

# **The KTurtle Handbook**

**Cies Breijs and Anne-Marie Mahfouf**



# The KTurtle Handbook

# Contents

|          |                               |          |
|----------|-------------------------------|----------|
| <b>1</b> | <b>Introduction</b>           | <b>1</b> |
| 1.1      | What is Logo? . . . . .       | 1        |
| 1.2      | Features of Kturtle . . . . . | 2        |
| <b>2</b> | <b>Using Kturtle</b>          | <b>3</b> |
| 2.1      | The Code Editor . . . . .     | 4        |
| 2.2      | The Canvas . . . . .          | 4        |
| 2.3      | The Menubar . . . . .         | 4        |
| 2.3.1    | The File Menu . . . . .       | 4        |
| 2.3.1.1  | New . . . . .                 | 4        |
| 2.3.1.2  | Open . . . . .                | 5        |
| 2.3.1.3  | Open Recent . . . . .         | 5        |
| 2.3.1.4  | Open Examples . . . . .       | 5        |
| 2.3.1.5  | Save . . . . .                | 5        |
| 2.3.1.6  | Save As . . . . .             | 5        |
| 2.3.1.7  | Save Canvas . . . . .         | 5        |
| 2.3.1.8  | Execution Speed . . . . .     | 5        |
| 2.3.1.9  | Execute . . . . .             | 6        |
| 2.3.1.10 | Pause . . . . .               | 6        |
| 2.3.1.11 | Stop . . . . .                | 6        |
| 2.3.1.12 | Print . . . . .               | 6        |
| 2.3.1.13 | Quit . . . . .                | 6        |
| 2.3.2    | The Edit Menu . . . . .       | 6        |
| 2.3.3    | The View Menu . . . . .       | 7        |
| 2.3.4    | The Tools Menu . . . . .      | 7        |
| 2.3.5    | The Settings Menu . . . . .   | 8        |
| 2.3.6    | The Help Menu . . . . .       | 8        |
| 2.4      | The Toolbar . . . . .         | 9        |
| 2.5      | The Statusbar . . . . .       | 9        |

## The KTurtle Handbook

|          |   |           |
|----------|---|-----------|
| <b>3</b> | <b>Getting Started</b>                            | <b>10</b> |
| 3.1      | First steps with Logo: meet the Turtle! . . . . . | 11        |
| 3.1.1    | The Turtle Moves . . . . .                        | 11        |
| 3.1.2    | More examples . . . . .                           | 11        |
| <b>4</b> | <b>KTurtle's Logo Programming Reference</b>       | <b>14</b> |
| 4.1      | Different Instruction Types . . . . .             | 14        |
| 4.1.1    | Commands . . . . .                                | 14        |
| 4.1.2    | Numbers . . . . .                                 | 14        |
| 4.1.3    | Strings . . . . .                                 | 15        |
| 4.1.4    | Names . . . . .                                   | 15        |
| 4.1.5    | Assignments . . . . .                             | 16        |
| 4.1.6    | Math Symbols . . . . .                            | 16        |
| 4.1.7    | Questions . . . . .                               | 16        |
| 4.1.8    | Question Glue-Words . . . . .                     | 16        |
| 4.1.9    | Comments . . . . .                                | 16        |
| 4.2      | Commands . . . . .                                | 17        |
| 4.2.1    | Moving the turtle . . . . .                       | 17        |
| 4.2.1.1  | forward (fw) . . . . .                            | 17        |
| 4.2.1.2  | backward (bw) . . . . .                           | 17        |
| 4.2.1.3  | turnleft (tl) . . . . .                           | 17        |
| 4.2.1.4  | turnright (tr) . . . . .                          | 18        |
| 4.2.1.5  | direction (dir) . . . . .                         | 18        |
| 4.2.1.6  | center . . . . .                                  | 18        |
| 4.2.1.7  | go . . . . .                                      | 18        |
| 4.2.1.8  | gox . . . . .                                     | 18        |
| 4.2.1.9  | goy . . . . .                                     | 19        |
| 4.2.2    | The turtle has a pen . . . . .                    | 19        |
| 4.2.2.1  | penup (pu) . . . . .                              | 19        |
| 4.2.2.2  | pendown (pd) . . . . .                            | 19        |
| 4.2.2.3  | penwidth (pw) . . . . .                           | 19        |
| 4.2.2.4  | pencolor (pc) . . . . .                           | 20        |
| 4.2.3    | Commands to control the canvas . . . . .          | 20        |
| 4.2.3.1  | canvassize (cs) . . . . .                         | 20        |
| 4.2.3.2  | canvascolor (cc) . . . . .                        | 20        |

