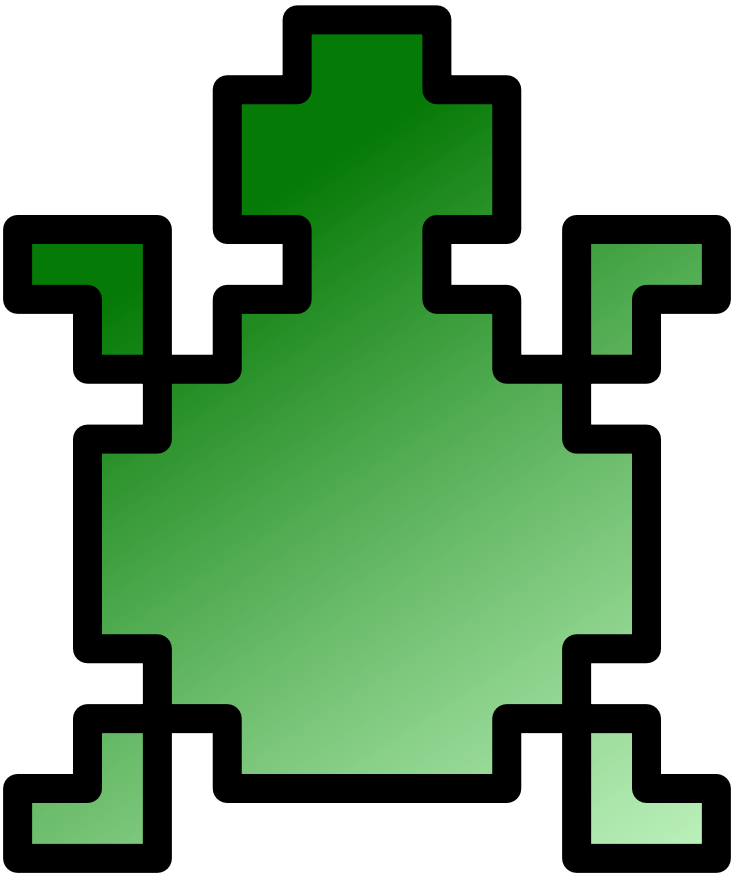
Руководство пользователя KTurtle

Cies Breijs
Anne-Marie Mahfouf
Mauricio Piacentini
Перевод на русский: Владимир Давыдов
Редакция перевода: Николай Шафоростов
Редакция перевода: Александр Поташев
Редакция перевода: Артём Золочевский



Руководство пользователя KTurtle

Оглавление

Список таблиц

Аннотация

KTurtle — это образовательная программная оболочка, которая стремится сделать процесс обучения программированию настолько простым, насколько это возможно. Для этого в KTurtle доступ ко всем инструментам программирования осуществляется из пользовательского интерфейса. В качестве языка программирования используется TurtleScript, все команды которого могут быть переведены на любой язык.

Глава 1

Введение

KTurtle — это образовательная программная оболочка, использующая язык программирования TurtleScript, основанный на языке Logo. Целью KTurtle является сделать программирование настолько лёгким и доступным, насколько это возможно. Это делает KTurtle подходящим инструментом для обучения детей основам математики, геометрии и... программирования. Одной из отличительных черт TurtleScript является возможность перевода его команд на родной разговорный язык программиста.

KTurtle назван в честь «Черепашки», которая является центральным персонажем программной оболочки. Чтобы создать рисунок на холсте ученик управляет передвижениями Черепашки, используя команды языка TurtleScript.

1.1 Что такое TurtleScript?

TurtleScript — язык программирования, использующийся в KTurtle, основывается на семействе языков Logo. Первая версия языка программирования Logo была создана Сеймуром Пейпертом (Seymour Papert) в Лаборатории Искусственного Интеллекта Массачусетского Технологического Института в 1967 году как ответвление языка программирования LISP. Впоследствии в свет вышло множество версий Logo. К 1980 году Logo становится очень популярным, активно используются его версии для MSX, Commodore, Atari, Apple II и IBM PC компьютеров — главным образом, в образовательных целях. Массачусетский Технологический Институт до сих пор поддерживает сайт, посвящённый Logo, где содержится список популярных реализаций этого языка.

TurtleScript обладает особенностью, которую можно найти и во многих других реализациях Logo: возможностью перевода команд на родной язык ученика. Эта особенность облегчает начало работы ученикам, не знающим, либо недостаточно знающим английский язык. Кроме этой особенности KTurtle обладает множеством других возможностей, облегчающих ученикам их первые шаги в программировании.

