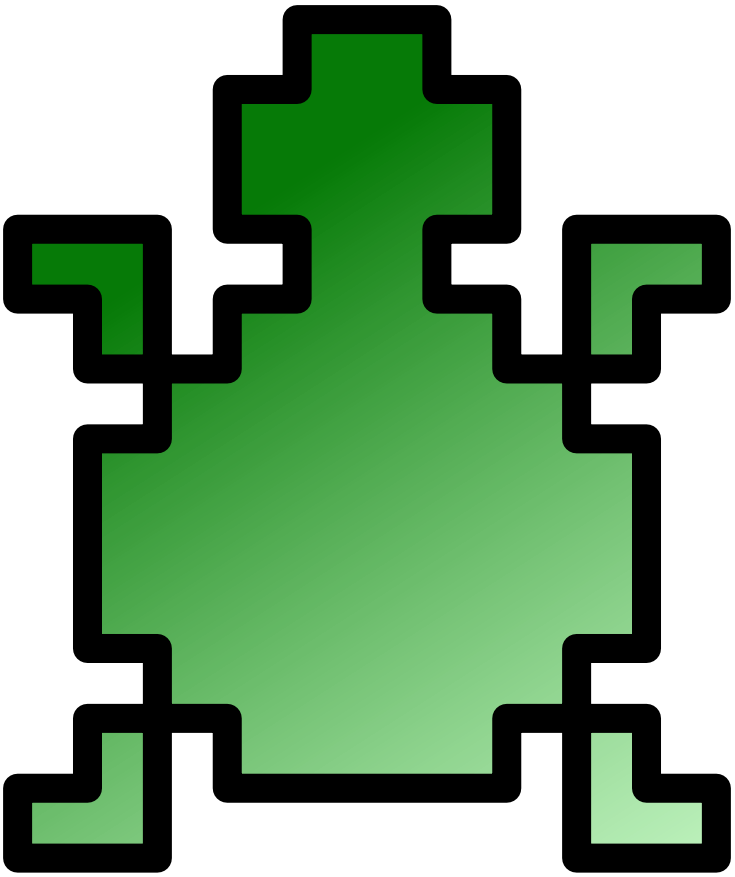
O Manual do KTurtle

Cies Breijs
Anne-Marie Mahfouf
Mauricio Piacentini
Tradução: José Pires



Conteúdo

Lista de Tabelas

Resumo

O KTurtle é um ambiente de programação educativa que permite aprender como programa da forma mais simples possível. Para conseguir isto, o KTurtle disponibiliza todas as ferramentas de programação a partir da interface de utilizador. A linguagem de programação usada é o TurtleScript, que permite a tradução dos seus comandos.

Capítulo 1

Introdução

O KTurtle é um ambiente educativo de programação utilizando a linguagem de programação TurtleScript, uma linguagem de programação ligeiramente baseada e inspirada pelo Logo. O objectivo do KTurtle é tentar manter a programação tão acessível quanto possível. Isto torna o KTurtle adequado para ensinar às crianças matemática, geometria e... programação. Uma funcionalidade única do TurtleScript é que os comandos são normalmente traduzidos para a língua falada pelo programador.

O KTurtle tem o nome com base na 'tartaruga' que desempenha um papel central no ambiente de programação. O utilizador programa a tartaruga, usando os comandos do TurtleScript, para desenhar uma imagem na área de desenho.

1.1 O que é o TurtleScript?

A TurtleScript, a linguagem de programação usada no KTurtle, é fortemente inspirada por alguns dos conceitos fundamentais da linguagem de programação Logo. A primeira versão da linguagem de programação Logo foi criada por Seymour Papert do Laboratório de Inteligência Artificial do MIT em 1967 como uma alternativa à linguagem de programação LISP. Desde então, foram lançadas várias versões do Logo. Em 1980, o Logo foi ganhando adeptos, com versões para o MSX, Commodore, Atari e sistemas IBM PC. Estas versões eram principalmente para fins educativos. A LCSI lançou o Mac®Logo em 1985 como uma ferramenta para programadores profissionais, mas nunca teve grande sucesso. O MIT ainda mantém um 'site' sobre Logo que poderá ser acedido em http://el.media.mit.edu/logo-foundation/ .

O TurtleScript partilha uma funcionalidade que é encontrada em muitas outras implementações do Logo: a capacidade de traduzir os comandos, de modo a adequar-se à língua nativa do aluno. Esta funcionalidade permite simplificar aos alunos que tenham pouco ou nenhum conhecimento de Inglês para começarem a aprender. Para além desta funcionalidade, o KTurtle tem muitas outras funcionalidades que permitem facilitar aos alunos a experiência inicial com a programação.

1.2 Funcionalidades do KTurtle

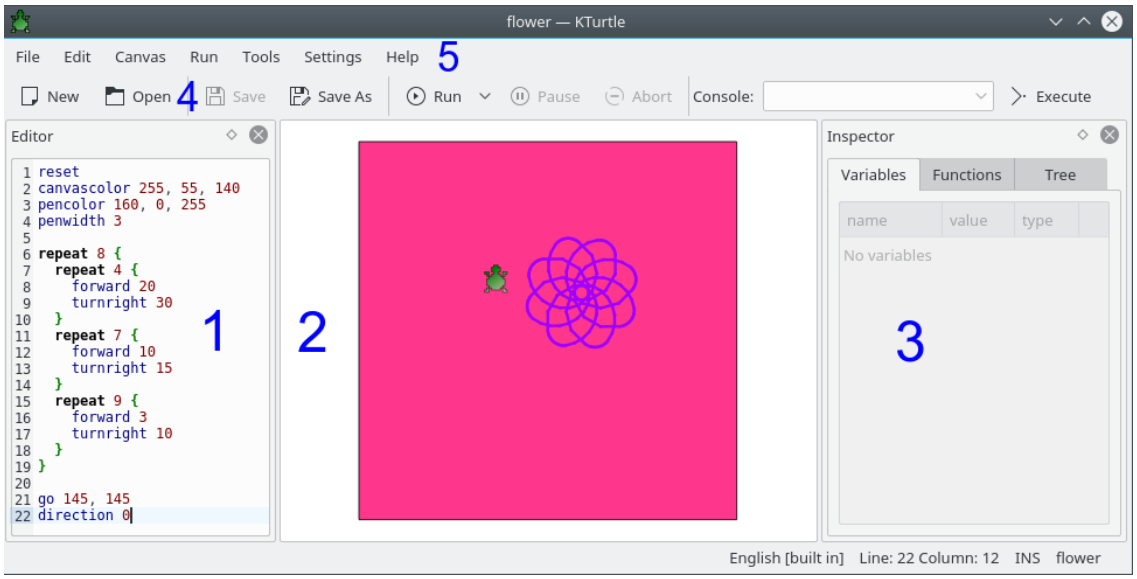
O KTurtle tem algumas funcionalidades giras que tornam a introdução à programação uma leve brisa. Veja aqui alguns dos detalhes das funcionalidades do KTurtle:

- Um ambientes integrado com o interpretador de TurtleScript , um editor, uma área de desenho e outras ferramentas, tudo numa única aplicação (sem dependências extra).
- A capacidade de traduzir os comandos do TurtleScript com a plataforma de traduções do KDE.

- O TurtleScript suporta funções definidas pelo utilizador, a recursividade e a mudança dinâmica de tipos.
- A execução pode ser tornada mais lenta, pausada ou interrompida em qualquer altura.
- Um editor poderoso com um realce de sintaxe intuitivo, com numeração de linhas, marcadores de erros, execução visual, entre outras coisas.
- A área de desenho, onde a tartaruga circula, pode ser impressa ou gravada como uma imagem (PNG) ou desenho (SVG).
- Ajuda de contexto: ajuda onde necessita dela. Basta carregar em **F2** (ou ver a opção **Ajuda** → **Ajuda sobre: ...**) para obter ajuda sobre o pedaço de código que está de momento sob o seu cursor.
- Uma janela de erros que associa as mensagens de erro aos erros propriamente ditos no programa, marcando-os a vermelho.
- Uma terminologia de programação simplificada.
- Programas de exemplo integrados que simplificam a sua introdução. Estes exemplos estão traduzidos com a plataforma de traduções do KDE.