## The KTurtle Handbook

|          |  |           |
|----------|--|-----------|
| 4.2.3.3  | wrapon . . . . .                                 | 20        |
| 4.2.3.4  | wrapoff . . . . .                                | 20        |
| 4.2.4    | Commands to clean up . . . . .                   | 21        |
| 4.2.4.1  | clear (cr) . . . . .                             | 21        |
| 4.2.4.2  | reset . . . . .                                  | 21        |
| 4.2.5    | The turtle is a sprite . . . . .                 | 21        |
| 4.2.5.1  | show . . . . .                                   | 21        |
| 4.2.5.2  | hide (sh) . . . . .                              | 22        |
| 4.2.6    | Can the turtles write? . . . . .                 | 22        |
| 4.2.6.1  | print . . . . .                                  | 22        |
| 4.2.6.2  | fontsize . . . . .                               | 22        |
| 4.2.7    | A command that rolls dice for you . . . . .      | 22        |
| 4.2.8    | Input and feedback through dialogs . . . . .     | 23        |
| 4.2.8.1  | message . . . . .                                | 23        |
| 4.2.8.2  | inputwindow . . . . .                            | 23        |
| 4.3      | Containers . . . . .                             | 24        |
| 4.3.1    | Variables: number containers . . . . .           | 24        |
| 4.3.2    | Containers that contain text (strings) . . . . . | 25        |
| 4.4      | Can the Turtle do math? . . . . .                | 25        |
| 4.5      | Asking questions, getting answers... . . . .     | 26        |
| 4.5.1    | Questions . . . . .                              | 26        |
| 4.5.2    | Question Glue . . . . .                          | 26        |
| 4.5.2.1  | and . . . . .                                    | 27        |
| 4.5.2.2  | or . . . . .                                     | 27        |
| 4.5.2.3  | not . . . . .                                    | 28        |
| 4.6      | Controlling execution . . . . .                  | 28        |
| 4.6.1    | Have the turtle wait . . . . .                   | 28        |
| 4.6.2    | Execute "if" . . . . .                           | 29        |
| 4.6.3    | The "while" loop . . . . .                       | 29        |
| 4.6.4    | If not, in other words: "else" . . . . .         | 29        |
| 4.6.5    | The "for" loop, a counting loop . . . . .        | 30        |
| 4.7      | Create your own commands with 'learn' . . . . .  | 30        |
| <b>5</b> | <b>Glossary</b>                                  | <b>32</b> |

## The KTurtle Handbook

|          |   |           |
|----------|---|-----------|
| <b>6</b> | <b>Translator's Guide to KTurtle</b>                        | <b>36</b> |
| 6.1      | Creating a Directory to hold the Translated Files . . . . . | 36        |
| 6.2      | How To Translate the Logo Keywords (commands) . . . . .     | 36        |
| 6.3      | How To Translate the Syntax Highlighting Files . . . . .    | 37        |
| 6.4      | How To Translate the Examples . . . . .                     | 38        |
| <b>7</b> | <b>Credits and License</b>                                  | <b>39</b> |
| <b>A</b> | <b>Installation</b>   | <b>41</b> |
| A.1      | How to obtain KTurtle . . . . .                             | 41        |
| A.2      | Compilation and Installation . . . . .                      | 41        |

## List of Tables

|     |   |    |
|-----|---|----|
| 4.2 | Types of questions . . . . .                                | 27 |
| 4.4 | Question glue-words . . . . .                               | 27 |
| 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 using the Logo programming language. The unique quality of LOGO is that the programming commands are translated to the language of the 'programmer' so he/she can program in his/her native language.

# Chapter 1

## Introduction

KTurtle is an educational programming environment using the [Logo](#) programming language. 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. The commands used to program are in the style of the Logo programming language. The unique feature of the Logo programming language is that the commands are often translated 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 Logo commands, to draw a picture on [the canvas](#).

### 1.1 What is Logo?

The first version Logo programming language 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](#). This version of Logo (KTurtle) is only focused on the educational qualities of the programming language and will not try to suit professional programmers' needs.

