The KTurtle Handbook

Cies Breijs, Anne-Marie Mahfouf, and Mauricio Piacentini

Contents

1 Introduction **1**

1.1 What is TurtleScript? 1

1.2 Features of KTurtle 2

2 Using KTurtle **3**

2.1 The Code Editor 4

2.2 The Inspector 4

2.3 The Canvas 4

2.4 The Menubar 4

2.4.1 The File Menu 5

2.4.2 The Edit Menu 6

2.4.3 The View Menu 6

2.4.4 The Tools Menu 6

2.4.5 The Settings Menu 7

2.4.6 The Help Menu 7

2.5 The Toolbar 7

2.6 The Statusbar 8

3 Getting Started **9**

3.1 First steps with TurtleScript: meet the Turtle! 10

3.1.1 The Turtle Moves 10

3.1.2 More examples 10

4 TurtleScript Programming Reference **13**

4.1 Different Instruction Types 13

4.1.1 Commands 13

4.1.2 Numbers 13

4.1.3 Strings . . . 14
4.1.4 Names . . . 14
4.1.5 Assignments . . . 15
4.1.6 Math Symbols . . . 15
4.1.7 Questions . . . 15
4.1.8 Question Glue-Words . . . 15
4.1.9 Comments . . . 16
4.2 Commands . . . 16
4.2.1 Moving the turtle . . . 16
4.2.2 Where is the turtle? . . . 18
4.2.3 The turtle has a pen . . . 18
4.2.4 Commands to control the canvas . . . 19
4.2.5 Commands to clean up . . . 19
4.2.6 The turtle is a sprite . . . 19
4.2.7 Can the turtle write? . . . 20
4.2.8 A command that rolls dice for you . . . 20
4.2.9 Input and feedback though dialogs . . . 21
4.3 Containers . . . 22
4.3.1 Variables: number containers . . . 22
4.3.2 Containers that contain text (strings) . . . 23
4.4 Can the Turtle do math? . . . 23
4.5 Asking questions, getting answers... . . . 24
4.5.1 Questions . . . 24
4.5.2 Question Glue . . . 24
4.5.2.1 and . . . 26
4.5.2.2 or . . . 26
4.5.2.3 not . . . 26
4.6 Controlling execution . . . 27
4.6.1 Have the turtle wait . . . 27
4.6.2 Execute "if" . . . 27
4.6.3 If not, in other words: "else" . . . 28
4.6.4 The "while" loop . . . 28
4.6.5 The "repeat" loop . . . 29
4.6.6 The "for" loop, a counting loop . . . 29
4.6.7 Stop the turtle . . . 29
4.7 Create your own commands with ‘learn’ . . . 29

5 Glossary **32**

6 Translator's Guide to KTurtle **36**

7 Credits and License **37**

A Installation **38**
A.1 How to obtain KTurtle 38
A.2 Compilation and Installation 38

B Index **39**

List of Tables

4.2 Types of questions 25
4.4 Question glue-words 25

5.2 Different types of code and their highlight color 34
5.4 Often used RGB combinations 35

Abstract

KTurtle is an educational programming environment that uses TurtleScript, a programming language inspired by Logo. The main quality of TurtleScript is that the programming commands can be translated to the language of the 'programmer' so he/she can program in his/her native language and KTurtle programming language reproduces this feature.

Chapter 1

Introduction

KTurtle is an educational programming environment that uses TurtleScript, a programming language loosely based on and inspired by Logo. The goal of KTurtle is to make programming as easy and accessible as possible. This makes KTurtle suitable for teaching kids the basics of math, geometry and... programming. One of the main features of TurtleScript is the ability to translate the commands into the speaking language of the programmer.

KTurtle is named after 'the turtle' that plays a central role in the programming environment. The user programs the turtle, using the TurtleScript commands, to draw a picture on the canvas.

1.1 What is TurtleScript?

TurtleScript, the programming language used in KTurtle, is heavily inspired by some of the fundamental concepts of the Logo programming language. The first version of Logo was created by Seymour Papert of MIT Artificial Intelligence Laboratory in 1967 as an offshoot of the LISP programming language. From then many versions of Logo have been released. By 1980 Logo was gaining momentum, with versions for MSX, Commodore, Atari, Apple II and IBM PC systems. These versions were mainly for educational purposes. LCSI released Mac®Logo in 1985 as a tool for professional programmers, but it never caught on. MIT is still maintaining a site on Logo which can be found on http://el.media.mit.edu/logo- foundation/ .

Today there are several versions of Logo around which can easily be found on MIT's Logo site and by a simple Google search.

In comparison with most modern versions of Logo, TurtleScript implements only the basic commands best suited for the educational purposes of the language, and does not try to fulfill professional programmer's needs.