Capítulo 2

Usar o KTurtle



A janela principal do KTurtle tem três componentes principais: o editor (1) à esquerda, onde você escreve os comandos de TurtleScript, o inspector (2) que lhe dá informações acerca das variáveis durante a execução do programa e a área de desenho (3) à direita, onde as instruções são visualizadas. Os três outros locais da janela principal são: o menu (5) onde todas as acções podem ser acedidas, a barra de ferramentas (4) que lhe permite seleccionar rapidamente as acções mais utilizadas, a **Consola**, que poderá usar para introduzir um comando numa única linha para o testar e a barra de estado (no fundo da janela) onde você irá encontrar alguma informação sobre o estado do KTurtle.

2.1 O Editor

No editor, você poderá escrever os comandos de TurtleScript. A maioria das suas funcionalidades são encontradas no menu Editar e Ficheiro. O editor pode ser acoplado a qualquer um dos lados da janela principal ou poderá ser destacado e colocado em qualquer local do seu ecrã.

Você tem várias formas de obter algum código no editor. A forma mais fácil é usar um exemplo já feito. Você escolhe o **Ficheiro** → **Exemplos** no menu Ficheiro, onde poderá carregar num ficheiro. O ficheiro que escolher será aberto no editor, você poderá ir então a **Ficheiro** → **Executar** no menu, ou **Executar** na barra de ferramentas, para correr o código se o desejar.

Você poderá abrir os ficheiros de TurtleScript se escolher **Ficheiro** → **Abrir...**.

A terceira forma é escrever directamente o seu código no editor ou copiar/colar algum código.

2.2 A Área de Desenho

A área de desenho é a área onde os comandos são visualizados, ou seja, onde estes 'desenham' uma imagem. Por outras palavras, é o espaço de recreio da tartaruga. Depois de introduzir algum código no editor e de o executar, duas coisas poderão acontecer: ou o código executa na perfeição e você poderá ver algo a mudar na área de desenho, ou você tem um erro no seu código e existirá uma mensagem que lhe dirá qual o erro que você cometeu.

Poderá ampliar e reduzir a área de desenho com a roda do seu rato.

2.3 O Inspector

O inspector informa-o das variáveis, as funções aprendidas e a árvore de código enquanto o programa está em execução.

O inspector poderá ser acoplado a cada extremo da janela principal ou poderá ser destacado e colocado em qualquer ponto do seu ecrã.

2.4 A Barra de Ferramentas

Aqui poderá aceder rapidamente às acções mais utilizadas. A barra de ferramentas também contém a **Consola**, onde poderá invocar rapidamente os comandos; isto poderá ser útil no caso de querer testar um comando sem modificar o conteúdo do Editor.

Você poderá configurar a barra de ferramentas se usar o menu **Configuração** → **Configurar as Barras de Ferramentas...**, para se ajustar às suas preferências.

2.5 A Barra de Menu

No menu, você irá encontrar todas as acções do KTurtle. Estas estão nos seguintes grupos: **Ficheiro**, **Editar**, **Área de Desenho**, **Executar**, **Ferramentas**, **Configuração** e **Ajuda**. Esta secção descreve-as a todas.

2.5.1 O Menu Ficheiro

Ficheiro → **Novo (Ctrl-N)**
: Cria um ficheiro de TurtleScript novo e vazio.

Ficheiro → **Abrir... (Ctrl-O)**
: Abre um ficheiro de TurtleScript.

Ficheiro → **Abrir um Recente**
: Abre um ficheiro de TurtleScript que foi aberto recentemente.

Ficheiro → Exemplos

Abrir programas de exemplo do TurtleScript. Os exemplos deverão estar na sua língua favorita, a qual poderá escolher em **Configuração → Linguagem de Programação**.

Ficheiro → Obter mais exemplos...

Abra a janela para **Obter Coisas Novas**, de modo a transferir novos ficheiros em TurtleScript da Internet.

Ficheiro → Gravar (Ctrl-S)

Grava o ficheiro de TurtleScript aberto de momento.

Ficheiro → Gravar Como...

Grava o ficheiro de TurtleScript aberto de momento num local à escolha.

Ficheiro → Exportar para HTML...

Exporta o conteúdo actual do Editor como um ficheiro em HTML que inclui as cores do realce.

Ficheiro → Imprimir... (Ctrl-P)

Imprime o código actual no editor.

Ficheiro → Sair (Ctrl-Q)

Sai do KTurtle.

2.5.2 O Menu Editar

Editar → Desfazer (Ctrl-Z)

Anula a última alteração ao código. O KTurtle pode fazer anulações de forma ilimitada.

Editar → Refazer (Ctrl-Shift-Z)

Volta a fazer uma alteração anulada ao código.

Editar → Cortar (Ctrl-X)

Corta o texto seleccionado do editor para a área de transferência.

Editar → Copiar (Ctrl-C)

Copia o texto seleccionado do editor para a área de transferência.

Editar → Colar (Ctrl-V)

Cola o texto na área de transferência no editor.

Editar → Seleccionar Tudo (Ctrl-A)

Selecciona todo o texto do editor.

Editar → Procurar... (Ctrl-F)

Com esta acção, você poderá procurar frases no código.

Editar → Procurar o Seguinte (F3)

Use isto para procurar a próxima ocorrência da frase que estava a pesquisar.

Editar → Procurar o Anterior (Shift-F3)

Use isto para procurar a ocorrência anterior da frase que estava a pesquisar.

Editar → Modo de Sobreposição (Ins)

Comuta entre o modo de 'inserção' e 'sobreposição'.

2.5.3 O Menu Área de Desenho

Área de Desenho → Exportar para Imagem (PNG)...

Exporta o conteúdo actual da Área de Desenho como uma imagem rasterizada no formato PNG (Portable Network Graphics).

Área de Desenho → Exportar para Desenho (SVG)...

Exporta o conteúdo actual da Área de Desenho como um desenho vectorial no formato SVG (Scalable Vector Graphics).

Área de Desenho → Imprimir a Área de Desenho...

Imprime o conteúdo actual da Área de Desenho.

2.5.4 O Menu Executar

Executar → Executar (F5)

Inicia a execução dos comandos no editor.

Executar → Pausa (F6)

Coloca a execução em pausa. Esta acção só fica activa quando os comandos estiverem de facto a ser executados.

Executar → Interromper (F7)

Pára a execução; esta acção só fica activa quando os comandos estiverem de facto a ser executados.

Executar → Velocidade da Execução

Apresenta uma lista com as velocidades de execução possíveis, consistindo em: Velocidade **Completa (sem realce nem inspecção)**, **Completa**, **Lento**, **Mais Lento**, **Lentíssimo** e **Passo-a-Passo**. Quando a velocidade é igual a **Velocidade Completa** (por omissão), poderá não conseguir ver o que se está a passar. Em alguns dos casos este comportamento poderá ser o desejado, mas noutros casos poder-se-á querer ter uma ideia da execução. No último caso, poderá querer configurar a velocidade da execução como **Lento**, **Mais Lento** e **Lentíssimo**. Quando um dos modos lentos for seleccionado, a posição actual de execução será mostrada no editor. O **Passo-a-Passo** irá executar um comando de cada vez.

2.5.5 O Menu Ferramentas

Ferramentas → Selector da Direcção...

Esta acção abre a janela de selecção da direcção.

Ferramentas → Extractor de Cores...