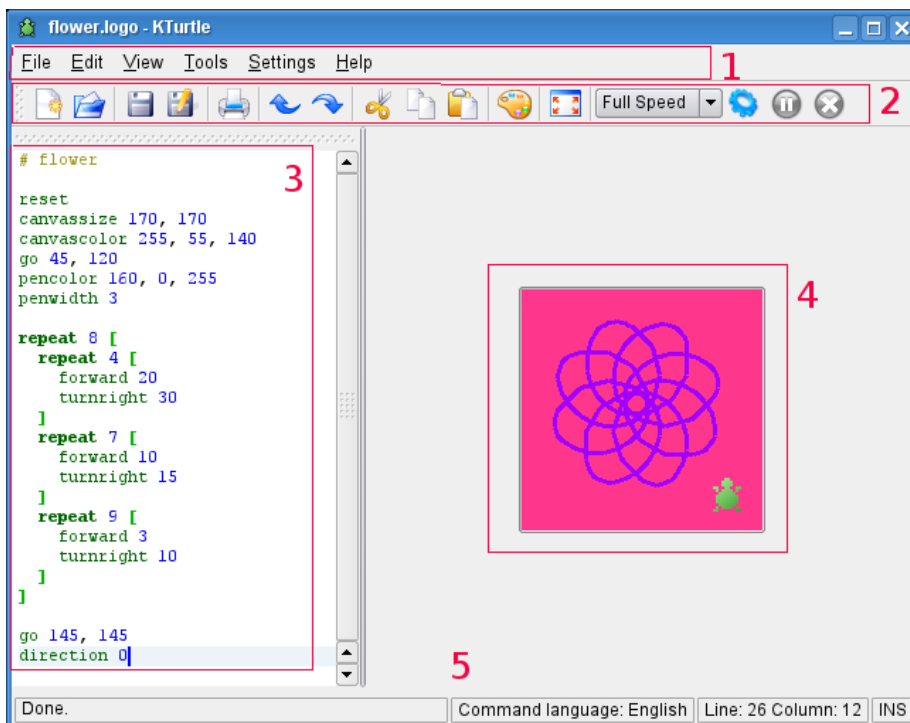
## 1.2 Features of KTurtle

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

- An integrated Logo interpreter (no extra dependencies) that uses XML files for the command translations, supports user defined functions and dynamic type switching.
- The execution can be slowed down, paused or stopped at any time.
- A powerful editor for the Logo commands with intuitive syntax highlighting, line numbering and more.
- The **canvas** can be saved as an image or printed.
- The **canvas** has a wrapping mode (which is on by default) so the turtle cannot get lost too easily.
- Context help for all Logo commands: Just press F2.
- The Logo commands are fully translatable (at the moment of writing Brazilian Portuguese, Dutch, French, German, Italian, Slovenian, Serbian (Cyrillic and Latin), Spanish and Swedish are in KDE).
- An error dialog that links the error messages to the mistakes in the program.
- Simplified programming terminology.
- Full-screen mode.
- Many integrated, internationalized example Logo programs make it easy to get started.

## Chapter 2

# Using KTurtle



The main window of KTurtle has two main parts: the [code editor](#) (3) on the left where you type the Logo commands, and the [canvas](#) (4) 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](#) (5) where you will find feedback on the state of KTurtle.

## 2.1 The Code Editor

In the code editor you type the Logo 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 → Open Examples in the [File menu](#) and click on a file. The filename will tell you what the example is about (e.g. `square.logo` will draw a square). The file you choose will be opened in the [the code editor](#), you can then use File → Execute Commands to run the code if you like.

You can open Logo 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 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 → Execute Commands, 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.

The picture that is drawn can be saved as an image (using File → Save Canvas) or printed (using File → Print...).

## 2.3 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.

### 2.3.1 The File Menu

#### 2.3.1.1 New

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

### 2.3.1.2 Open

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

### 2.3.1.3 Open Recent

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

### 2.3.1.4 Open Examples

**File** → **Open Examples (Ctrl-E)** Show the folder with examples Logo programs. The examples should be in your favorite language that you can choose in **Settings** → **Configure KTurtle...**

### 2.3.1.5 Save

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

### 2.3.1.6 Save As

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

### 2.3.1.7 Save Canvas

**File** → **Save Canvas** Saves the current drawing on canvas into an image.

### 2.3.1.8 Execution Speed

**File** → **Execution Speed** Present a list of possible execution speeds, consisting of: Full Speed, Slow, Slower and Slowest. 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.

### 2.3.1.9 Execute

**File** → **Execute Commands (Alt-Return)** Starts the execution of the commands in the code editor.

### 2.3.1.10 Pause

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

### 2.3.1.11 Stop

**File** → **Stop Execution (Escape)** Stops the execution. This action is only enabled when the commands are actually executing.