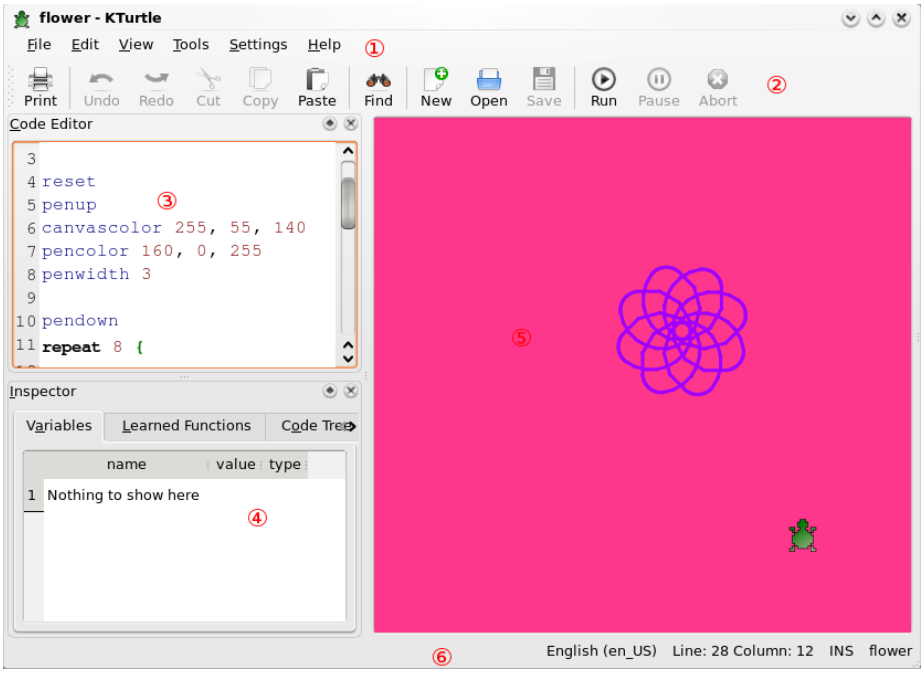
1.2 Features of KTurtle

KTurtle has some nice features that make starting to program a breeze. See here some of the highlights of KTurtle feature set:

- An integrated TurtleScript interpreter (no extra dependencies) that includes the ability to use a fully translated set of commands, and supports user defined functions and dynamic type switching.
- The execution can be slown down, paused or stopped at any time.
- A powerful editor for the TurtleScript commands with intuitive syntax highlighting, line numbering and more.
- The TurtleScript commands are fully translatable.
- An error dialog that links the error messages to the mistakes in the program.
- Simplified programming terminology.
- Many integrated, internationalized example programs make it easy to get started.

Chapter 2

Using KTurtle



The main window of KTurtle has three main parts: the code editor (3) on the left where you type the TurtleScript commands, the inspector (4) which gives you information about variables when the program runs, and the canvas (5) on the right where the instructions are visualized. The canvas is the turtle's playground: it is on the canvas that the turtle actually moves and draws. The three other places on the main window are: the menu bar (1) from where all the actions can be reached, the toolbar (2) that allows you to quickly select the most used actions, and the statusbar (6) where you will find feedback on the state of KTurtle.

2.1 The Code Editor

In the code editor you type the TurtleScript commands. It has all of the features you would expect from a modern editor. Most of its features are found in the Edit and the Tools menus. The code editor can be docked on each border of the main window or it can be detached and placed anywhere on your desktop.

You have several ways to get some code in the editor. The easiest way is to use an already-made example: choose File → Examples in the File menu and select an example. The file example you choose will be opened in the the code editor, you can then use File → Run to run the code if you like.

You can open TurtleScript files by choosing File → Open....

The third way is to directly type your own code in the editor or to copy/paste some code from this user guide.

The cursor position is indicated in the statusbar, on the right with the Line number and Column number.

2.2 The Inspector

The inspector informs you about the variables, the learned functions and the code treee while the program is running.

The inspector can be docked on each border of the main window or it can be detached and placed anywhere on your desktop.

2.3 The Canvas

The canvas is the area where the commands are visualized, where the commands 'draw' a picture. In other words, it is the turtle's playground. After getting some code in the the code editor, and executing it using File → Run, two things can happen: either the code executes fine, and will you most likely see something change on the canvas; or you have made an error in your code and there will be a message telling you what error you made.

This message should help you to resolve the error.

You can zoom in and out the canvas with your mouse wheel.

2.4 The Menubar

In the menubar you find all the actions of KTurtle. They are in the following groups: File, Edit, View, Tools, Settings, and Help. This section describes them all.

2.4.1 The File Menu

File → New (Ctrl-N) Creates a new, empty TurtleScript file.

File → Open... (Ctrl-O) Opens a TurtleScript file.

File → Open Recent Opens a TurtleScript file that has been opened recently.

File → Save (Ctrl-S) Saves the currently opened TurtleScript file.

File → Save As... Saves the currently opened TurtleScript file on a specified location.

File → Examples Open example TurtleScript programs. The examples are in your favorite language that you can choose in Settings → Script Language.

File → Speed Present a list of possible execution speeds, consisting of: Full Speed, Slow, Slower, Slowest and Step-by-Step. When the execution speed is set to Full Speed (default) we can barely keep up with what is happening. Sometimes this behavior is wanted, but sometimes we want to keep track of the execution. In the latter case you want to set the execution speed to Slow, Slower or Slowest. When one of the slow modes is selected the current position of the executor will be shown in the editor. Step-by-Step will execute one command at a time.

File → Run (F5) Starts the execution of the commands in the code editor.

File → Pause (F6) Pauses the execution. This action is only enabled when the commands are actually executing.

File → Abort (F7) Stops the execution. This action is only enabled when the commands are actually executing.

File → Print... (Ctrl-P) Prints the current code in the editor.

File → Quit (Ctrl-Q) Quits KTurtle.

2.4.2 The Edit Menu

Edit → Undo (Ctrl-Z) Undoes the last change to code. KTurtle has unlimited undos.

Edit → Redo (Ctrl-Shift-Z) Redoes an undone change to the code.

Edit → Cut (Ctrl-X) Cuts the selected text from the code editor to the clipboard.

Edit → Copy (Ctrl-C) Copies the selected text from the code editor to the clipboard.

Edit → Paste (Ctrl-V) Pastes the text from the clipboard to the editor.

Edit → Select All (Ctrl-A) Selects all the text from the editor.

Edit → Find... (Ctrl-F) With this action you can find phrases in the code.

Edit → Find Next (F3) Use this to find the next occurrence of the phrase.

Edit → Find Previous (Shift-F3) Use this to find the previous occurrence of the phrase.

Edit → Toggle Insert (Ins) Toggle between the 'insert' and 'overwrite' mode.