Esta acção abre a janela de extracção de cores.

2.5.6 O Menu Configuração

Configuração → Linguagem de Programação

Escolhe a língua para o código.

Configuração → Mostrar o Editor (Ctrl-E)

Mostra ou esconde o Editor.

Configuração → Mostrar o Inspector (Ctrl-I)

Mostra ou esconde o Inspector.

Configuração → **Mostrar os Erros**

Mostra ou esconde a página de **Erro** com uma lista dos erros resultantes da execução do código. Se esta opção estiver activa, carregue na **Área de Desenho** para ver a tartaruga de novo.

Configuração → **Mostrar os Números de Linha (F11)**

Com esta acção, você poderá mostrar os números de linha no editor. Isto poderá ser útil para procurar um erro.

Configuração → **Mostrar a Barra de Ferramentas**

Activa ou desactiva a Barra Principal

Configuração → **Mostrar a Barra de Estado**

Comutar a Barra de Estado

Configuração → **Configurar os Atalhos...**

A janela-padrão do KDE para configurar os atalhos de teclado.

Configuração → **Configurar as Barras de Ferramentas...**

A janela normal do KDE para configurar as barras de ferramentas.

2.5.7 O Menu Ajuda

Ajuda → **Manual do KTurtle (F1)**

Invoca a ajuda do KDE, aberta na documentação do KTurtle. (este documento).

Ajuda → **O que é Isto? (Shift+F1)**

Muda o cursor do rato para uma mistura de uma seta com um ponto de interrogação. Ao carregar nos itens do KTurtle irá abrir uma janela de ajuda (se existir alguma para o item em particular) que explica a função do item.

Ajuda → **Comunicar um Erro...**

Abre a janela de Relato de Erros onde pode comunicar um erro ou 'pedir' uma funcionalidade.

Ajuda → **Acerca do KTurtle**

Mostra a versão da aplicação e as informações do autor.

Ajuda → **Acerca do KDE**

Mostra a versão do KDE bem como outras informações básicas.

Ajuda → **Ajuda sobre: ... (F2)**

Esta é uma função muito útil: ela oferece ajuda sobre o código sob o cursor na janela do editor. Por isso, isto é, poderá ter usado o comando **`print`** no seu código e querer ler e aprender o que o manual diz sobre este comando. Basta mover o seu cursor para cima do **`print`** e carregar então em **F2**. O manual irá então mostrar toda a informação sobre o comando **`print`**.

Esta função poder-se-á tornar útil enquanto aprende o TurtleScript.

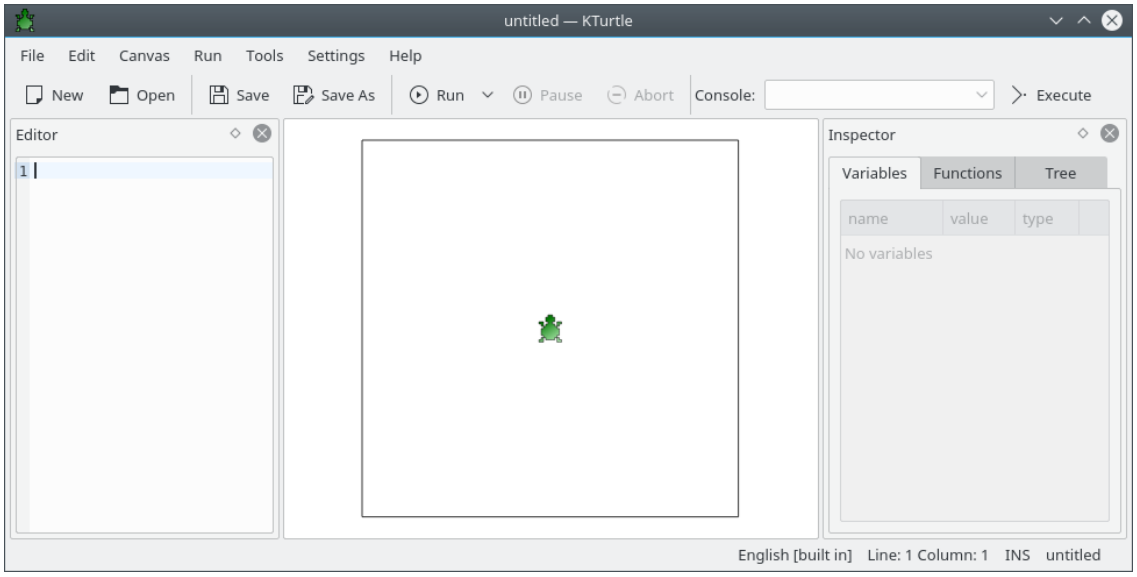
2.6 A Barra de Estado

Na barra de estado você poderá saber o estado do KTurtle. Do lado esquerdo, dá o resultado sobre a última acção. Do lado direito, você encontra a localização actual do cursor (os números da linha e da coluna). A meio da barra de estado, é indicada a língua usada para os comandos.

Capítulo 3

Começar

Quando você inicia o KTurtle você irá ver algo semelhante a isto:



Neste guia introdutório, iremos assumir que a língua dos comandos é o Inglês. Você poderá mudar esta língua em **Configuração** → **Linguagem de Programação**. Tenha em atenção que a língua que definir para o KTurtle necessita de ser a mesma que a língua que usa para escrever os comandos de TurtleScript, não a língua usada pelo seu computador para mostrar a interface e menus do KTurtle.

3.1 Primeiros passos no TurtleScript: apresentamos-lhe a Tartaruga!

Você já deve ter reparado que a tartaruga está no meio da área de desenho: você está agora prestes a aprender como controlá-la, usando os comandos no editor de código.

3.1.1 A Tartaruga Move-se

Vamos começar por pôr a tartaruga a andar. A nossa tartaruga tem 3 tipos de movimentos: (1) pode ir para a frente e para trás, (2) pode virar à esquerda ou à direita e (3) poderá ir directamente

para uma posição do ecrã. Tente isto, por exemplo:

```
forward 100
turnleft 90
```

Escreva ou copie e cole o código no editor e execute-o (usando o **Executar** → **Executar**) para ver o resultado.

Quando tiver escrito e executado os comandos acima no editor de código, você irá reparar em uma ou mais das seguintes coisas:

1. Que — depois de executar os comandos — a tartaruga vai-se movendo, desenhar uma linha e depois dá um quarto de volta para a esquerda. Isto acontece porque você usou os comandos **`forward`** e **`turnleft`**.
2. Que a cor do código mudou à medida que o foi escrevendo; esta funcionalidade chama-se *realce intuitivo* — os diferentes tipos de comandos são realçados de forma diferente. Isto torna a leitura de grandes blocos de código mais fácil.
3. Que a tartaruga desenha uma linha preta fina.
4. Talvez tenha obtido uma mensagem de erro. Isto poderá simplesmente significar duas coisas: você poderá ter cometido um erro ao copiar os comandos, ou você precisa de definir a língua correcta para os comandos de TurtleScript (o que você pode fazer escolhendo **Configuração** → **Linguagem de Programação**).

Você irá da mesma forma perceber que o **`forward 100`** mandou a tartaruga andar em frente, deixando uma linha, e que o **`turnleft 90`** disse à tartaruga virar 90 graus à esquerda.

Por favor, veja as seguintes referências para o manual para uma explicação completa dos comandos introduzidos: **`forward`**, **`backward`**, **`turnleft`** e **`turnright`**.

3.1.2 Mais exemplos

O primeiro exemplo foi muito simples, por isso vamos continuar!

```
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
```

Mais uma vez, você deverá escrever ou copiar e colar o código para o editor ou abrir o exemplo `arrow` no menu **Exemplos** e executá-lo (usando a opção **Executar** → **Executar**) para ver o resultado. Nos próximos exemplos, você já é suposto saber a mecânica do assunto.