1.2 Возможности KTurtle

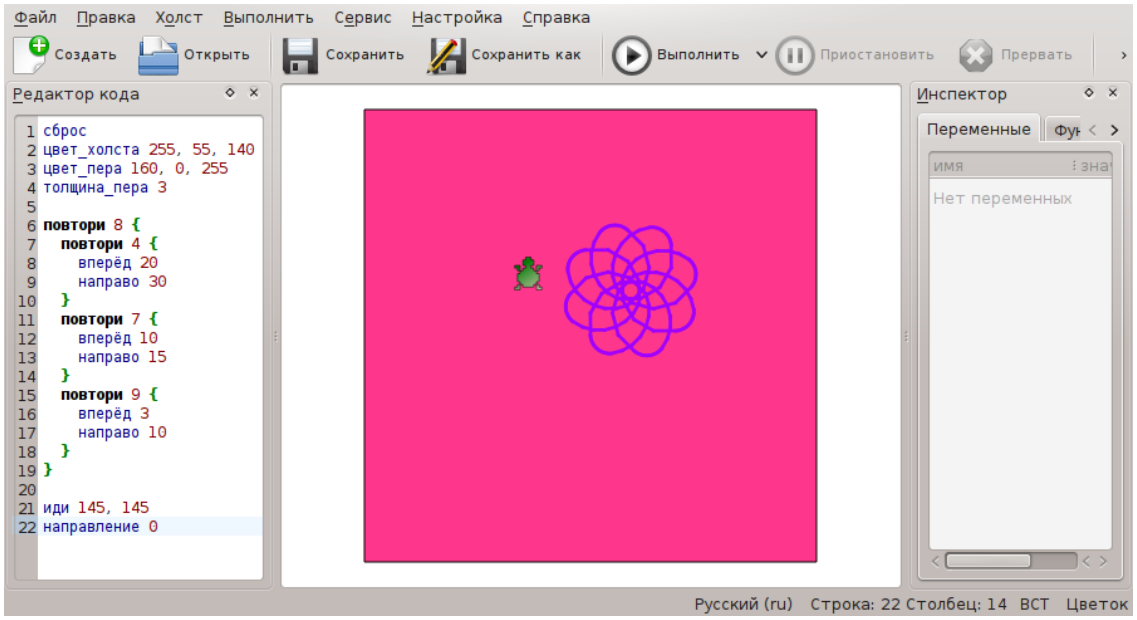
KTurtle обладает замечательными особенностями, которые позволят начать программировать легко и непринуждённо. Вот небольшой список возможностей KTurtle:

- Интегрированная среда с интерпретатором TurtleScript, редактором, холстом и другими инструментами внутри одной программы (без дополнительных зависимостей);
- Возможность перевода команд TurtleScript, используя инструменты перевода KDE;
- TurtleScript поддерживает пользовательские функции, рекурсию и динамическую типизацию;

- Выполнение можно замедлить, приостановить и прервать в любое время;
- Мощный редактор с интуитивной подсветкой синтаксиса, нумерацией строк, маркером ошибок, визуальным контролем выполнения и многим другим;
- Холст, на котором рисует Черепашка, может быть распечатан или сохранён как изображение в растровом (PNG) или векторном (SVG) формате;
- Контекстная справка: помощь там, где вы в ней нуждаетесь. Просто нажмите **F2** (или выберите **Справка** → **Справка по ...**) для получения справки по находящемуся под курсором фрагменту кода;
- Диалоговое окно с сообщениями об ошибках, указывающее на ошибки в программе и подсвечивающее их красным цветом;
- Технология упрощённого программирования;
- Встроенные примеры программ, облегчающие начало работы. Эти примеры переведены с использованием инструментов перевода KDE.

Глава 2

Использование KTurtle



Главное окно KTurtle имеет три основные области: редактор (1), расположенный слева, который предназначен для ввода команд TurtleScript, холст (2) , расположенный справа, на котором Черепашка рисует, и инспектор (3), который предоставляет вам информацию во время выполнения программы. Кроме этого имеется меню (5), предоставляющее доступ ко всем функциям программы, панель инструментов (4), которая предоставляет быстрый доступ к часто используемым командам, поле **команда**, которое можно использовать для тестирования команд, и строка состояния (вдоль нижнего края окна), на которой отображается различная информация о состоянии KTurtle.