2.4.3 The View Menu

View → Show Code Editor Show or hide the Code Editor.

View → Show Line Numbers (F11) With this action you can show the line numbers in the code editor. This can be handy for finding errors.

View → Show Inspector Show or hide the Inspector.

2.4.4 The Tools Menu

Tools → Direction Chooser This action opens the direction chooser dialog.

2.4.5 The Settings Menu

Settings → **Script Language** Choose the language for the code.

Settings → **Show Toolbar** Toggle the Main Toolbar

Settings → **Show Statusbar** Toggle the Statusbar

Settings → **Configure Shortcuts...** Standard KDE dialog to configure the shortcuts.

Settings → **Configure Toolbars...** Standard KDE dialog to configure the toolbars.

2.4.6 The Help Menu

Help → **KTurtle Handbook (F1)** This action shows the handbook that you are currently reading.

Help → **What's This? (Shift-F1)** After activating this action the mouse arrow will be changed into a 'question mark arrow'. When this arrow is used to click on parts of KTurtle main window, a description of the particular part pops-up.

Help → **Report Bug...** Use this to report a problem with KTurtle to the developers. These reports can be used to make future versions of KTurtle even better.

Help → **About KTurtle** Here you find information on KTurtle, like the authors and the license it comes with.

Help → **About KDE** Here you can find information on KDE. If you do not know yet what KDE is, this is a place you should not miss.

2.5 The Toolbar

Here you can quickly reach the most used actions. By default, you will find here all main useful commands ending with the Run, Pause and Abort icons.

You can configure the toolbar using Settings → Configure Toolbars...

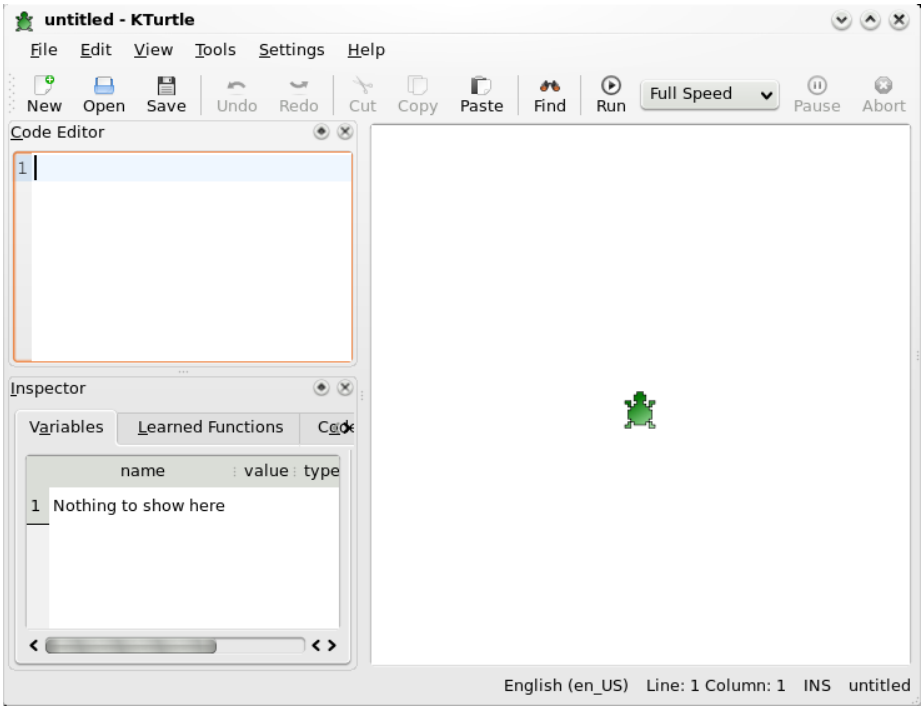
2.6 The Statusbar

On the status bar you get feedback of the state of KTurtle. On the left side it shows the feedback on the last action. On the right side you find the current location of the cursor (line and column numbers). In the middle of the status bar is indicated the current language used for the commands.

Chapter 3

Getting Started

When you start KTurtle you will see something like this:



In this Getting Started guide we assume that the language of the TurtleScript commands is English. You can change this language with Settings → Script Language. Be aware that the language you set here for KTurtle is the one you use to type the TurtleScript commands, not the language used by KDE on your computer and used to display the KTurtle interface and menus.

3.1 First steps with TurtleScript: meet the Turtle!

You must have noticed the turtle in the middle of the canvas: you are just about to learn how to control it using commands in the code editor.

3.1.1 The Turtle Moves

Let us start by getting the turtle moving. Our turtle can do 3 types of moves, (1) it can go forwards and backwards, (2) it can turn left and right and (3) it can go directly to a position on the screen. Try this for example:

```
forward 100
turnleft 90
```

Type or copy-paste the code to the code editor and execute it (using FileRun) to see the result.

When you typed and executed the commands like above in the code editor you might have noticed one or more of the following things:

1. That — after executing the commands — the turtle moves up, draws a line, and then turns a quarter turn to the left. This because you have used the forward and the turnleft commands.
2. That the color of the code changed while you where typing it: this feature is called *intuitive highlighting* — different types of commands are highlighted differently. This makes reading large blocks of code more easy.
3. That the turtle draws a thin black line.
4. Maybe you got an error message. This could simply mean two things: you could have made a mistake while copying the commands, or you should still set the correct language for the TurtleScript commands (you can do that by choosing Settings → Script Language).

You will likely understand that `forward 100` commanded the turtle to move forward leaving a line, and that `turnleft 90` commanded the turtle to turn 90 degrees to the left.

Please see the following links to the reference manual for a complete explanation of the new commands: forward, backward, turnleft, and turnright.

