

You can just skip the word Dim , so the following is also correct:

```
variablename as type
```

The variable name could be whatever you want. The type however has to be chosen. You have a choice of integer, string, variant and some others. This tells the computer what type of information the variable holds. Singles and integers hold numbers, while strings hold text.

```
'Dimensioning
Dim i as Integer
Dim x as Variant
'assignment
i = 31            'This is ok
i = "i do"     'Error: type mismatch(i is an integer;"i do" is a string)
x = 341           'This is ok
x = "books"       'This is all right
```

In Gambas you cannot dim an array as you were used in VB. The following is wrong:

```
    xxxxx  Dim x(1 to 10) as integer  xxxxxx
```

Dimensioning can also be used to tell the computer that variables are public or private. When a variable is public, it may be used and changed in all sub-programs. To make a variable public you must dim it in the "declarations" area of the code. These declarations are placed on the very first lines in your code, above all sub programs and other code.

Examples: You need a command button on your form to get it going.

```
PUBLIC SUB Button1_Click()
'Declaration
x AS integer
y AS float
z AS String
'assignment
x = 2
y = 2.378
z = "That ist correct"
'the use of variables
print x,y,z,x*y
END
```

The following program will cause an error: x already declared

```
'xxxxxxxxxxxxxxxxxxxx
PUBLIC SUB Button1_Click()
DIM x AS integer
DIM x AS float
DIM x AS String
x = 2
x = 2.378
y = "That ist not correct"
print x,x,x
END
'xxxxxxxxxxxxxxxxxxxxxx
```

In the following program you also cause some error warnings, as there is no difference between upcase and lowcase signs in Gambas.

```
PUBLIC SUB Button1_Click()
DIM x AS integer
DIM X AS float
DIM x$ AS String
x = 2
X = 2.378
x$ = "that ist correct "
print x,X,x$
END
```

List of Gambas predefined datatypes for the declaration of variables

Name	Description	Memory size	Default value
`Boolean`	True or False	1 byte	False
`Byte`	0 ... 255	1 byte	0
`Short`	-32768 ... +32767	2 bytes	0
`Integer`	-2147483648 ... +2147483647	4 bytes	0
`Float`	Like the *double* datatype in *C*	8 bytes	0.0
`Date`	Date and time, each stored in an *integer* .	8 bytes	Null
`String`	A reference to a variable length string.	4 bytes	Null
`Variant`	Any datatype.	12 bytes	Null
`Object`	A anonymous reference to an object.	4 bytes	Null

Assignment

Once you have declared a variable , you can assign a value to it.

```
Variable = Expression
```

This will assign the value of an expression to one of the following elements :

- A local variable.
- A function parameter.
- A global (class) variable.
- An array slot.
- An object public variable.
- An object property.

Example

```
iVal = 1972
Name = "Gambas"
hObject.Property = iVal
cCollection[sKey] = Name
```

Structures

Some versions of basic, allow structured variables to be created by using TYPE definitions as follows:

```
' This will not work in gambas
TYPE rhubarbstructure
  foo AS STRING * 32
  bar AS INTEGER
END TYPE
```

```
PUBLIC rhubarb AS rhubarbstructure
```

Currently, gambas does not support the TYPE keyword. However, it is possible to use class definitions for the same effect.

Creating a class

From the Integrated Development Environment file, create a class file rhubarbstructure.class as follows:

```
PUBLIC foo AS STRING
PUBLIC bar AS INTEGER
```

It is now possible to define a variable utilizing that class, by creating an appropriate definition in the code module as follows:

```
DIM rhubarb AS rhubarbstructure
```

Concept Index

This is the concept index for the Gambas programming book.

- whitespace

X

Y

Z

Function Reference

Back to Gambas

Function Reference

Gambas has a lot of functions!

- http://gambaswiki.org/wiki/cat/arith
- http://gambaswiki.org/wiki/cat/trigo
- http://gambaswiki.org/wiki/cat/bit
- http://gambaswiki.org/wiki/cat/externfunc
- http://gambaswiki.org/wiki/cat/conv
- http://gambaswiki.org/wiki/cat/char
- http://gambaswiki.org/wiki/cat/type
- http://gambaswiki.org/wiki/lang/localize
- http://gambaswiki.org/wiki/cat/time

Sorry. I cannot list all functions!

Useful Modules

Back to Gambas

Useful Modules

All useful modules are grouped in components.

Try to master the Settings component at the beginning!

In project properties, in components, activate the `gb.settings`!

You can use it like this (for example, in a Form named FMain with a Label named Label1) :

```
' Gambas class file

' FMain

Private $counter As Integer = 0

Public Sub Form_MouseUp()

  Settings.Reload
  Message.Info(Settings["My/Ref", "default"])

End

Public Sub Label1_MouseDown()

  Inc $counter
  Settings["My/Ref"] = CStr($counter)
  Settings.Save

End
```

Reference Cards

Back to Gambas

Nothing for now.