2.1 Редактор

Редактор предназначен для ввода команд TurtleScript. Большинство команд вызывается из меню Файл и Правка. Редактор кода может быть пристыкован к любой границе главного окна или же может быть расположен в любом месте рабочего стола как отдельное окно.

Существуют разные пути получения кода в редакторе. Простейший — использовать готовый пример. Для этого необходимо вызвать пункт **Файл** → **Примеры** в меню Файл и выбрать необходимый пример. Выбранный файл примера будет открыт в редакторе и его можно будет запустить на выполнение, вызвав пункт **Выполнить** → **Выполнить** из меню или нажав кнопку **Выполнить** на панели инструментов.

Вы можете открыть файлы TurtleScript, вызвав меню **Файл** → **Открыть...**.

Третий путь — ввод кода в редакторе вручную или методом копирования/вставки.

2.2 Холст

Холст — это пространство Черепашки. Здесь она рисует в соответствии с командами, которые получает. После ввода каких-либо команд в редакторе и их выполнения могут произойти две вещи: или программа выполнится успешно, и вы увидите некоторые изменения на холсте, или же, если были допущены ошибки, вы получите сообщение во вкладке ошибок, которое уведомит вас о характере допущенной ошибки.

Вы можете увеличить или уменьшить отображение холста используя колесо мыши.

2.3 Инспектор

Инспектор информирует вас о переменных, выученных функциях и отображает иерархическую структуру программы во время её выполнения.

Инспектор может быть пристыкован к любой границе главного окна или же может быть расположен в любом месте рабочего стола как отдельное окно.

2.4 Панель инструментов

Здесь вы можете быстро получить доступ к наиболее часто используемым действиям. Панель также содержит поле для ввода **Команда**, где вы можете выполнять команды, что полезно, когда вы хотите протестировать команду, не изменяя содержимого редактора.

Вы можете настроить панель инструментов в соответствии с своими предпочтениями, воспользовавшись меню **Настройка** → **Панели инструментов...**.

2.5 Меню

В главном меню вы найдёте все команды KTurtle. Они находятся в следующих группах: **Файл**, **Правка**, **Холст**, **Выполнить**, **Сервис**, **Настройка** и **Справка**. Данный раздел посвящён их описанию.

2.5.1 Меню Файл

Файл → **Создать (Ctrl-N)**

Создаёт новый, пустой файл TurtleScript.

Файл → **Открыть... (Ctrl-O)**

Открывает файл TurtleScript.

Файл → **Последние файлы**

Открывает файлы TurtleScript, которые открывались недавно.

Файл → Примеры

Показывает примеры программ на TurtleScript. Примеры открываются на языке, выбранном в **Настройка → Язык команд**.

Файл → Загрузить новые примеры...

Открывает диалоговое окно **Загрузить новые примеры** для загрузки дополнительных файлов TurtleScript из интернета.

Файл → Сохранить (Ctrl-S)

Сохраняет текущий открытый файл TurtleScript.

Файл → Сохранить как...

Сохраняет текущий открытый файл TurtleScript в выбранном месте.

Файл → Экспорт кода в HTML...

Экспортирует текущее содержимое редактора в файл HTML с подсветкой цветами.

Файл → Печать... (Ctrl-P)

Печатает текущий код, введённый в редакторе.

Файл → Закрыть (Ctrl-Q)

Выход из KTurtle.

2.5.2 Меню Правка

Правка → Отменить действие (Ctrl-Z)

Отменяет последние изменения в коде. KTurtle имеет неограниченное число отмен!

Правка → Повторить (Ctrl-Shift-Z)

Возвращает отменённые изменения в коде.

Правка → Вырезать (Ctrl-X)

Вырезает выделенный текст из редактора в буфер обмена.

Правка → Копировать (Ctrl-C)

Копирует выделенный текст из редактора в буфер обмена.

Правка → Вставить (Ctrl-V)

Вставляет текст из буфера обмена в редактор.

Правка → Выделить всё (Ctrl-A)

Копирует весь текст из редактора в буфер обмена.

Правка → Найти... (Ctrl-F)

Ищет выражения в тексте.

Правка → Продолжить поиск (F3)

Ищет следующее вхождение фразы в тексте.