3.1.2 More examples

The first example was very simple, so let us go on!

```
reset

canvassize 200,200
canvascolor 0,0,0
pencolor 255,0,0
penwidth 5

go 20,20
direction 135

forward 200
turnleft 135
forward 100
turnleft 135
forward 141
turnleft 135
forward 100
turnleft 45

go 40, 100
```

Again you can type or copy-paste the code to the code editor or open the `arrow` example in the Examples menu and execute it (using FileRun) to see the result. In the next examples you are expected to know the drill.

You might have noticed that this second example uses a lot more code. You have also seen a couple of new commands. Here a short explanation of all the new commands:

After a `reset` command everything is like is was when you had just started KTurtle.

`canvassize 200,200` sets the canvas width and height to 200 pixels. The width and the height are equal, so the canvas will be a square.

`canvascolor 0,0,0` makes the canvas black. `0,0,0` is a RGB-combination where all values are set to `0`, which results in black.

`pencolor 255,0,0` sets the color of the pen to red. `255,0,0` is a RGB-combination where only the red value is set to `255` (fully on) while the others (green and blue) are set to `0` (fully off). This results in a bright shade of red.

If you do not understand the color values, be sure to read the glossary on RGB-combinations

`penwidth 5` sets the width (the size) of the pen to `5` pixels. From now on every line the turtle draw will have a thickness of `5`, until we change the `penwidth` to something else.

`go 20,20` commands the turtle to go to a certain place on the canvas. Counted from the upper left corner, this place is 20 pixels across from the left, and 20 pixels down from the top of the canvas. Note that using the `go` command the turtle will not draw a line.

`direction 135` set the turtle's direction. The `turnleft` and `turnright` commands change the turtle's angle starting from its current direction. The `direction` command changes the turtle's angle from zero, and thus is not relative to the turtle previous direction.

After the `direction` command a lot of `forward` and `turnleft` commands follow. These command do the actual drawing.

At last another `go` command is used to move the turtle aside.

Make sure you follow the links to the reference. The reference explains each command more thoroughly.

Chapter 4

TurtleScript Programming Reference

This is the reference for KTurtle's TurtleScript. In this chapter we first briefly touch all the different instruction types. Then the commands are explained one by one. Then containers, math, questions and execution controllers are explained. At last you are shown how to create you own commands with learn.

4.1 Different Instruction Types

As in any language, TurtleScript has different types of words and symbols. Here the differences between the types are briefly explained.

4.1.1 Commands

Using commands you tell the turtle or KTurtle to do something. Some commands need input, some give output.

```
# forward is a command that needs input, in this case the ↩
    number 100:
forward 100
```

For a detailed overview of all commands that KTurtle supports go here.

4.1.2 Numbers

Most likely you already know quite a bit about numbers. The way numbers are used in KTurtle is not much different from spoken language, or math.

We have the so called natural numbers: `0`, `1`, `2`, `3`, `4`, `5`, etc. The negative numbers: `-1`, `-2`, `-3`, etc. And the numbers with decimals, or dot-numbers, for example: `0.1`, `3.14`, `33.3333`, `-5.05`, `-1.0`.

Numbers can be used in mathematical calculations and questions. They can also be put in containers.

Numbers are highlighted with dark red in the code editor.

4.1.3 Strings

First an example:

```
print "Hello, I'm a string."
```

In this example `print` is a command where `"Hello, I'm a string."` is a string. Strings start and end with the `"` mark, by these marks KTurtle knows it is a string.

Strings can be put in containers. Yet they cannot be used in mathematical calculations and questions.

Strings are highlighted with red in the code editor.

4.1.4 Names

When using the TurtleScript programming language you create new things. If you write a program you will often need containers and in some cases you need learn to create new commands. When making a new command with learn you will have to specify a name.

You can choose any name, as long as it does not already have a meaning. For instance you cannot name a function forward, since that name is already used for an internal command.

```
# here forward is used as a new command,
# but it already has a meaning so
# this will produce an error:
learn forward {
  print "this is invalid"
}

# this works:
learn myforward {
  print "this is ok"
}
```

Names can contain only letters, numbers and underscores (_). Yet they have to start with a letter. Container names have to start with the container prefix ($).

```
# here forward is used as a container,
# starting with the $ prefix, so it does
# not conflict with the forward command
$forward = 20
print $forward
```

Containers are highlighted with bolded purple in the code editor.

Please read the documentation on containers and the learn command for a better explanation and more examples.

4.1.5 Assignments

Assignment are done with the = symbol. In programming languages it is better to read the single = not as 'equals' but as 'becomes'. The word 'equals' is more appropriate for the == which is a question.

Assignments are generally use for two reasons, (1) to add content containers, and (2) to modify the content of a container. For example:

```
$x = 10
# the container x now contains the number 10
$W = "My age is: "
# the container W now contains the string "My age is: "
# this prints the content of the containers 'W' and 'x' on ↩
    the canvas
print $W + $x
```

For more examples see the section that explains containers.

4.1.6 Math Symbols

KTurtle supports all basic math symbols: add (+), subtract (-), multiply (*), divide (/) and the brackets (and).

For a complete explanation and more examples see the math section.

4.1.7 Questions

We can ask simple questions on which the answer will be 'true' or 'false'.

Using questions is extensively explained in the questions section.

4.1.8 Question Glue-Words

Questions can be glued together with so called 'question glue'. The glue words are `and`, `or`, and a special glue-word: `not`.

Using question-glue is explained in the Question Glue section.

4.1.9 Comments

Comments are lines that start with a `#`. For example:

```
# this is a comment!
print "this is not a comment"
# the previous line is not a comment, but the next line is:
# print "this is not a comment"
```

We can add comments to the code for ourselves or for someone else to read. Comments are used for: (1) adding a small description to the program, (2) explaining how a piece of code works if it is a bit cryptic, and (3) to 'comment-out' lines of code that should be (temporarily) ignored (see the last line of the example).