Você poderá já ter notado que este segundo exemplo usa bastante mais código. Você poderá ter visto também um conjunto de comandos novos. Aqui está uma breve explicação de todos os comandos novos:

Depois de um comando **`reset`**, tudo fica como estava quando iniciou o KTurtle.

O **`canvassize 200,200`** configura a largura e a altura da área de desenho para 200 pontos. A largura e a altura são iguais em tamanho, o que significa que a área de desenho deverá ser agora um quadrado.

O **`canvascolor 0,0,0`** coloca a área de desenho a preto. O **`0,0,0`** é uma combinação RGB onde todos os valores ficam iguais a **`0`**, o que corresponde a preto.

O **`pencolor 255,0,0`** coloca a cor do traço a vermelho. O **`255,0,0`** é uma combinação RGB em que só o valor do 'vermelho' fica igual a **`255`** enquanto que os outros (verde e azul) ficam a **`0`**, o que resulta num tom claro de vermelho.

Se não compreender os valores das cores, tente por favor ler o glossário sobre as combinações RGB.

O **`penwidth 5`** configura a espessura (ou tamanho) do traço a **`5`** pontos. A partir de agora, todas as linhas que a tartaruga desenhar irão ter uma espessura de **`5`** pontos, até que se mude o **`penwidth`** para outra coisa qualquer.

O **`go 20,20`** manda a tartaruga ir para um dado lugar da área de desenho. A contar do canto superior esquerdo, este lugar fica a 20 pontos a contar da esquerda e a 20 pontos da parte superior. Lembre-se que, ao usar o comando **`go`**, a tartaruga não irá desenhar uma linha.

O **`direction 135`** define a direcção da tartaruga. O **`turnleft`** e o **`turnright`** mudam o ângulo da tartaruga a partir da direcção actual dela. O **`direction`** muda o ângulo da tartaruga a partir do zero e, como tal, não é relativo à posição anterior da tartaruga.

Depois do comando de direcção, segue-se um conjunto de comandos **`forward`** e **`turnleft`**. Estes comandos fazem, de facto, o desenho.

Por fim, é usado outro comando **`go`** para mover a tartaruga para o lado.

Certifique-se que segue as referências. Estas explicam cada comando em mais detalhe.

Capítulo 4

Referência de Programação de TurtleScript

Esta é a referência para o TurtleScript do KTurtle. Na primeira secção deste capítulo dê uma vista de olhos em alguns aspectos da gramática dos programas em TurtleScript. A segunda secção lida em exclusivo com os operadores matemáticos, os operadores booleanos (verdadeiro/falso) e os operadores de comparação. A terceira secção é uma lista enorme com todos os comandos, explicando-os um-a-um. A quarta secção explica como atribuir valores às variáveis. Finalmente, é explicado como organizar a execução dos comandos com as instruções de controlo da execução na quinta secção e como criar os seus próprios comandos com o aprender, na última secção.

4.1 A Gramática do TurtleScript

Como em qualquer linguagem, o TurtleScript tem diferentes tipos de palavras e símbolos. No Português, é feita a distinção entre os verbos (como 'andar' ou 'cantar') e os substantivos (como 'irmã' ou 'casa'), sendo usados para diferentes fins. O TurtleScript é uma linguagem de programação, sendo usada para indicar ao KTurtle o que fazer.

Nesta secção, alguns dos diferentes tipos de palavras e símbolos do TurtleScript são explicados de forma breve. Serão explicados os comentários, os comandos e os três diferentes tipos de literais: números, cadeias de caracteres e os valores booleanos (verdadeiro/falso).

4.1.1 Comentários

Um programa contém instruções que são executadas quando o programa é executado e outros itens, chamados de comentários. Os comentários não são executados; o KTurtle simplesmente ignora-os ao executar o seu programa. Os comentários estão lá para que os outros programadores possam compreender melhor o seu programa. Tudo o que estiver a seguir a um símbolo **#** é considerado um comentário no TurtleScript. Por exemplo, este pequeno programa não faz nada:

```
# este pequeno programa não faz nada, é apenas um comentário!
```

É um pouco inútil, mas explica bem o conceito.

Os comentários tornam-se bastante úteis quando o programa se torna um pouco mais complexo. Podê-lo-á ajudar a dar alguns conselhos aos outros programas. No seguinte programa, irá ver os comentários a serem usados em conjunto com o comando imprimir.

```
# este programa foi feito pelo Cies Breijs.
imprimir "este texto será impresso na área de desenho"
# a linha anterior não é um comentário, mas a seguinte é:
# imprimir "este texto não será impresso!"
```

A primeira linha descreve o programa. A segunda é executada pelo KTurtle e imprime a mensagem **este texto será impresso na área de desenho** na área de desenho propriamente dita. A terceira linha é um comentário. A quarta linha é um comentário que contém um pedaço de TurtleScript, se for retirado o símbolo **#** da quarta linha, a instrução 'imprimir' será executada pelo KTurtle. Os programadores dizem: a instrução da quarta linha está 'comentada'.

As linhas comentadas ficam realçadas com um cinzento claro no editor.

4.1.2 Comandos

Ao usar os comandos, você diz à tartaruga ou ao KTurtle para fazerem algo. Alguns comandos necessitam de dados de entrada, enquanto outros geram resultados.

```
# o 'avancar' é um comando que necessita de dados de entrada, neste caso do ↩
    número 100:
avancar 100
```

A primeira linha é um comentário. A segunda linha contém o comando **avancar** e o número **100**. O número não faz parte do comando, mas é considerada uma 'entrada' para o comando.

Alguns comandos, como isto é o **go** necessitam de mais que um parâmetro. Os vários parâmetros, nesse caso, deverão ser separados pelo carácter **,** (vírgula).

Para uma ideia geral de todos os comandos que o KTurtle suporta, venha aqui. Os comandos incorporados são realçados a azul-escuro

4.1.3 Números

O mais provável é que você já conheça alguma coisa sobre os números. A forma como estes são usados no KTurtle não é muito diferente da língua falada ou da matemática.

Temos os chamados números naturais: **0**, **1**, **2**, **3**, **4**, **5**, etc. Os números negativos: **-1**, **-2**, **-3**, etc. E os números decimais, por exemplo: **0,1**, **3,14**, **33,3333**, **-5,05**, **-1,0**. O carácter **,** (vírgula) é usado como separador decimal.

Os números poderão ser usados nos operadores matemáticos e nos operadores de comparação. Também poderão ser guardados em variáveis. Os números são realçados a vermelho escuro.

4.1.4 Cadeias de caracteres

Primeiro um exemplo:

```
print "Olá, sou uma cadeia de caracteres."
```

Neste exemplo, o **print** é um comando, enquanto o **´´Olá, sou uma cadeia de caracteres.´´** é, de facto, uma cadeia de caracteres. Estas começam e terminam com o símbolo **´´**; com estes símbolos, o KTurtle sabe que é uma cadeia de caracteres.

As cadeias de caracteres poderão ser colocadas em variáveis, como acontece com os números. Contudo, ao contrário dos números, as cadeias de caracteres não podem ser usadas nos operadores matemáticos ou nos operadores de comparação. As cadeias de caracteres são realçadas a vermelho.

4.1.5 Valores booleanos (verdadeiro/falso)

Existem apenas dois valores booleanos: **verdadeiro** e **falso**. Algumas vezes também são chamados de: ligado e desligado, um e zero. Contudo, no TurtleScript são sempre chamados de **verdadeiro** e **falso**. Dê uma vista de olhos neste excerto de TurtleScript:

```
$a = verdadeiro
```

Se olhar para o inspector, poderá ver que a variável **$a** está igual a **verdadeiro**, tendo um tipo booleano.