Правка → Найти предыдущее (Shift-F3)

Ищет предыдущее вхождение фразы в тексте.

Правка → Режим замены (Ins)

Переключает между режимами «вставки» и «замены».

2.5.3 Меню Холст

Холст → Экспорт в изображение (PNG)...

Экспортирует текущее состояние холста в растровое изображение PNG (Portable Network Graphics).

Холст → Экспорт в рисунок (SVG)...

Экспортирует текущее состояние холста в векторный рисунок SVG (Scalable Vector Graphics).

Холст → Печать холста...

Печатает текущее содержимое холста.

2.5.4 Меню Выполнить

Выполнить → Выполнить (F5)

Запускает на выполнение команды, введённые в редакторе.

Выполнить → Приостановить (F6)

Приостанавливает выполнение.

Выполнить → Прервать (F7)

Прекращает выполнение команд. Этот пункт доступен только тогда, когда выполняются команды.

Выполнить → Скорость выполнения

Выводит список возможных скоростей выполнения, состоящий из: **С максимальной скоростью (без подсветки и инспектора)**, **С обычной скоростью**, **Замедленно**, **Медленно**, **Очень медленно** и **Шаг за шагом**. Если скорость установлена как **С обычной скоростью** (по умолчанию), то бывает трудно уследить за тем, что происходит. Иногда это то, что нужно, но порой хочется отследить выполнение программы. В этом случае вы можете установить скорость в **Замедленно**, **Медленно** или **Очень медленно**. Когда выбрана одна из медленных скоростей выполнения в редакторе показывается текущее положение выполнения. **Шаг за шагом** выполняет по одной команде за раз.

2.5.5 Меню Сервис

Сервис → Выбор направления...

Открывает диалоговое окно выбора направления.

Сервис → Выбор цвета...

Открывает диалоговое окно выбора цвета.

2.5.6 Меню Настройка

Настройка → Язык команд

Выбирает язык команд для кода.

Настройка → Редактор кода (Ctrl-E)

Показывает или скрывает редактор.

Настройка → Инспектор (Ctrl-I)

Показывает или скрывает инспектор.

Настройка → Сообщения об ошибках

Показывает или скрывает вкладку **Ошибки** со списком всех ошибок, возникших при выполнении кода. Если этот параметр включён, щёлкните **холст**, чтобы снова увидеть Черепашку.

Настройка → Номера строк (F11)

Включает показ нумерации строк в редакторе. Это может быть полезно при поиске ошибок.

Настройка → Показать панель инструментов

Управляет отображением панели инструментов.

Настройка → Показать строку состояния

Управляет отображением статусной строки.

Настройка → Комбинации клавиш...

Стандартное диалоговое окно KDE для настройки клавиатурных комбинаций.

Настройка → Панели инструментов...

Стандартное диалоговое окно KDE для настройки панелей инструментов.

2.5.7 Меню Справка

Справка → Руководство пользователя KTurtle (F1)

Запускает Центр справки KDE и открывает справочное руководство по KTurtle (этот документ вы и читаете).

Справка → Что это? (Shift+F1)

Вид курсора меняется на стрелку со знаком вопроса. Нажимая на различные элементы внутри KTurtle, вы увидите подсказку с объяснениями назначения и действия данного элемента (если она для него существует).

Справка → Сообщить об ошибке...

Открывается диалог отправки сообщения об ошибке. Он служит для того, чтобы вы могли отправить сведения об обнаруженной вами ошибке в программе или своё «пожелание», чтобы какая-либо пока что отсутствующая возможность была реализована в будущих версиях программы.

Справка → Сменить язык интерфейса приложения...

Открывает диалоговое окно, с помощью которого вы сможете изменить **Основной язык** и **Резервный язык** интерфейса программы.

Справка → О программе KTurtle

Будут выведены сведения о версии и авторах программы.

Справка → О KDE

Будут выведены сведения о версии KDE.

Справка → Справка по ... (F2)

Очень полезная функция — вызывает помощь по тому элементу кода, рядом с которым находится курсор в редакторе. Например, вы используете команду `напиши` в своей программе и хотите узнать побольше о ней из руководства. Вам достаточно переместить курсор, чтобы он оказался на команде `напиши` и нажать **F2**. Руководство откроется на странице с описанием команды `напиши`.

Эта функция может оказаться полезной при изучении TurtleScript.