Commented lines are highlighted with light grey in the code editor.

4.2 Commands

Using commands you tell the turtle or KTurtle to do something. Some commands need input, some give output. In this section we explain all the commands that can be used in KTurtle. Please note that all build in commands we discuss here are highlighted with blue in the code editor, this can help you to distinguish them.

4.2.1 Moving the turtle

There are several commands to move the turtle over the screen.

forward (fw)

```
forward X
```

`forward` moves the turtle forward by the amount of X pixels. When the pen is down the turtle will leave a trail. `forward` can be abbreviated to `fw`

backward (bw)

```
backward X
```

`backward` moves the turtle backward by the amount of X pixels. When the pen is down the turtle will leave a trail. `backward` can be abbreviated to `bw`.

turnleft (tl)

```
turnleft X
```

`turnleft` commands the turtle to turn an amount of X degrees to the left. `turnleft` can be abbreviated to `tl`.

turnright (tr)

```
turnright X
```

`turnright` the turtle to turn an amount of X degrees to the right. `turnright` can be abbreviated to `tr`.

direction (dir)

```
direction X
```

`direction` set the turtle's direction to an amount of X degrees counting from zero, and thus is not relative to the turtle's previous direction. `direction` can be abbreviated to `dir`.

center

```
center
```

`center` moves the turtle to the center on the canvas.

go

```
go X,Y
```

`go` commands the turtle to go to a certain place on the canvas. This place is X pixels from the left of the canvas, and Y pixels form the top of the canvas.

gox

```
gox X
```

`gox` using this command the turtle will move to X pixels from the left of the canvas whilst staying at the same height.

goy

```
goy Y
```

`gox` using this command the turtle will move to Y pixels from the top of the canvas whilst staying at the same distance from the left border of the canvas.

> **NOTE**
> Using the commands `go`, `gox`, `goy` and `center` the turtle will not draw a line, no matter if the pen is up or down.

4.2.2 Where is the turtle?

There are two commands which return the position of the turtle on the screen.

getx `getx` returns the number of pixels from the left of the canvas to the current position of the turtle.

gety `gety` returns the number of pixels from the top of the canvas to the current position of the turtle.

4.2.3 The turtle has a pen

The turtle has a pen that draws a line when the turtle moves. There are a few commands to control the pen. In this section we explain these commands.

penup (pu)

```
penup
```

`penup` lifts the pen from the canvas. When the pen is 'up' no line will be drawn when the turtle moves. See also `pendown`. `penup` can be abbreviated to `pu`.

pendown (pd)

```
pendown
```

`pendown` presses the pen down on the canvas. When the pen is press 'down' on the canvas a line will be drawn when the turtle moves. See also `penup`. `pendown` can be abbreviated to `pd`.

penwidth (pw)

```
penwidth X
```

`penwidth` sets the width of the pen (the line width) to an amount of X pixels. `penwidth` can be abbreviated to `pw`.

pencolor (pc)

```
pencolor R,G,B
```

`pencolor` sets the color of the pen. `pencolor` takes an RGB combination as input. `pencolor` can be abbreviated to `pc`.

4.2.4 Commands to control the canvas

There are several commands to control the canvas.

canvassize (cs)

```
canvassize X,Y
```

With the `canvassize` command you can set the size of the canvas. It takes X and Y as input, where X is the new canvas width in pixels, and Y is the new height of the canvas in pixels. `canvassize` can be abbreviated to `cs`.

canvascolor (cc)

```
canvascolor R,G,B
```

`canvascolor` set the color of the canvas. `canvascolor` takes an RGB combination as input. `canvascolor` can be abbreviated to `cc`.

4.2.5 Commands to clean up

There are two commands to clean up the canvas after you have made a mess.

clear (ccl)

```
clear
```

With `clear` you can clean all drawings from the canvas. All other things remain: the position and angle of the turtle, the canvascolor, the visibility of the turtle, and the canvas size.

reset

```
reset
```

`reset` cleans much more thoroughly than the `clear` command. After a `reset` command everything is like is was when you had just started KTurtle. The turtle is positioned at the middle of the screen, the canvas color is white, the turtle draws a black line on the canvas and the canvassize is set to 400 x 400 pixels.

4.2.6 The turtle is a sprite

First a brief explanation of what sprites are: sprites are small pictures that can be moved around the screen, like we often see in computer games. Our turtle is also a sprite. For more info see the glossary on sprites.

Next you will find a full overview on all commands to work with sprites.

[The current version of KTurtle does not yet support the use of sprites other than the turtle. With future versions you will be able to change the turtle into something of your own design]

spriteshow (ss)

```
spriteshow
```

`spriteshow` makes the turtle visible again after it has been hidden. `spriteshow` can be abbreviated to `ss`.

spritehide (sh)

```
spritehide
```

`spritehide` hides the turtle. This can be used if the turtle does not fit in your drawing. `spritehide` can be abbreviated to `sh`.

4.2.7 Can the turtle write?

The answer is: 'yes'. The turtle can write: it writes just about everything you command it to.

print

```
print X
```

The `print` command is used to command the turtle to write something on the canvas. `print` takes numbers and strings as input. You can `print` various numbers and strings using the '+' symbol. See here a small example:

```
$year = 2003
$author = "Cies"
print $author + " started the KTurtle project in " +  ↩
    $year + " and still enjoys working on it!"
```

fontsize

```
fontsize X
```

`fontsize` sets the size of the font that is used by `print`. `fontsize` takes one input which should be a number. The size is set in pixels.