Normalmente, os valores booleanos são o resultado de um operador de comparação, como acontece no seguinte excerto de TurtleScript:

```
$resposta = 10 > 3
```

A variável **$resposta** é configurada como **verdadeiro**, dado que **10** é maior que **3**.

Os valores booleanos, **true** (verdadeiro) e **false** (falso), estão realçados a vermelho escuro.

4.2 Operadores matemáticos, booleanos e de comparação

O título deste secção poderá soar muito complicado, mas não é tão difícil de compreender como parece.

4.2.1 Operadores matemáticos

Estes são os símbolos matemáticos básicos conhecidos: a adição (**+**), a subtracção (**−**), a multiplicação (*****), a divisão (**/**) e a potência (**^**).

Aqui está um pequeno exemplo dos operadores matemáticos que poderá usar no TurtleScript:

```
$somar      = 1 + 1
$subtrair = 20 - 5
$multiplicar = 15 * 2
$dividir    = 30 / 30
$elevar     = 2 ^ 2
```

Os valores que resultam das operações matemáticas serão atribuídos às diversas variáveis. Se utilizar o inspector, poderá ver os valores.

Se você somente queria fazer um cálculo simples, você poderá fazer algo semelhante a isto:

```
imprimir 2010-12
```

Agora, um exemplo com parêntesis:

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

O que estiver entre parêntesis será calculado em primeiro lugar. Neste exemplo, o 20-5 será calculado, depois será multiplicado por 2, dividido por 30 e depois é adicionado 1 (o que dá 2). Os parêntesis também poderão ser usados noutros casos.

O KTurtle também tem mais funcionalidades sob a forma de comandos. Dê uma vista de olhos sobre os seguintes comandos, mas tenha em mente que diz respeito a operações avançadas: round, random, sqrt, pi, sin, cos, tan, arcsin, arccos, arctan.

4.2.2 Operadores booleanos (verdadeiro/falso)

Enquanto os operadores matemáticos são usados principalmente para os números, os operadores booleanos são para os valores booleanos (**verdadeiro** e **falso**). Existem apenas três operadores booleanos, o **e**, **ou** e **nao**. O seguinte exemplo de TurtleScript mostra como usá-los:

```
$e_1_1 = verdadeiro e verdadeiro    # -> verdadeiro
$e_1_0 = verdadeiro e falso   # -> falso
$e_0_1 = falso e verdadeiro   # -> falso
$e_0_0 = falso e falso  # -> falso

$ou_1_1 = verdadeiro ou verdadeiro    # -> verdadeiro
$ou_1_0 = verdadeiro ou falso   # -> verdadeiro
$ou_0_1 = falso ou verdadeiro   # -> verdadeiro
$ou_0_0 = falso ou falso  # -> falso

$nao_1 = nao verdadeiro   # -> falso
$nao_0 = nao falso  # -> verdadeiro
```

Se usar o inspector, poderá ver os valores; contudo, fornecemos estes resultados como pequenos comentários no fim das linhas. O **e** é avaliado como **verdadeiro** apenas se ambos os lados forem **verdadeiro**. O **ou** é avaliado como **verdadeiro** se qualquer um dos lados for **verdadeiro**. Finalmente, o **nao** transforma um **verdadeiro** num **falso** e um **falso** num **verdadeiro**.
Os operadores booleanos são realçados a rosa.

4.2.2.1 Alguns exemplos mais avançados

Veja o exemplo seguinte com o **e**:

```
$a = 1
$b = 5
se (($a < 10) e ($b == 5)) e ($a < $b) {
  imprimir "olá"
}
```

Neste excerto de TurtleScript, o resultado dos três operadores de comparação é reunido com os operadores **e**. Isto significa que todos os três têm de ser avaliados como ˝verdadeiro˝ para imprimir o ˝olá˝.

Um exemplo com o **ou**:

```
$n = 1
se ($n < 10) ou ($n == 2) {
  imprimir "olá"
}
```

Neste pedaço de TurtleScript, o lado esquerdo do **ou** é avaliado como 'verdadeiro', enquanto o direito é avaliado como 'falso'. Dado que um dos lados do operador **ou** é 'verdadeiro', o operador **ou** é avaliado como 'verdadeiro'. Isto significa que o 'olá' é impresso.

Finalmente, um exemplo com o **não**, que muda o 'verdadeiro' para 'falso' e o 'falso' para 'verdadeiro'. Dê uma vista de olhos:

```
$n = 1
se nao ($n == 3) {
  imprimir "olá"
} senao {
  imprimir "não olá ;-)"
}
```

4.2.3 Operadores de comparação

Considere esta simples comparação:

```
$resposta = 10 > 3
```

Aqui o **10** é comparado com o **3**, através do operador 'maior que'. O resultado desta comparação, o valor booleano **verdadeiro** é guardado na variável **$resposta**.

Todos os números e variáveis (que contenham números) poderão ser comparados entre si, com os operadores de comparação.

Aqui estão todos os possíveis operadores de comparação:

A == B	é igual	a resposta é 'verdadeira' se o **A** for igual a **B**
A != B	é diferente	a resposta é 'verdadeira' se o **A** não for igual ao **B**
A > **B**	maior que	a resposta é 'verdadeira' se o **A** for maior que o **B**
A < **B**	menor que	a resposta é 'verdadeira' se **A** for menor que **B**
A >= B	maior ou igual a	a resposta é 'verdadeira' se **A** for maior ou igual ao **B**
A <= B	menor ou igual a	a resposta é 'verdadeira' se **A** for menor ou igual a **B**

Tabela 4.1: Tipos de perguntas

Lembre-se que o A e o B têm de ser números ou variáveis que contenham números.

4.3 Comandos

Ao usar os comandos, você diz à tartaruga ou ao KTurtle para fazer algo. Alguns comandos precisam de dados introduzidos, enquanto outros trazem resultados. Nesta secção iremos explicar todos os comandos incorporados que podem ser usados no KTurtle. Em alternativa, usando o aprender, poderá criar os seus próprios comandos. Os comandos incorporados ficam realçados a azul escuro.

4.3.1 Mover a tartaruga

Existem vários comandos para mover a tartaruga pelo ecrã.

forward (fw)

```
forward X
```

O **forward** move a tartaruga para a frente X pixels. Quando o traço está em baixo, a tartaruga irá deixar um rasto. O **forward** pode ser abreviado para **fw**

backward (bw)

```
backward X
```

O **backward** move a tartaruga para trás X pixels. Quando o traço está em baixo, a tartaruga irá deixar um rasto. O **backward** pode ser abreviado para **bw**.

turnleft (tl)

```
turnleft X
```

O **turnleft** diz à tartaruga para se virar X graus para a esquerda. O **turnleft** pode ser abreviado para **tl**.

turnright (tr)

```
turnright X
```

O **turnright** diz à tartaruga para se virar X graus para a direita. O **turnright** pode ser abreviado para **tr**.

direction (dir)

```
direction X
```

O **direction** configura a direcção da tartaruga para um ângulo de X graus a contar do zero, e isto não é relativo à direcção anterior da tartaruga. O **direction** pode ser abreviado para **dir**.

getdirection

```
getdirection
```

O **getdirection** devolve a direcção da tartaruga como um ângulo de X graus a contar do zero, onde o zero é a direcção em que a tartaruga aponta para cima.

center

```
center
```

O **center** move a tartaruga para o centro da área de desenho.

go

```
go X,Y
```

O **go** manda a tartaruga ir para um dado local da área de desenho. Este local está a X pixels do lado esquerdo da área de desenho e a Y pixels do topo da área.

gox

```
gox X
```

Ao usar o comando **gox**, a tartaruga irá mover-se X pixels a partir da esquerda da área de desenho, mantendo à mesma a sua altura.

goy

```
goy Y
```

Ao usar o comando **goy**, a tartaruga irá mover-se Y pixels a partir do topo da área de desenho, mantendo à mesma a sua distância ao lado esquerdo da área de desenho.