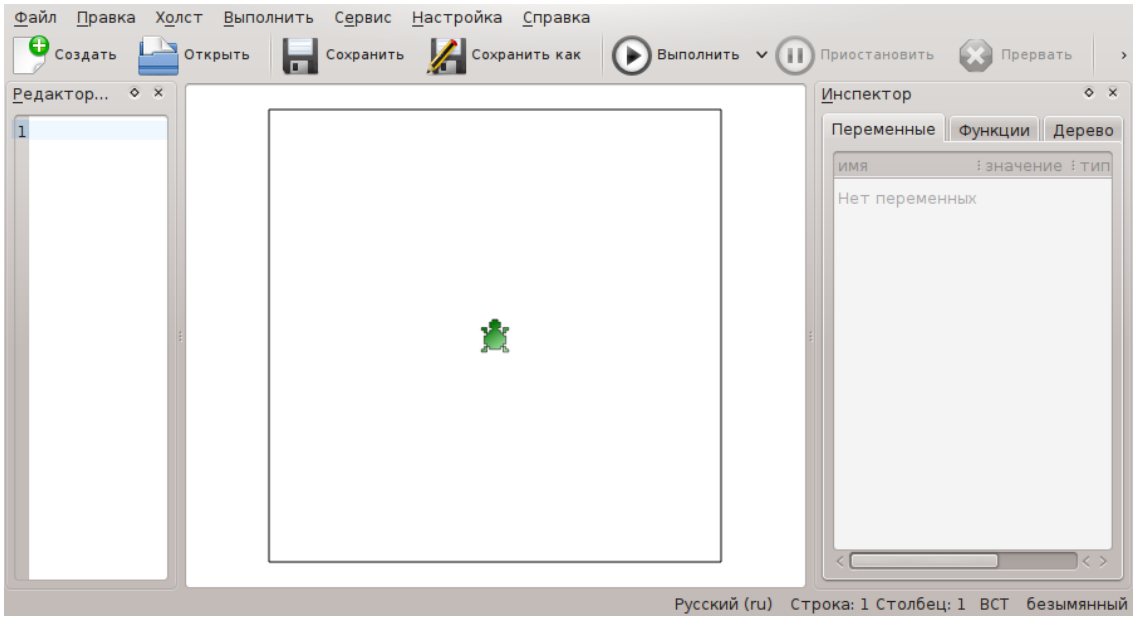
2.6 Строка состояния

В строке состояния выводится информации от KTurtle. Слева показывается последнее действие. Справа показывается текущая позиция курсора (номера строк и столбцов). Посередине показывается текущий язык, используемый для команд.

Глава 3

Начало работы

После запуска KTurtle вы увидите нечто подобное:



В данном руководстве мы будем использовать команды TurtleScript на русском языке. Вы можете поменять язык в **Настройка → Язык команд**. Обратите внимание, что выбранный здесь для KTurtle язык является языком команд TurtleScript, а не языком, используемым в KDE на вашем компьютере для отображения интерфейса и меню KTurtle.

3.1 Первые шаги в TurtleScript: познакомьтесь с Черепашкой!

Вы, наверное, уже заметили Черепашку в центре холста, сейчас мы научимся ей управлять, используя команды в редактора кода.

3.1.1 Движения Черепашки

Давайте начнём с изучения движений Черепашки. Наша Черепашка может перемещаться тремя способами: (1) вперёд и назад, (2) налево или направо или (3) перескочить сразу к определённой точке экрана. Попробуйте, например, это:

```
вперёд 100
налево 90
```

Введите или скопируйте эти строчки в редактор кода (обратите внимание на использование буквы «ё») и запустите код на выполнение, используя **Выполнить → Выполнить**), чтобы увидеть результат.

После ввода и запуска команд, подобных приведённым выше, вы можете наблюдать одну или несколько подобных вещей:

1. После запуска команд на выполнение Черепашка продвинулась вверх, рисуя линию, и затем повернулась на 90 градусов влево. Это произошло потому, что мы использовали команды `вперёд` и `налево`.
2. Как вы могли заметить, при вводе кода его цвет меняется. Это называется *подсветка кода* (разные типы команд подсвечиваются по-разному). Подсветка делает чтение блоков кода удобнее.
3. Черепашка рисовала тонкой чёрной линией.
4. Может быть вы получили сообщение об ошибке. Это может произойти по одной из двух причин: вы совершили промашку при копировании команд или же не выставили правильный язык ввода команд TurtleScript (вы можете сделать это в меню **Настройка → Язык команд**).