Websites

Back to Gambas

Nothing for now.

Constants

Back to Gambas

Predefined Constants in Gambas

Gambas has some predefined constants.

Please read the official wiki! http://gambaswiki.org/wiki/cat/constant

Here is a trick!

- Type `gb.`!
- After the dot, you will see these constants.

Commands

Back to Gambas

See

The actual life wiki (2006) http://gambasdoc.org/help/lang

The static wiki (old) http://www.binara.net/gambas-wiki/static/Gambas/IndexByName.html

Components

Back to Gambas

GB Components

Components are Gambas plug-ins that add new functionality to your applications (by adding new classes to the interpreter).

A Gambas project has a list of components to be loaded at startup; you can find and modify this list by selecting Project > Properties from the Project window, and clicking the Components tab.

A Gambas project without any components is a simple text-only application. To become a true graphical application, a project must use at least the gb.qt component. To access a database, use the gb.db component. And so forth...

- 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.gtk - Graphical GTK+ toolkit component
- gb.net - Networking component
- gb.net.curl - Network high-level protocols management
- gb.pcre - Perl-compatible Regular Expression Matching
- gb.qt - Graphical QT toolkit component
- gb.qt.editor - Gambas editor with syntax highlighting
- gb.qt.ext - Graphical QT toolkit extension component
- gb.qt.kde - KDE integration and scripting
- gb.qt.kde.html - KDE web browser
- gb.sdl - Library based on SDL
- gb.sdl.image - 2D Library based on SDL
- gb.sdl.sound - Sound library based on SDL
- 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

Add some new components

You can load new components using by going to the main Project window and selecting Project > Properties.

Now, choosing the Components tab, you'll find a selectable list of the Gambas components. Tick the ones you want, untick the ones you don't.

You'll also be shown a hint with the name of the additional controls e.g.: QT when you load gb.qt.ext.

List of the Gambas components

- 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
        - Controls: DnsClient, ServerSocket, SerialPort, Socket, UdpSocket
- gb.qt
    - Graphical QT toolkit component
        - Controls: 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
        - Controls: GambasEditor
- gb.qt.ext
    - Graphical QT toolkit extension component
        - Controls: LCDNumber, Dial, SpinBox, ScrollBar, Slider, MovieBox, TableView, HSplit, VSplit, Workspace, TextView
- gb.qt.kde
    - KDE integration and scripting
        - Controls: URLLabel, ColorBox, DatePicker

Experimental Components

WARNING: These are in the BETA stage of development and might be changed at any time.

- gb.qt.kde.html
    - KDE web browser
- 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

VB

Back to Gambas

See

The actual Gambas wiki (2006) http://gambasdoc.org/help/doc/diffvb

The old wiki, now static http://www.binara.net/gambas-wiki/static/Gambas/DifferencesFromVB.html

Error

Error Handling

To avoid an error from crashing your program, you can execute the program line under provision and afterwards check if an error occurred.

The line that might give an error is

```
doSomething(iParameter)
```

The first step is to use TRY

```
TRY doSomething(iParameter)
```

Then you have to catch the error:

```
IF ERROR
  doLogError(Error.Text)
ENDIF
```

The error handling can by anything, from DEBUG statements to show the error in the debug window while working on the program, over just logging the error somewhere, to more extensive error handling. The complete code becomes:

```
TRY doSomething(iParameter)
IF ERROR
  doLogError(Error.Text)
ENDIF
```

Typical use for TRY / IF ERROR is when doSomething is reading records from a database; the database connection might be broken and that might crash your program here.

Difficult

What I think makes Gambas hard to use.

The only thing I find hard about Gambas is the lack of good tutorials.

Take the "goto" tutorial for example. I think this it is a very bad example on how to use the goto command.

Why? Because after studying it I still have trouble using the goto command. What would have been better is if the example just showed me how to goto a certain part.

like this:

- a = print "hello"
- goto a

or

- a = print "hello"
- b = print "gamabs"
- goto a then b
- goto b then a

Edit 10/3/06

First be aware that goto is bad programming practice. It is a holdover from the old days of GWBasic with its line numbers and lack of proper control structures.

Using goto creates code that is hard to follow and debug and lacks logical structure - so called spaghetti code.

That said, goto works like this

goto a

a:

Print "Hello World"

b:

Print "Hello Wicked World"

The output of this will be BOTH print statements because goto simply branches to the defined label and then continues to the end of the sub/function.

Change the goto to 'goto b" and only the second statement prints.

This example will not work in Gambas, but it wasn't meant to, I just wrote this to help you understand what I mean. I think all Gambas tutorials should start with very simple examples and work their way up to advance examples.

FAQ

Back to Gambas

See http://sourceforge.net/mailarchive/forum.php?forum=gambas-user

Hotkeys

If you want to use some hotkeys for a project, you might code something like this:

```
Public sub Form_KeyPress ()
 If key.code = ... then
 Something happens
```

Unfortunately this only works, if the form has no buttons and other controls. If the form has other controls and one of the controls has the focus, then it does not work.