NOTA
Usando os comandos **go**, **gox**, **goy** e **center**, a tartaruga não irá desenhar a linha, não interessando se a caneta está activa ou não.

4.3.2 Onde está a tartaruga?

Existem dois comandos que devolvem a posição da tartaruga no ecrã.

getx

O **getx** devolve o número de pixels da esquerda da área de desenho até à posição actual da tartaruga.

gety

O **gety** devolve o número de pixels de cima da área de desenho até à posição actual da tartaruga.

4.3.3 A tartaruga tem um traço

A tartaruga tem um traço e vai desenhando uma linha à medida que a tartaruga se move. Existem alguns comandos para controlar o traço. Nesta secção iremos explicar estes comandos.

penup (pu)

```
penup
```

O **penup** levanta o traço da área de desenho. Quando o traço está 'em cima', não é desenhada nenhuma linha à medida que a tartaruga se move. Veja também o **pendown**. O **penup** pode ser abreviado para **pu**.

pendown (pd)

```
pendown
```

O **pendown** carrega no traço para baixo na área de desenho. Quando o traço está 'em baixo', é desenhada uma linha à medida que a tartaruga se move. Veja também o **penup**. O **pendown** pode ser abreviado para **pd**.

penwidth (pw)

```
penwidth X
```

O **penwidth** configura a espessura do traço para X pixels. O **penwidth** pode ser abreviado para **pw**.

pencolor (pc)

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

O **pencolor** configura a cor do traço. O **pencolor** recebe uma combinação de RGB como parâmetro. O **pencolor** pode ser abreviado para **pc**.

4.3.4 Comandos para controlar a área de desenho

Existem vários comandos para controlar a área de desenho.

canvassize (cs)

```
canvassize X,Y
```

Com o comando **canvassize**, poderá definir o tamanho da área de desenho. Recebe um X e um Y como entrada, onde o o X é a nova largura da área de desenho em pixels e o Y é a nova altura da área de desenho em pixels. O **canvassize** pode ser abreviado para **cs**.

canvascolor (cc)

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

O **canvascolor** configura a cor do traço. O **canvascolor** recebe uma combinação de RGB como parâmetro. O **canvascolor** pode ser abreviado para **cc**.

4.3.5 Comandos para limpar

Existem dois comandos para limpar a área de desenho, depois de você ter deixado tudo confuso.

clear (ccl)

```
clear
```

Com o **clear**, você poderá limpar todos os desenhos da área respectiva. Tudo o resto permanece igual: a posição e o ângulo da tartaruga, a cor da área de trabalho, a visibilidade da tartaruga e o tamanho da área de desenho.

reset

```
reset
```

O **reset** limpa tudo de forma mais abrangente que o comando **clear**. Depois de um comando **reset**, tudo fica tal e qual estava quando você iniciou o KTurtle. A tartaruga é posicionada no meio do ecrã, a cor da área de desenho é branca e a tartaruga irá desenhar uma linha preta na área de desenho, que fica com um tamanho de 400 x 400 0 pixels.

4.3.6 A tartaruga é uma imagem móvel

Muitas das pessoas não sabem o que são as imagens móveis ('sprites'), daí uma breve explicação: as imagens móveis são pequenas imagens que podem percorrer o ecrã (para mais informações, veja o glossário sobre as imagens móveis).

A seguir você irá encontrar uma apresentação completa de todos os comandos que lidam com imagens móveis.

[A versão actual do KTurtle não suporta ainda o uso de imagens móveis que não apenas a tartaruga. Nas versões futuras, você poderá mudar a tartaruga para outra coisa ao seu gosto]

spriteshow (ss)

```
spriteshow
```

O **spriteshow** torna a tartaruga visível de novo depois de ter ficado escondida. O **spriteshow** pode ser abreviado para **ss**.

spritehide (sh)

```
spritehide
```

O **spritehide** esconde a tartaruga. Isto pode ser usado se a tartaruga não couber no seu desenho. O **spritehide** pode ser abreviado para **sh**.

4.3.7 Será que a tartaruga escreve?

A resposta é: 'sim'. A tartaruga sabe escrever e pode escrever tudo o que lhe disser para escrever.

print

```
print X
```

O comando **print** é usado para dizer à tartaruga para escrever algo na área de desenho. O **print** recebe números e texto como parâmetros. Você poderá executar o **print** para vários parâmetros com o sinal '+'. Veja aqui um pequeno exemplo:

```
$ano = 2004
$autor = "Ze"
print "O " + $autor + " iniciou o projecto do KTurtle em " + $ano + " e ↩
     ainda continua a gostar de trabalhar nele!"
```

fontsize

```
fontsize X
```

O **fontsize** configura o tamanho da letra que é usado pelo **print**. O **fontsize** recebe um parâmetros que deverá ser um número. O tamanho é definido em pixels.

4.3.8 Comandos matemáticos

Os seguintes comandos são as instruções matemáticas mais avançadas do KTurtle.

round

```
round(x)
```

O **round** arredonda o número indicado ao inteiro mais próximo.

```
imprimir round(10.8)
avancar 20
imprimir round(10.3)
avancar 20
```

Com este código, a tartaruga iria apresentar os números 11 e 10.

random (rnd)

```
random X,Y
```

O **random** é um comando que recebe parâmetros e devolve resultados. Como parâmetros são necessários dois números, em que o primeiro define o resultado mínimo (X) e o segundo o máximo (Y). O resultado é um número escolhido aleatoriamente que é maior ou igual ao mínimo e menor ou igual ao máximo. Aqui está um pequeno exemplo:

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

Com o comando **random**, você poderá adicionar um pouco de confusão ao seu programa.

mod

```
mod X,Y
```

O comando **mod** devolve o resto da divisão do primeiro número pelo segundo.

sqrt

```
sqrt X
```

O comando **sqrt** é usado para descobrir a raiz quadrada de um número X.

pi

```
pi
```

Este comando devolve a constante Pi, **3,14159**.

sin, cos, tan

```
sin X
cos X
tan X
```

Estes três comandos representam as conhecidas funções trigonométricas **sin** (seno), **cos** (coseno) e **tan** (tangente). O argumento de entrada destes comandos, o X, é um número.

arcsin, arccos, arctan

```
arcsin X
arccos X
arctan X
```

Estes comandos são as funções inversas do sin, cos e tan. O argumento de entrada destes comandos, o X, é um número.

4.3.9 Entrada de dados e reacção através de janelas

Uma janela poderá pedir alguma alguma reacção em especial ou a introdução de determinados dados. O KTurtle tem dois comandos para janelas, nomeadamente o **message** e o **ask**

message

```
message X
```

O comando **message** recebe uma cadeia de caracteres à entrada. Mostra então uma janela que contém o texto da cadeia de caracteres.

```
mensagem "O Ze iniciou o projecto do KTurtle em 2003 e ainda continua a ↩
    gostar de trabalhar nele!"
```

ask

```
ask X
```

O **ask** recebe uma cadeia de caracteres à entrada. Mostra uma janela que contém o texto da cadeia de caracteres, tal como acontece no message. Contudo, para além disso, também mostra um campo de texto na janela. Através deste campo, o utilizador poderá introduzir um número ou uma cadeia de caracteres que poderá ser guardada numa variável ou passado como argumento a um comando. Por exemplo

```
$entrada = perguntar "Que idade tem?"
$saida = 2003 - $entrada
imprimir "Em 2003, você tinha " + $saida + " anos a dada altura."
```

Quando um utilizador cancelar a janela ou não introduzir nada de todo, a variável fica vazia.

4.4 Atribuição de variáveis

Vejamos primeiro as variáveis, e depois iremos ver como atribuir valores a essas variáveis.

As variáveis são palavras que começam por um '$'; no editor, são realçadas a púrpura.