4.2.8 A command that rolls dice for you

There is one command that rolls dice for you, it is called `random`, and it is very useful for some unexpected results.

random (rnd)

```
random X,Y
```

`random` is a command that takes input and gives output. As input are required two numbers, the first (X) sets the minimum output, the second (Y) sets the maximum. The output is a randomly chosen number that is equal or greater then the minimum and equal or smaller than the maximum. Here a small example:

```
repeat 500 {
  $x = random 1,20
  forward $x
  turnleft 10 - $x
}
```

Using the `random` command you can add a bit of chaos to your program.

4.2.9 Input and feedback though dialogs

A dialog is a small pop-up window that provides some feedback or asks for some input. KTurtle has two commands for dialogs, namely: `message` and `ask`

message

```
message X
```

The `message` command takes a string as input. It shows a pop-up dialog containing the text from the string.

```
message "Cies started the KTurtle project in 2003 and ↩
    still enjoys working on it!"
```

ask

```
ask X
```

`ask` takes a string as input. It shows a pop-up dialog containing the text from the string, just like the message. But in addition to it also puts an input field on the dialog. Through this input filed the user can enter a number or a string which can be stored in a container. For example

```
$in = ask "What is you age?"
$out = 2003 - $in
print "In 2003 you where " + $out + " years old at some ↩
    point."
```

When a user cancels the input dialog, or does not enter anything at all the container is emptied.

4.3 Containers

Containers are letters or words that can be used by the programmer to store a number or a text. Containers that contain a number are called variables, containers that can contain text are called string. Containers can be identified by the container character '$' that precedes their usage.

Containers that are not used contain nothing. An example:

```
print $N
```

This will print nothing and you get an error message.

4.3.1 Variables: number containers

Let us start with an example:

```
$x = 3
print $x
```

In the first line the letter `x` is made into a variable (number container). As you see the value of the variable `x` is set to 3. On the second line the value is printed.

Note that if we wanted to print an 'x' that we should have written

```
print "x"
```

That was easy, now a bit harder example:

```
$A = 2004
$B = 25
$C = $A + $B

# the next command prints "2029"
print $C
backward 30
# the next command prints "2004 plus 25"
print $A + " plus " + $B
backward 30
# the next command prints "1979"
print $A - $B
```

In the first two lines the variables `A` and `B` are set to 2004 and 25. On the third line the variable `C` is set to `A + B`, which is 2029. The rest of the example consists of 3 `print` commands with `backward 30` in between. The `backward 30` is there to make sure every output is on a new line. In this example you also see that variables can be used in mathematical calculations.

4.3.2 Containers that contain text (strings)

In programming code the regular text is usually started and ended with quotes. As we have already seen:

```
print "Hello programmer!"
```

The regular is delimited with quotes. These pieces of regular text we call strings.

Strings can also be stored in containers just like numbers Strings are a lot like variables. The biggest difference is that they contain text in stead of numbers. For this reason strings cannot be used in mathematical calculations and questions. An example of the use of strings:

```
$x = "Hello "
$name = ask "Please enter your name..."
print $x + $name + ", how are you?"
```

On the first line the string `x` is set to 'Hello '. On the second line the string `name` is set to the output of the `ask` command. On the third line the program prints a composition of three strings on the canvas.

This program ask you to enter your name. When you, for instance, enter the name 'Paul', the program prints 'Hello Paul, how are you?'. Please note that the plus (+) is the only math symbol that you can use with strings.

4.4 Can the Turtle do math?

Yes, KTurtle will do your math. You can add (+), subtract (-), multiply (*), and divide (/). Here is an example in which we use all of them:

```
$a = 20 - 5
$b = 15 * 2
$c = 30 / 30
$d = 1 + 1
print "a: "+$a+", b: "+$b+", c: "+$c+", d: "+$d
```

Do you know what value a, b, c and d have? Please note the use of the assignment symbol =.

If you just want a simple calculation to be done you can do something like this:

```
print 2004-12
```

Now an example with parentheses:

```
print ( ( 20 - 5 ) * 2 / 30 ) + 1
```

The expressions inside parentheses will be calculated first. In this example, 20-5 will be calculated, then multiplied by 2, divided by 30, and then 1 is added (giving 2).

KTurtle has advanced mathematical features. It knows the number `pi` and trigonometrical functions like `sin`, `cos`, `tan`, `arcsin`, `arccos`, `arctan` and the functions `sqrt` and `exp`.

4.5 Asking questions, getting answers...

if and while are execution controllers that we will discuss in the next section. In this section we use the if command to explain questions.

4.5.1 Questions

A simple example of a question:

```
$x = 6
if $x > 5 {
  print "hello"
}
```

In this example the question is the `x > 5` part. If the answer to this question is 'true' the code between the brackets will be executed. Questions are an important part of programming and often used together with execution controllers, like if. All numbers and variables (number containers) can be compared to each other with questions.

Here are all possible questions:

4.5.2 Question Glue

Question glue-words enable us to glue questions into one big question.

```
$a = 1
$b = 5
if ($a < $b) and ($b == 5) {
  print "hello"
}
```

In this example the glue-word `and` is used to glue 2 questions (`a < 5`, `b == 5`) together. If one side of the `and` would answer 'false' the whole question would answer 'false', because with the glue-word `and` both sides need to be 'true' in order to answer 'true'. Please do not forget to use the brackets around the questions!

Here is a schematic overview; a more detailed explanation follows below:

`a == b`	equals	answer is 'true' if `a` equals `b`
`a != b`	not-equals	answer is 'true' if `a` does not equal `b`
`a > b`	greater than	answer is 'true' if `a` is greater than `b`
`a < b`	smaller than	answer is 'true' if `a` is smaller than `b`
`a >= b`	greater than or equals	answer is 'true' if `a` is greater than or equals `b`
`a <= b`	smaller than or equals	answer is 'true' if `a` is smaller than or equals `b`

Table 4.2: Types of questions

`and`	Both sides need to be 'true' in order to answer 'true'
`or`	If one of the sides is 'true' the answer is 'true'
`not`	Special case: only works on one question! Changes 'true' into 'false' and 'false' into 'true'.