### 2.3.1.12 Print

**File** → **Print... (Ctrl-P)** Prints either the current code in the editor or the current drawing on the canvas.

### 2.3.1.13 Quit

**File** → **Quit (Ctrl-Q)** Quits KTurtle.

## 2.3.2 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** → **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** → **Replace... (Ctrl-R)** With this action you can replace phrases in the code.

### 2.3.3 The View Menu

**View** → **Full Screen Mode (Ctrl-Shift-F)** With this action you toggle the full screen mode.

Note: When code is executed while in full screen mode everything but the canvas is hidden. This makes it possible to write 'full screen' programs in KTurtle.

**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.

### 2.3.4 The Tools Menu

**Tools** → **Color Picker (Alt-C)** This action opens the color picker. Using the color picker you can easily select a color code and insert it in [the code editor](#).

**Tools** → **Indent (Ctrl-I)** This action 'indents' (adds white space at the beginning of) the lines that are selected. When 'indentation' is used properly this can make code much easier to read. All examples use indentation, please check them out.

**Tools** → **Unindent (Ctrl-Shift-I)** This action 'unindents' (removes the white space at the beginning of) the lines that are selected.

**Tools** → **Clean Indentation** This action cleans 'indentation' (removes all the white space at the beginning of) the lines that are selected.

**Tools** → **Comment (Ctrl-D)** This action add comment characters (#) in from of the lines that are selected. Lines that start with a comment character are ignored when the code is executed. Comments allow the programmer to explain a bit about his code or they can be used to temporarily prevent a certain piece of code from being executed.

**Tools** → **Uncomment (Ctrl-Shift-D)** This action removes the comment characters from the selected lines.

### 2.3.5 The Settings Menu

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

**Settings** → **Show/Hide Statusbar** Toggle the Statusbar

**Settings** → **Advanced Settings** Here you can change things you normally do not need to change. The Advanced Settings submenu has three items: **Configure Editor...** (the standard Kate editor settings dialog), **Configure Shortcuts...** (the standard KDE shortcut settings dialog), and **Configure Toolbars...** (the standard KDE toolbars setting dialog).

**Settings** → **Configure KTurtle...** This is used to configure KTurtle. Here you can change the language of the Logo commands or set a new initial canvas size.

### 2.3.6 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** → **Help on: ... (F2)** This is a very useful function: it provides help on the code where the cursor in the code editor is at. So, e.g., you have used the `print` command in your code, and you want to read and to know what the handbook says on this command. You just move your cursor so it is in the `print` command and you press F2. The handbook will then show all info on the `print` command.

This function is very important while learning programming.

**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.

## 2.4 The Toolbar

Here you can quickly reach the most used actions. By default, you will find here all main useful commands ending with the Execute Commands and Stop Execution icons.