As variáveis poderão conter qualquer número, texto ou valor booleano (verdadeiro/falso). Ao usar a atribuição **=**, uma variável ficará com conteúdo associado. Irá manter esse conteúdo até que o programa termine a execução ou até a variável ser atribuída de novo a outra coisa qualquer.

Poderá usar as variáveis, uma vez atribuídas, como se fossem o seu próprio conteúdo. Por exemplo, no seguinte excerto de TurtleScript:

```
$x = 10
$x = $x / 3
imprimir $x
```

Primeiro, é atribuído à variável **$x** o valor **10**. Depois, o **$x** terá como novo valor o seu próprio valor dividido por **3** — isto significa na prática que o **$x** terá atribuído o valor do produto **10 / 3**. Finalmente, é impresso o valor do **$x**. Nas linhas 2 e 3, irá ver que o **$x** é usado como se fosse o seu próprio conteúdo.

As variáveis têm de estar atribuídas para poderem ser usadas. Por exemplo, um:

```
imprimir $N
```

Isto não irá imprimir nada e irá obter uma mensagem de erro.

Considere o pequeno excerto de código em TurtleScript:

```
$a = 2004
$b = 25

# o próximo comando imprime "2029"
imprimir $a + $b
recuar 30
# o próximo comando imprime "2004 mais 25 é igual a 2029"
imprimir $a + " mais " + $b + " é igual a " + ($a + $b)
recuar 30
```

Nas duas primeiras linhas, as variáveis **a** e **b** são configuradas como sendo iguais a 2004 e 25. Depois nos dois comandos **imprimir** existe um comando **recuar 30** no meio. Os comentários antes dos comando **imprimir** explicam o que está a fazer. Como poderá ver, as variáveis poderão ser usadas como se fossem o valor que contêm, podendo usá-las com qualquer tipo de operadores ou fornecendo-os como entradas na invocação dos comandos.

Mais um exemplo:

```
$nome = perguntar "Como te chamas?"
imprimir "Olá " + $nome + "! Boa sorte a aprender a arte da programação..."
```

É bastante simples. Mais uma vez, poderá ver que a variável **$nome**, é tratada apenas como texto.

Ao usar as variáveis, o inspector torna-se bastante útil. Mostra-lhe o conteúdo de todas as variáveis que estão a ser usadas de momento.

4.5 Controlar a execução

Os controladores de execução permitem-lhe — como o nome deles indica — controlar a execução.

Os comandos de controlo da execução ficam realçados a verde escuro e a negrito. Os parêntesis rectos, que são mais usados em conjunto com os controladores de execução, ficam realçados a preto e em negrito.

4.5.1 Fazer a tartaruga esperar

Se já tentou programar um pouco no KTurtle, você já poderá ter reparado que a tartaruga pode ser bastante rápida a desenhar. Este comando faz a tartaruga andar um pouco mais devagar.

wait

```
wait X
```

O **wait** faz a tartaruga esperar X segundos.

```
repeat 36 [
  forward 5
  turnright 10
  wait 0.5
]
```

Este código irá desenhar uma circunferência, mas a tartaruga irá esperar meio segundo a cada passo. Isto dá a noção de uma tartaruga vagarosa.

4.5.2 Executar o ˝if˝

if

```
se booleano [ ... ]
```

O código que é colocado entre os parêntesis só será executado **se** o valor booleano for 'verdadeiro'.

```
$x = 6
se $x > 5 [
  imprimir "O x é maior que cinco!"
]
```

Na primeira linha, o **$x** é inicializado a 6. Na segunda linha, a operador de comparação é usado para avaliar **x** > **5**. Dado que a resposta a esta pergunta é 'verdadeira', o controlador de execução **se** irá permitir que o código entre chavetas seja executado.

4.5.3 Se não, por outras palavras: ˝else˝

else

```
se booleano { ... } senao { ... }
```

O **senao** pode ser usado para além do controlador de execução **se**. O código entre chavetas a seguir ao **senao** só é executado se o valor booleano for 'falso'.

```
reset
$x = 4
se $x > 5 {
  imprimir "O x é maior que cinco!"
} senao {
  imprimir "O x é menor que seis!"
}
```

O operador de comparação testa a expressão **x** > **5**. Dado que o **x** fica igual a 4 na primeira linha, a resposta à pergunta é 'falso'. Isto significa que o código entre chavetas a seguir ao **senao** é executado.

4.5.4 O ciclo ˝while˝

while

```
enquanto booleano { ... }
```

O controlador de execução **while** é um pouco como o **se**. A diferença é que o **while** continua a repetir o código entre parêntesis até que a resposta à valor booleano seja 'falso'.

```
$x = 1
enquanto $x < 5 {
  avancar 10
  esperar 1
  $x = $x + 1
}
```

Na primeira linha, o **$x** fica igual a 1. Na segunda, a expressão **x** < **5** é avaliada. Dado que a resposta a esta pergunta é 'verdadeiro', o controlador de execução **enquanto** começa a execução do código entre chavetas até que a condição **$x** < **5** seja 'falso'. Neste caso, o código entre parêntesis será executado 4 vezes, dado que, de cada vez que a quinta linha é executada, o **$x** fica um número acima.

4.5.5 O ciclo ˝repeat˝

repeat

```
repetir número { ... }
```

O controlador de execução **repetir** funciona um pouco como o **enquanto**. A diferença é que o **repetir** continua a repetir (em ciclo) o código entre parêntesis para o número indicado.

4.5.6 O ciclo ˝for˝, um ciclo de contagem

for

```
para variável = número até número { ... }
```

O ciclo **para** é um 'ciclo de contagem', isto é, faz de contador para si. O primeiro membro configura a variável com o valor do primeiro ciclo. Em cada iteração, o número é aumentado até atingir o segundo número.

```
para $x = 1 ate 10 {
  imprimir $x * 7
  avancar 15
}
```

De cada vez que o código entre chavetas é executado, o **$x** é incrementado de uma unidade, até que o valor do **x** chegue a 10. O código entre chavetas imprime o valor de **$x** multiplicado por 7. Depois de este programa terminar a sua execução, você irá ver a tabuada dos 7 na área de desenho.
O tamanho por omissão do passo de um ciclo é 1; poderá usar outro valor qualquer com

```
para variável = número até number passo número { ... }
```

4.5.7 Sair de um ciclo

break

```
break
```

Termina imediatamente o ciclo actual e transfere o controlo para a instrução imediatamente a seguir a esse ciclo.

4.5.8 Pára a a execução do seu programa

exit

```
exit
```

Termina a execução do seu programa.

4.6 Crie os seus próprios comandos com o 'learn'

O **aprender** é um comando muito especial, porque é usado para criar os seus próprios comandos. O comando que criar poderá receber parâmetros e devolver resultados. Vamos ver como é que é criado um novo comando.

```
aprender circunferencia $x {
  repetir 36 {
    avancar $x
    esquerda 10
  }
}
```

O novo comando chama-se **circunferencia**. O **circunferencia** recebe um parâmetro, um número, para definir o tamanho da circunferência. O **circunferencia** não devolve nenhum resultado. O comando **circunferencia** pode agora ser usado como um comando normal. Veja este exemplo:

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

go 30,30
circunferencia 20

go 40,40
circunferencia 50
```

No próximo exemplo, vai ser criado um comando com um valor de resultado devolvido.

```
aprender faculdade $x {
  $r = 1
  para $i = 1 ate $x {
    $r = $n * $i
  }
  devolver $r
}

imprimir faculdade 5
```

Neste exemplo, existe agora um comando novo chamado **faculdade**. Se o parâmetro deste comando for **5**, então o resultado é igual a **5*4*3*2*1**. Ao usar o **devolver**, o valor do resultado é indicado e a execução é devolvida.

Os comandos poderão ter mais de uma entrada. No exemplo seguinte, é criado um comando que desenha um rectângulo.