Table 4.4: Question glue-words

4.5.2.1 and

When two questions are glued together with `and`, both sides of the `and` have to be 'true' in order to result in 'true'. An example:

```
$a = 1
$b = 5
if (($a < 10) and ($b == 5)) and ($a < $b) {
  print "hello"
}
```

In this example you see a glued question glued onto an other question.

4.5.2.2 or

If one of the two questions that are glued together with `or` is 'true' the result will be 'true'. An example:

```
$a = 1
$b = 5
if (($a < 10) or ($b == 10)) or ($a == 0) {
  print "hello"
}
```

In this example you see a glued question glued onto an other question.

4.5.2.3 not

`not` is a special question glue-word because it only works for one question at the time. `not` changes 'true' into 'false' and 'false' into 'true'. An example:

```
$a = 1
$b = 5
if not (($a < 10) and ($b == 5)) {
  print "hello"
}
else
{
  print "not hello ;-)"
}
```

In this example the glued question is 'true' yet the `not` changes it to 'false'. So in the end `"not hello ;-)"` is printed on the canvas.

4.6 Controlling execution

The execution controllers enable you — as their name implies — to control execution.

Execution controlling commands are highlighted with dark green in a bold font type. The brackets are mostly used together with execution controllers and they are highlighted with bolded black.

4.6.1 Have the turtle wait

If you have done some programming in KTurtle you have might noticed that the turtle can be very quick at drawing. This command makes the turtle wait for a given amount of time.

wait
```
wait X
```

`wait` makes the turtle wait for X seconds.

```
repeat 36 {
  forward 5
  turnright 10
  wait 0.5
}
```

This code draws a circle, but the turtle will wait half a second after each step. This gives the impression of a slow-moving turtle.

4.6.2 Execute "if"

if
```
if question { ... }
```

The code that is placed between the brackets will only be executed `if` the answer to the question is 'true'. Please read for more information on questions in the question section.

```
$x = 6
if $x > 5 {
  print "x is greater than five!"
}
```

On the first line `x` is set to 6. On the second line the question `x > 5` is asked. Since the answer to this question is 'true' the execution controller `if` will allow the code between the brackets to be executed

4.6.3 If not, in other words: "else"

else
```
if question { ... } else { ... }
```

`else` can be used in addition to the execution controller if. The code between the brackets after `else` is only executed if the answer to the question that is asked is 'false'.

```
reset
$x = 4
if $x > 5 {
  print "x is greater than five!"
}
else
{
  print "x is smaller than six!"
}
```

The question asks if `x` is greater than 5. Since `x` is set to 4 on the first line the answer to the question is 'false'. This means the code between the brackets after `else` gets executed.

4.6.4 The "while" loop

while
```
while question { ... }
```

The execution controller `while` is a lot like if. The difference is that `while` keeps repeating (looping) the code between the brackets until the answer to the question is 'false'.

```
$x = 1
while $x < 5 {
  forward 10
  wait 1
  $x = $x + 1
}
```

On the first line `x` is set to 1. On the second line the question `x < 5` is asked. Since the answer to this question is 'true' the execution controller `while` starts executing the code between the brackets until the answer to the question is 'false'. In this case the code between the brackets will be executed 4 times, because every time the fifth line is executed `x` increases by 1.

4.6.5 The "repeat" loop

repeat

```
repeat number { ... }
```

The execution controller `repeat` is a lot like while. The difference is that `repeat` keeps repeating (looping) the code between the brackets for the given number.

4.6.6 The "for" loop, a counting loop

for

```
for start point to end point { ... }
```

The `for` loop is a 'counting loop', i.e. it keeps count for you.

```
for $x = 1 to 10 {
  print $x * 7
  forward 15
}
```

Every time the code between the brackets is executed the `x` is increased by 1, until `x` reaches the value of 10. The code between the brackets prints the `x` multiplied by 7. After this program finishes its execution you will see the times table of 7 on the canvas.
The default step size of a loop is 1, you can use an other value with

```
for start point to end point step step size { ... }
```

4.6.7 Stop the turtle

exit

```
exit
```

Finishes the execution of the code.

4.7 Create your own commands with 'learn'

`learn` is a very special command, because it is used to create your own commands. The command you create can take input and return output. Let us take a look at how a new command is created:

```
learn circle $x {
  repeat 36 {
    forward $x
    turnleft 10
  }
}
```

The new command is called `circle`. `circle` takes one input, a number, to set the size of the circle. `circle` returns no output. The `circle` command can now be used like a normal command in the rest of the code. See this example:

```
learn circle $X {
  repeat 36 {
    forward $X
    turnleft 10
  }
}

go 200,200
circle 20

go 300,200
circle 40
```

In the next example, a command with a return value is created.

```
reset

learn multiplyBySelf $n {
  $r = $n * $n
  return $r
}
$i = ask "Please enter a number and press OK"
print $i + " multiplied by itself is: " + multiplyBySelf $i
```

In this example a new command called `multiplyBySelf` is created. The input of this command is multiplied by itself and then returned, using the `return` command. The `return` command is the way to output a value from a function you have created.

Commands can have more than one input. In the next example, a command that draws a rectangle is created.

```
learn box $X, $Y {
  forward $Y
  turnright 90
  forward $X
  turnright 90
  forward $Y
  turnright 90
```

```
  forward $X
  turnright 90
}
```

Now you can run `box 50, 100` and the turtle will draw a rectangle on the canvas.