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

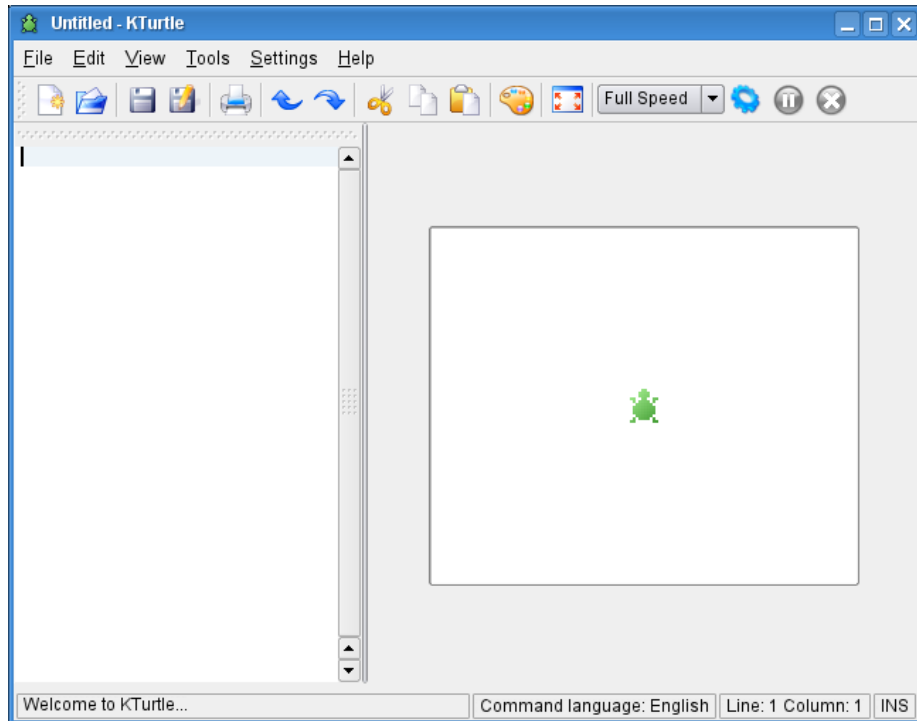
## 2.5 The Statusbar

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 Logo commands is English. You can change this language in Settings → Configure KTurtle... in the Language section. Be aware that the language you set here for KTurtle is the one you use to type the Logo commands.

## 3.1 First steps with Logo: 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 [FileExecute Commands](#)) 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 Logo commands (you can do that by choosing Settings → Configure KTurtle..., in the Language section).

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!

## The KTurtle Handbook

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

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.logo` file in the Open Examples folder and execute it (using [FileExecute Commands](#)) 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:

`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.

`clear` clear the canvas, that is all it does.

`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`

## The KTurtle Handbook

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

# KTurtle's Logo Programming Reference

This is the reference for the KTurtle's Logo. 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, LOGO 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.

## The KTurtle Handbook

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 blue 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 dark red in the [code editor](#).

### 4.1.4 Names

When using the Logo 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 [container](#) or 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 container `forward`, since that name is already used for a command, and thus has a meaning.

```
# here forward is used as a container, but it already has a ↔  
  meaning  
# so this will produce an error:  
forward = 20  
  
# this works:  
forward 20
```

Names can contain only letters, numbers and underscores (`_`). Yet they have to start with a letter.

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 dark yellow 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 dark green 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.

#### 4.2.1.1 forward (fw)

```
forward  
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`

#### 4.2.1.2 backward (bw)

```
backward  
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`.

#### 4.2.1.3 turnleft (tl)

```
turnleft  
turnleft X
```

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

#### 4.2.1.4 turnright (tr)

```
turnright  
turnright X
```

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

#### 4.2.1.5 direction (dir)

```
direction  
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.

#### 4.2.1.6 center

```
center  
center
```

center moves the turtle to the center on the canvas.

#### 4.2.1.7 go

```
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 from the top of the canvas. Note that using the go command the turtle will not draw a line.

#### 4.2.1.8 gox

```
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.

#### 4.2.1.9 goy

```
goy Y
```

goy 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.

### 4.2.2 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.

#### 4.2.2.1 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.

#### 4.2.2.2 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.

#### 4.2.2.3 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.

#### 4.2.2.4 pencolor (pc)

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

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

### 4.2.3 Commands to control the canvas

There are several commands to control the canvas.

#### 4.2.3.1 canvassize (cs)

```
canvassize  
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`.

#### 4.2.3.2 canvascolor (cc)

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

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

#### 4.2.3.3 wrapon

```
wrapon  
wrapon
```

With the `wrapon` command you can set wrapping 'on' for the canvas. Please see the glossary if you want to know what wrapping is.

#### 4.2.3.4 wrapoff

```
wrapoff  
wrapoff
```

With the `wrapoff` command you can set wrapping 'off' for the canvas: this means the turtle can move off the canvas and can get 'lost'. Please see the glossary if you want to know what wrapping is.

## 4.2.4 Commands to clean up

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

### 4.2.4.1 clear (cr)

```
clear  
clear
```

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

### 4.2.4.2 reset

```
reset  
reset
```

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

## 4.2.5 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]

### 4.2.5.1 show

```
show (ss)  
show
```

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

#### 4.2.5.2 hide (sh)

```
hide  
hide
```

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

#### 4.2.6 Can the turtles write?

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

##### 4.2.6.1 print

```
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!"
```

##### 4.2.6.2 fontsize

```
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.

#### 4.2.7 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  
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.

## 4.2.8 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 `inputwindow`

### 4.2.8.1 message

#### **message**

```
message X
```

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

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

### 4.2.8.2 inputwindow

#### **inputwindow**

```
inputwindow X
```

`inputwindow` 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 field the user can enter a [number](#) or a [string](#) which can be stored in a [container](#). For example

```
in = inputwindow "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 that are not used contain nothing. An example:

```
print N
```

This will print nothing. If we try to do [math](#) with empty containers we will get errors.