There is no interface in the QT component to globally intercept key events yet, but there is a trick:

If you have a menu in your form, its keyboard shortcuts are globally managed. So you can add an hidden menu to your form to solve your problem.

KDE panel applet ?

Is it possible to write a KDE panel applet with Gambas?

No, currently it is not possible to do that. KDE panel applets are implemented as shared libraries with specific interfaces.

Listview

How can I add a listview item after an existing one, when there is an item missing:

Wrong: Listview.add("Key","Name",,"Key2") -- Error: Comma missing.

You have to pass NULL as picture argument. You cannot "jump" arguments when you call a function in Gambas.

Listview.add("Key","Name",NULL,"Key2")

Variables

Does Gambas support structured data types?

In some versions of basic, it is possible to create structured data types using by using type definitions as follows:

```
' This will not work in Gambas
TYPE rhubarbstructure
  foo AS STRING * 32
  bar AS INTEGER
END TYPE
```

```
PUBLIC rhubarb AS rhubarbstructure
```

Currently, Gambas does not support the TYPE keyword. Instead, you can use the STRUCT or STRUCT[] keywords.

Additionally, it is possible to use class definitions for the same effect.

Project Building

Does gambas support include files?

Does gambas support conditional compilation directives?

Application Startup

Making the application startup from a main routine

In gambas, to make an application startup from a main routine:

- Create a new module called MMain
- In the MMain module, create a public sub called Main as follows:

```
PUBLIC SUB Main()
  ' This is the start of the program
END
```

- Right click the MMain module, then select Startup class from the context menu

Obtaining the command line parameters

```
PUBLIC SUB main()
  ' This reads and displays the command line parameters
  DIM l AS Integer
  DIM numparms AS Integer
  DIM parm AS String
  numparms = Application.Args.Count
  FOR l = 0 TO numparms - 1
    parm = Application.Args[l]
    PRINT l; " : "; parm
  NEXT
END SUB
```

Window Geometry

Determining the maximum window size

Overview

In gambas, the trick to determining the maximum window size that will fit on the screen is to create a form that is maximized and then query its dimensions from within a Form_Resize() event. Note that the form can be invisible during this process, and typically we would use the main modal window (FMain in this example).

Creating the form

From with the project create a form (FMain) with the following properties set:

```
FMain.Maximized = True
FMain.Visible = False    ' The form can be invisible
```

From within the project view, rightclick the FMain form and select Edit class from the contextmenu. This will display a form class file (FMain.class) as follows:

```
PUBLIC SUB _new()
```

```
END
```

```
PUBLIC SUB Form_Open()
```

```
END
```

We can now add a Form_Resize() event to the class file with the necessary code to obtain the screen dimensions as follows:

```
PUBLIC SUB Form_Resize()
  PRINT "The maximum window size that can be used is "; FMain.Width; " x "; FMain.Height
END
```

Links

Gambas

- http://gambas.sourceforge.net/
  - The main Gambas Site
- http://www.gambasdoc.org/help/
  - All Commands
- http://www.gambas-es.org/
  - Spanish Gambas Site
- http://www.gambas-it.org/wp/
  - Italian Gambas Site
- http://gambaslinux.eg2.fr/index.php?lng=fr
  - France Gambas Site
- http://wiki.gambas-es.org/index.php?title=Tabla_de_Traducci%C3%B3n_de_ordenes_de_VB6_a_Gambas
  - Gambas in comparison to VB
- http://sourceforge.net/mailarchive/forum.php?forum=gambas-user
  - Forum
- http://web.archive.org/20110927000819/jsbsan.blogspot.com/p/manuales.html
  - Manuales en español
- http://jsbsan.blogspot.com/p/video-tutoriales.html
  - Video tutoriales en español
- http://www.kudla.org/index.php?wl_mode=more&wl_eid=73 an activist
- http://groups.yahoo.com/subscribe/GAMBas-it
  - The Italian Mailing list and support
- http://bedna.kdesi.sk/index.php?odkaz=15
  - Gambas tutorial v Slovencine

- http://slashdot.org/article.pl?sid=04/10/26/142212
  - Slashdot has recognized Gambas

Learning Basic

- http://jsbsan.blogspot.com/
  - Gambas: Blog 2 etapa: con pequeños programas y explicaciones en Español. Ademas de enlaces a foros y paginas relacionadas con Gambas (jsbsan)
- http://web.archive.org/20090411104032/jsbsan.wordpress.com/
  - Gambas: Blog con pequeños programas y explicaciones en Español. Ademas de enlaces a foros y paginas relacionadas con Gambas (jsbsan)

Retrieved from "https://en.wikibooks.org/w/index.php?title=Gambas/Printable_version&oldid=4083873"

This page was last edited on 12 July 2022, at 06:12.