Chapter 5

Glossary

In this chapter you will find an explanation of most of the 'uncommon' words that are used in the handbook.

degrees Degrees are units to measure angles or turns. A full turn is 360 degrees, a half turn 180 degrees and a quarter turn 90 degrees. The commands `turnleft`, `turnright` and `direction` need an input in degrees.

input and output of commands Some commands take input, some commands give output, some commands take input *and* give output and some commands neither take input nor give output.

Some examples of commands that only take input are:

```
forward 50
pencolor 255,0,0
print "Hello!"
```

The `forward` command takes `50` as input. `forward` needs this input to know how many pixels it should go forward. `pencolor` takes a color as input and `print` takes a string (a piece of text) as input. Please note that the input can also be a container. The next example illustrates this:

```
$x = 50
print $x
forward 50
$str = "hello!"
print $str
```

Now some examples of commands that give output:

```
$x = ask "Please type something and press OK... thanks!"
$r = random 1,100
```

The `ask` command takes a string as input, and outputs the number or string that is entered. As you can see, the output of `ask` is stored in the container `x`. The `random` command also gives output. In this case it outputs a number between 1 and 100. The output of the random is again stored in a container, named `r`. Note that the containers `x` and `r` are not used in the example code above.

There are also commands that neither need input nor give output. Here are some examples:

```
clear
penup
```

intuitive highlighting This is a feature of KTurtle that makes coding even easier. With intuitive highlighting the code that you write gets a color that indicates what type of code it is. In the next list you will find the different types of code and the color they get in the code editor.

pixels A pixel is a dot on the screen. If you look very close you will see that the screen of your monitor uses pixels. All images on the screen are built with these pixels. A pixel is the smallest thing that can be drawn on the screen.

A lot of commands need a number of pixels as input. These commands are: `forward`, `backward`, `go`, `gox`, `goy`, `canvassize` and `penwidth`.

RGB combinations (color codes) RGB combinations are used to describe colors. The 'R' stand for 'red', the 'G' stands for 'green' and the 'B' stands for 'blue'. An example of an RGB combination is `255,0,0`: the first value ('red') is 255 and the others are 0, so this represents a bright shade of red. Each value of an RGB combination has to be in the range 0 to 255. Here a small list of some often used colors:

Two commands need an RGB combination as input: these commands are `canvascolor` and `pencolor`.

sprite A sprite is a small picture that can be moved around the screen. Our beloved turtle, for instance, is a sprite.

Note: with this version of KTurtle the sprite cannot be changed from a turtle into something else. Future versions of KTurtle will be able to do this.

regular commands	dark green	The regular commands are described here.
execution controllers	black (bold)	The special commands control execution, read more on them here.
comments	dark yellow	Lines that are commented start with a comment characters (#). These lines are ignored when the code is executed. Comments allow the programmer to explain a bit about his code or can be used to temporarily prevent a certain piece of code from executing.
brackets {, }	light green (bold)	Brackets are used to group portions of code. Brackets are often used together with execution controllers.
the learn command	light green (bold)	The learn command is used to create new commands.
numbers	blue	Numbers, well not much to say about them.
strings	dark red	Not much to say about (text) strings either, except that they always start and end with the double quotes (").
mathematical characters	grey	These are the mathematical characters: +, -, *, /, (, and). Read more about them here.
questions characters	blue (bold)	Read more about questions here.
question glue-words	pink	Read more about the question glue-words (and, or, not) here.
regular text	black	

Table 5.2: Different types of code and their highlight color

0,0,0	black
255,255,255	white
255,0,0	red
150,0,0	dark red
0,255,0	green
0,0,255	blue
0,255,255	light blue
255,0,255	pink
255,255,0	yellow

Table 5.4: Often used RGB combinations

Chapter 6

Translator's Guide to KTurtle

As you probably already know, the unique feature of the Logo programming language is that the Logo commands are often translated to the language of the programmer. This takes away a barrier for some learners to understand the basics of programming. When translating KTurtle to a new language, the commands and default examples are included in the standard .pot files used for translation in KDE.

Please look at http://edu.kde.org/kturtle/translator.php for more information about the translation process. Thanks a lot for your work!

Chapter 7

Credits and License

KTurtle

- Cies Breijs cies AT kde DOT nl
- Anne-Marie Mahfouf annma AT kde DOT org
- Some proofreading changes by Philip Rodrigues phil@kde.org
- Updated translation how-to and some proofreading changes by Andrew Coles andrew_coles AT yahoo DOT co DOT uk

Appendix A

Installation

A.1 How to obtain KTurtle

KTurtle is part of the KDE project http://www.kde.org/ .

KTurtle can be found in the kdeedu package on ftp://ftp.kde.org/pub/kde/ , the main FTP site of the KDE project.

A.2 Compilation and Installation

In order to compile and install KTurtle on your system, type the following in the base directory of the KTurtle distribution:

```
% ./configure
% make
% make install
```

Since KTurtle uses **autoconf** and **automake** you should have no trouble compiling it. Should you run into problems please report them to the KDE mailing lists.

Appendix B

Index

A
and, 26
arcsin, arccos, arctan, 24
ask, 21

B
backward (bw), 16

C
canvascolor (cc), 19
canvassize (cs), 19
center, 17
clear (ccl), 19

D
direction (dir), 17

E
else, 28
exit, 29

F
fontsize, 20
for, 29
forward (fw), 16

G
getx, 18
gety, 18
go, 17
gox, 17
goy, 17

I
if, 27

L
learn, 29

M
message, 21

N
not, 26

O
or, 26

P
pencolor (pc), 18
pendown (pd), 18
penup (pu), 18
penwidth (pw), 18
pi, 24
print, 20

R
random (rnd), 20
repeat, 29
reset, 19

S
sin, cos, tan, 24
spritehide (sh), 20
spriteshow (ss), 20
sqrt, exp, 24
step, 29

T
turnleft (tl), 16
turnright (tr), 17

W
wait, 27
while, 28