### 4.3.1 Variables: number containers

Let us start with an example:

```
x = 3
print x
```

In the first line the letter `x` 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 = inputwindow "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 inputwindow 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).

## 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:

Questions are highlighted with light blue in the [code editor](#).

### 4.5.2 Question Glue

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

```
a = 1
b = 5
if (a < 5) and (b == 5) [
    print "hello"
]
```

## The KTurtle Handbook

|                        |                        |   |
|------------------------|------------------------|---|
| <code>a == b</code>    | equals                 | answer is 'true' if a equals b                    |
| <code>a != b</code>    | not-equal              | answer is 'true' if a does not equal b            |
| <code>a &gt; b</code>  | greater than           | answer is 'true' if a is greater than b           |
| <code>a &lt; b</code>  | smaller than           | answer is 'true' if a is smaller than b           |
| <code>a &gt;= b</code> | greater than or equals | answer is 'true' if a is greater than or equals b |
| <code>a &lt;= b</code> | smaller than or equals | answer is 'true' if a is smaller than or equals b |

Table 4.2: Types of questions

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:

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

Table 4.4: Question glue-words

Question glue-words are highlighted with purple in the [code editor](#).

### 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"
]
27
```

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:

### 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 square brackets are mostly used together with execution controllers and they are highlighted with light green.

### 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 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.

## 4.6.4 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.

#### 4.6.5 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.

### 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 KTurtle Handbook

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 30,30  
circle 20  
  
go 40,40  
circle 50
```

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

```
reset  
  
learn multiplyBySelf n [  
  r = n * 1  
  r = n * n  
  return r  
]  
i = inputwindow "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.

## 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
str = "hello!"
print str
```

Now some examples of commands that give output:

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

## The KTurtle Handbook

The `inputwindow` command takes a string as input, and outputs the number or string that is entered. As you can see, the output of `inputwindow` 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
wrapon
hide
```

**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:

To easily find the RGB combinations of a color you should try the color picker! You can open the color picker using [ToolsColor Picker](#).

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.

**wrapping** Wrapping is what happens when the turtle draws something that is too big to fit in on the canvas and wrapping is set 'on'.

The KTurtle Handbook

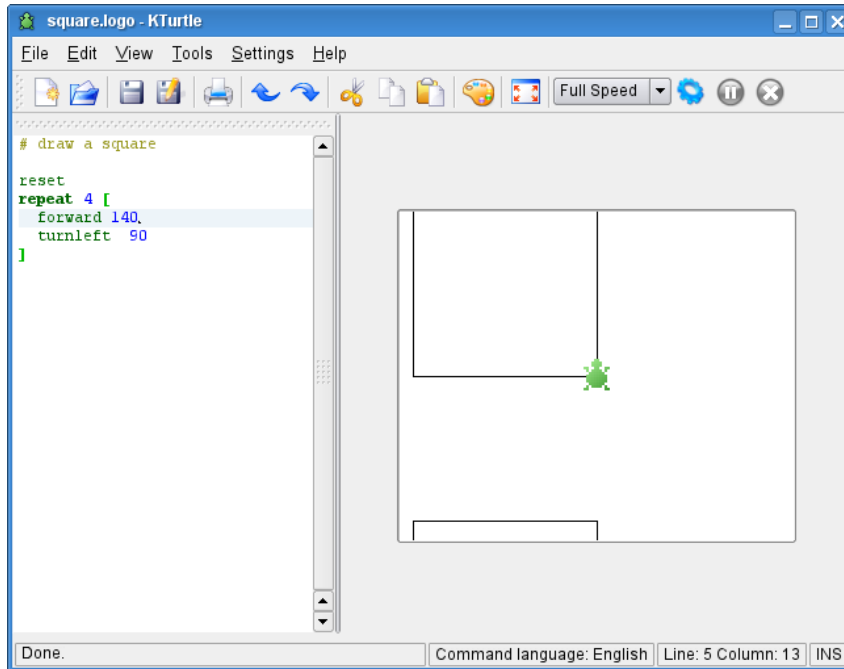
|                                   |                    |  |
|-----------------------------------|--------------------|--|
| regular commands                  | dark green         | The regular commands are described <a href="#">here</a> .  |
| execution controllers             | black (bold)       | The special commands control execution, read more on them <a href="#">here</a> .   |
| 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 <a href="#">execution controllers</a> .   |
| the <a href="#">learn</a> command | light green (bold) | The <a href="#">learn</a> 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 <a href="#">here</a> .   |
| questions characters              | blue (bold)        | Read more about questions <a href="#">here</a> .   |
| question glue-words               | pink               | Read more about the question glue-words (and, or, not) <a href="#">here</a> .  |
| regular text                      | black              |  |

Table 5.2: Different types of code and their highlight color

## The KTurtle Handbook

|             |            |
|-------------|------------|
| 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



When the turtle moves off a border of the canvas it is instantly taken to the opposite border so it can continue its move. This way the turtle will always stay on the screen while it moves. This happens when wrapping is on.

Wrapping can be turned on and off with the `wrapon` and `wrapoff` commands. When KTurtle starts wrapping is turned on by default.

## 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 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 some more files have to be translated in addition to the usual strings and documentation. Yet most it is autogenerated by Rafael Beccar's scripts. These scripts can be found in `kdeedu/kturtle/scripts`, the files that needs translation can be found in `kdeedu/kturtle/data`, in those directories you also find README files which contains instructions for using/translating them.

### 6.1 Creating a Directory to hold the Translated Files

First, you need to create a directory to store the translated files. Create a directory called `kde-i18n/code/data/kdeedu/kturtle/` in your KDE CVS directory, where `code` is your country code (the 2- or 4- letter ISO code).

Copy the `Makefile.am` file from `kdeedu/kturtle/data/` into this directory. Open it using your favorite text editor, replace all instances of `'en_US'` in the file with your country code (the one used above), and save the file.

### 6.2 How To Translate the Logo Keywords (commands)

- Copy the `logokeywords.en_US.xml` file from `kdeedu/kturtle/data/` to the directory you have just created, and rename it to `logokeywords.code.xml` where `code` is your country code (the 2- or 4- letter ISO code).
- Translate the contents of the `<keyword>` tag (i.e. the information between `<keyword>` and `</keyword>`) into your own language wherever possible.

Also, translate the contents of the `<alias>` tag, (i.e. the information between the `<alias>` and `</alias>`): these are used as shortcuts for the keyword.

For example, translate 'while' in: `<keyword>while</keyword>`

Please do not translate anything else and do not translate the English words in `<command name="english_word">`: these must stay in English.

Last bit: do not change the order of this file, this is needed for Rafael Beccar's automatic translation generation scripts.

- Save your file as UTF-8 (in Kate, use Save As... and change to utf8 in the box on the right of the file name).
- Commit your file (add your filename in the `Makefile.am`) or send it to Anne-Marie.
- In case of any doubt, please contact Anne-Marie Mahfouf [annemarie.mahfouf@free.fr](mailto:annemarie.mahfouf@free.fr) for more information.

### 6.3 How To Translate the Syntax Highlighting Files

Translating the `logohighlightstyle.en_US.xml` is a breeze when using Rafael Beccar's script in `kdeedu/kturtle/data/`. Please make sure to read the README file in that directory.

For backward compatibility with all people who do not want to the perl scripted blessing mentioned in the previous paragraph, here the old fashioned way of doing it:

- Copy the `logohighlightstyle.en_US.xml` file from `kdeedu/kturtle/data/` to the directory you created to store the translated keywords file, and rename it to `logohighlightstyle.code.xml` where `code` is your country code (the 2- or 4- letter ISO code).
- In line 4 of the file, there is `<language name="en_US">...`: here you change 'en\_US' to your language's ISO code (2 or 4 letters).
- Translate into your own language the content of the `<item>` tag (i.e. the information between `<item>` and `</item>`). This content must match the `logokeyword` file. For example, translate 'while' in: `<item> while </item>` and leave the spaces as they are (one at the beginning and one at the end). Please do not translate anything else.
- Save your file as UTF-8 (in Kate, use Save As... and change to utf8 in the box on the right of the file name).
- Commit your file (add your filename in the `Makefile.am`) or send it to Anne-Marie.
- In case of any doubt, please contact Anne-Marie Mahfouf [annemarie.mahfouf@free.fr](mailto:annemarie.mahfouf@free.fr) for more information.

## 6.4 How To Translate the Examples

Again this task is simplified a lot by Rafael Beccar's script in `kdeedu/kturtle/data/`. Please make sure to read the README file in that directory, since some work still has to be done after the example logo files are autotranslated.

When you followed the instructions, given in the README file that you found in the scripts directory, you should now be almost ready. Please do not forget to test the translated example logo code you created, since it is very common that an error sneaks in. Also make sure the `Makefile.am` in `kde-i18n/code/data/kdeedu/kturtle/` is updated according to the new files. For the Dutch the `Makefile.am` should look like this:

```
txt_DATA = advertentie.logo driehoeken.logo krullen.logo ↔
          tafels.logo \
bloem.logo driehoek.logo logohighlightstyle.nl.xml pijl.logo ↔
          vierkanten.logo \
kleuren.logo logokeywords.nl.xml randomnaam.logo vierkant. ↔
          logo
txt_dir = $(kde_datadir)/kturtle/examples/nl

xml_DATA = logohighlightstyle.nl.xml
xml_dir = $(kde_datadir)/katepart/syntax

keywords_DATA = logokeywords.nl.xml
keywords_dir = $(kde_datadir)/kturtle/data
EXTRA_DIST = $(txt_DATA) $(xml_DATA) $(keywords_DATA)
```

Here a description of how to do the translation WITHOUT the use of Rafael's perl scripts:

- Copy the English example files from `kdeedu/kturtle/data/` to the directory used to store the translated keyword and highlighting files. Translate the file-names of the examples in your directory: this will allow users to easily and quickly understand what the example is about.
- Translate the keywords in the examples, using those in the `logokeywords.xml` for your language. The keywords file must be done, first, before translating the examples.
- Save your file as UTF-8 (in Kate, use Save As... and change to utf8 in the box on the right of the file name)
- Commit your folder (add a `Makefile.am` inside) or send it to Anne-Marie.
- In case of any doubt, please contact Anne-Marie Mahfouf, [annemarie.mahfouf@free.fr](mailto:annemarie.mahfouf@free.fr) for more information.
- Finally, if you want, you can add your own examples in this folder.

## Chapter 7

# Credits and License

KTurtle

Program copyright 2003-2005 Cies Breijs [cies AT kde DOT nl](mailto:cies AT kde DOT nl)

Contributors:

- Coding help, editor part: Anne-Marie Mahfouf [annma@kde.org](mailto:annma@kde.org)
- Author of WS BASIC (<http://wsbasic.sourceforge.net>) which is the base for the interpreter of KTurtle: Walter Schreppers [Walter DOT Schreppers AT ua DOT ac DOT be](mailto:Walter DOT Schreppers AT ua DOT ac DOT be)
- German Data Files: Matthias Meßmer [bmlmessmer AT web DOT de](mailto:bmlmessmer AT web DOT de)
- German Data Files: Burkhard Lück [lueck AT hube-lueck DOT de](mailto:lueck AT hube-lueck DOT de)
- Swedish Data Files: Stefan Asserhäll [stefan DOT asserhal AT telia DOT com](mailto:stefan DOT asserhal AT telia DOT com)
- Slovenian Data Files: Jure Repinc [jlp@holodeck1.com](mailto:jlp@holodeck1.com)
- Serbian (Cyrillic and Latin) Data Files: Chusslove Illich [caslav.ilic@gmx.net](mailto:caslav.ilic@gmx.net)
- Italian Data Files: Pino Toscano [toscانو.pino@tiscali.it](mailto:toscانو.pino@tiscali.it)
- English GB Data Files: Andy Potter [A.J.Potter@rhul.ac.uk](mailto:A.J.Potter@rhul.ac.uk)
- Spanish Data Files: Rafael Beccar [rafael.beccar@kdemail.net](mailto:rafael.beccar@kdemail.net)
- Brazilian Portuguese Data Files: Riverson Rios [riverson@ccv.ufc.br](mailto:riverson@ccv.ufc.br)
- Norwegian Nynorsk and Bokmål Data Files: Karl Ove Hufthammer [karl@huftis.org](mailto:karl@huftis.org)
- Parser Cyrillic support: Albert Astals Cid [astals11@terra.es](mailto:astals11@terra.es)

Documentation copyright 2004

- Cies Breijs [cies AT kde DOT nl](mailto:cies AT kde DOT nl)

## The KTurtle Handbook

- Anne-Marie Mahfouf [annma AT kde DOT org](mailto:annma@kde.org)
- Some proofreading changes by Philip Rodrigues [phil@kde.org](mailto:phil@kde.org)
- Updated translation how-to and some proofreading changes by Andrew Coles [andrew\\_coles AT yahoo DOT co DOT uk](mailto:andrew_coles AT yahoo DOT co DOT uk)

This documentation is licensed under the terms of the [GNU Free Documentation License](#).

This program is licensed under the terms of the [GNU General Public License](#).

# 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 kdedu 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.