```
learn caixa $x, $y {
  avancar $y
  direita 90
  avancar $x
  direita 90
  avancar $y
  direita 90
  avancar $x
  direita 90
}
```

Agora, poderá usar o **caixa 50, 100** para que a tartaruga desenhe um rectângulo na área de desenho.

Capítulo 5

Glossário

Neste capítulo, você irá obter uma explicação para a maioria das palavras 'pouco comuns' que são usadas no manual.

graus

Os graus são uma unidade para medir ângulos ou voltas. Uma volta completa corresponde a 360 graus, o que corresponde a uma meia-volta como 180 graus e um quarto-de-volta como 90 graus. Os comandos **`turnleft`**, **`turnright`** e **`direction`** necessitam de um parâmetro em graus.

parâmetros e resultado dos comandos

Alguns comandos recebem parâmetros, outros devolvem resultados, outros fazem *ambas* as coisas e finalmente existem outros que não fazem nenhuma delas.

Alguns exemplos de comandos que só recebem parâmetros são:

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

O comando **`forward`** recebe o **`50`** como parâmetro, porque o **`forward`** precisa deste parâmetro para saber quantos pontos deverá andar em frente. O **`pencolor`** recebe um parâmetro e o **`print`** recebe uma cadeia de caracteres com parâmetro. Lembre-se que o parâmetro também poderá ser um contentor. O próximo exemplo ilustra isto:

```
$x = 50
print $x
forward 50
$texto = "olá!"
print $texto
```

Agora alguns exemplos de comandos que devolvam resultados:

```
$x = inputwindow "por favor escreva algo e carregue em OK... obrigado!"
$r = random 1,100
```

O comando **`inputwindow`** recebe um texto como parâmetro e devolve o número ou o texto que é introduzido. Como poderá ver, o resultado do **`inputwindow`** é guardado no contentor **`x`**. O comando **`random`** também devolve um resultado. Neste caso, devolve um número entre 1 e 100. O resultado do **`random`** é de novo guardado num contentor, chamado **`r`**. Lembre-se que os contentores **`x`** e **`r`** não são usados no código de exemplo acima.

Também há alguns comandos que não precisam de parâmetros nem devolvem nada. Alguns exemplos:

```
clear
penup
```

realce intuitivo

Esta é uma funcionalidade do KTurtle que torna a codificação ainda mais simples. Com o realce intuitivo, o código que você escrever ganha uma cor que indica qual o tipo de código que é. Na próxima lista, você irá encontrar os diferentes tipos de código e a cor que obtêm no editor.

comandos normais	azul escuro	Os comandos normais estão descritos aqui.
Comandos de controlo da execução	preto (negrito)	Estes comandos especiais controlam a execução; poderá ler mais sobre eles aqui.
comentários	cinzento	As linhas que estão comentadas começam por caracteres de comentário (#); estas linhas são ignoradas quando o código é executado. Os comentários permitem ao programador explicar um pouco do seu código ou podem ser usadas para evitar temporariamente que um pedaço de código seja executado.
chavetas {, }	verde escuro (negrito)	Os parêntesis rectos são usados para agrupar pedaços de código. Os parêntesis rectos são usados normalmente com os controladores de execução.
o comando learn	verde claro (negrito)	O comando learn é usado para criar comandos novos.
texto	vermelho	Também não há muito a dizer sobre o texto nas cadeias de caracteres, a não ser que começam e terminam com aspas (˝).
números	vermelho escuro	Os números não têm muito que se lhe diga.
valores booleanos	vermelho escuro	Existem exactamente dois valores booleanos: verdadeiro e falso.
variáveis	púrpura	Começa com um '$' e poderá conter números, textos ou valores booleanos.

operadores matemáticos	cinzento	Estes são os operadores matemáticos: **+**, **−**, *****, **/** e **^**.
operadores de comparação	azul claro (negrito)	Estes são os operadores de comparação: **==**, **!=**, **<**, **>**, **<=** e **>=**.
operadores booleanos	rosa (negrito)	Estes são os operadores booleanos: **and**, **or** e **not**.
texto normal	preto	

Tabela 5.1: Os diferentes tipos de código e a sua cor de realce

pontos

Um ponto é um ponto no ecrã. Se você olhar muito de perto para o que vê no ecrã do seu monitor, irá constatar que ele usa pontos. Todas as imagens do ecrã são criadas com estes pontos. Um ponto é a menor coisa que poderá ser desenhada no ecrã.

Existem vários comandos que precisam de uma quantidade de pontos como parâmetro, e são: o **forward**, **backward**, **go**, **gox**, **goy**, **canvassize** e o **penwidth**.

Nas versões anteriores do KTurtle, a área de desenho era uma imagem rasterizada; agora, nas versões mais recentes, é um desenho vectorial. Isto significa que a área de desenho poderá ser ampliada ou reduzida, pelo que um ponto na área de desenho não corresponderá necessariamente a um ponto no ecrã.

Combinações de RGB (códigos de cores)

As combinações de RGB são usadas para descrever cores. O 'R' vem de 'red' (vermelho), o 'G' de 'green' (verde) e o 'B' de 'blue' (azul). Um exemplo de uma combinação RGB é o **255,0,0**, onde o valor da componente vermelha é 255 e as outras são 0, o que resulta num tom claro de vermelho. Cada valor de uma combinação RGB terá de estar no intervalo entre 0 e 255. Aqui está uma lista com as cores mais usadas:

0,0,0	preto
255,255,255	branco
255,0,0	vermelho
150,0,0	vermelho escuro
0,255,0	verde
0,0,255	azul
0,255,255	azul claro
255,0,255	cor de rosa
255,255,0	amarelo

Tabela 5.2: Combinações RGB mais usadas

Dois comandos necessitam de uma combinação RGB como parâmetro, e são eles: o **canvascolor** e o **pencolor**.

imagem móvel

Uma imagem móvel é uma pequena imagem que pode ser movida pelo ecrã. A nossa tartaruga é uma imagem móvel, por exemplo.

Nota: com esta versão do KTurtle, a imagem móvel não consegue ser alterada de uma tartaruga para outra coisa. As versões futuras do KTurtle serão capazes de o fazer.

Capítulo 6

Guia do Tradutor do KTurtle

Como já deve saber, a linguagem de programação do KTurtle - o TurtleScript -, permite a sua própria tradução. Isto retira uma barreira para alguns alunos, especialmente os mais novos, no seu esforço para compreender as bases da programação.

Ao traduzir o KTurtle para uma nova língua, irá encontrar incluídos, para além dos textos da interface, os comandos de programação, os exemplos e as mensagens de erro nos ficheiros .pot normais que são usados nas traduções do KDE. Tudo é traduzido com o método normal de traduções que está implementado no KDE, se bem que é altamente aconselhável aprender um pouco sobre como traduzi-los (o que também irá ler nos comentários do tradutor).

Por favor veja em http://edu.kde.org/kturtle/translator.php mais informações sobre o processo de traduções. Muito obrigado pelo seu trabalho! O KTurtle depende em grande medida das suas traduções.

Capítulo 7

Créditos e Licença

Apêndice A

Instalação

A.1 Como obter o KTurtle

O KTurtle faz parte do projecto do KDE http://www.kde.org/ .

O KTurtle pode ser encontrado no pacote kdeedu em ftp://ftp.kde.org/pub/kde/ , o servidor principal do projecto do KDE.

A.2 Compilação e Instalação

Para poder compilar e instalar o KTurtle no seu sistema escreva o seguinte na pasta de base da distribuição do KTurtle:

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

Dado que o KTurtle usa o **autoconf** e o **automake** não deve ter quaisquer problemas a compilá-lo. Se tiver, comunique-os para as listas do KDE.

Apêndice B

Índice