Как вы могли понять, команда `вперёд 100` указала Черепашке на то, что ей необходимо двигаться вперёд, оставляя за собой линию, а команда `налево 90` — повернуть на 90 градусов влево.

Обратитесь, пожалуйста, к следующим ссылкам справочного руководства для полного описания новых команд: `вперёд`, `назад`, `налево` и `направо`.

3.1.2 Примеры

Первый пример был совсем простенький, продолжаем!

```
сброс

размер_холста 200,200
цвет_холста 0,0,0
цвет_пера 255,0,0
толщина_пера 5

иди 20,20
направление 135

вперёд 200
налево 135
вперёд 100
налево 135
вперёд 141
налево 135
вперёд 100
налево 45

иди 40,100
```

Чтобы увидеть результат, вы снова можете набрать код вручную, либо скопировать его в редактор или же открыть пример `стрелка` в меню **Примеры** и запустить его, используя **Выполнить → Выполнить**. В следующих примерах ожидается, что вы уже ознакомлены с описанными способами ввода кода.

Как вы могли заметить, второй пример содержит гораздо больше кода. Также вы увидели пару новых команд. Ниже дано краткое описание новых команд:

После команды `сброс` всё возвращается в состояние, которое было сразу после старта KTurtle.

`размер_холста 200,200` устанавливает ширину и высоту холста в 200 пикселов. Ширина и высота одинаковы, холст будет квадратным.

`цвет_холста 0,0,0` задаёт чёрный цвет холста. `0,0,0` — это RGB комбинация: комбинация красной, зелёной и синей составляющих цвета (по-английски: red, green, blue, или сокращённо — RGB). Если все значения установлены в `0`, получится чёрный цвет.

`цвет_пера 255,0,0` устанавливает красный цвет для пера. `255,0,0` — это RGB комбинация, где красная составляющая равна `255`, а все остальные — `0`. Результатом будет красный цвет.

Если вы не разбираетесь в значениях цвета, обратитесь к статье глоссария RGB комбинации.

`толщина_пер 5` устанавливает толщину (размер) пера в `5` пикселов. С этого момента Черепашка будет рисовать линию толщиной `5` до тех пор, пока мы не зададим другую толщину пера при помощи команды `толщина_пер`.

`иди 20,20` указывает Черепашке перескочить в определённое место холста. Отсчитывается от верхнего левого угла. В данном случае Черепашка должна перескочить на 20 пикселов вправо от левой границы холста и на 20 пикселов вниз от верхней границы холста. Примечание: при использовании команды `иди` Черепашка не рисует линию.

`направление 135` задаёт направление Черепашки. Команды `налево` и `направо` изменяют направление Черепашки на заданный угол относительно текущей позиции. Команда `направление` изменяет направление Черепашки на заданный угол относительно 0 и не обращает внимание на предыдущее направление.

После команды `направление` следует множество команд `вперёд` и `налево`. Эти команды выполняют рисование.

В конце используется ещё одна команда `иди` для отвода Черепашки в сторону.

Ознакомьтесь с описанием всех этих команд, перейдя по ссылкам в справочник. Там они объяснены более подробно.

Глава 4

Руководство программиста TurtleScript

Это руководство по языку программирования TurtleScript, используемому в KTurtle. В первой части рассматриваются некоторые аспекты грамматики программ на TurtleScript. Вторая часть описывает исключительно математические операторы, логические (истина/ложь) операторы и операторы сравнения. В третьей главе перечислены все команды с их описаниями. Четвёртая глава рассматривает вопросы присвоения значений переменным. В заключение, в пятой главе рассказывается, как управлять выполнением программы при помощи управляющих операторов, и в шестой — о создании собственных команд при помощи команды выучи.

4.1 Грамматика TurtleScript

Как и в любом языке, в TurtleScript есть различные типы слов и символов. В русском языке мы различаем глаголы (например, «ходить» или «петь») и существительные (например, «сестра» или «дом»). Эти слова используются для различных целей. TurtleScript — это язык программирования, используя который вы можете сообщить KTurtle, что необходимо сделать.

В этом разделе кратко рассказано о некоторых типах слов и символов в TurtleScript. Мы объясняем комментарии, команды и три различных типа литералов: числа, строки и логические (истина/ложь) значения.

4.1.1 Комментарии

Программа состоит из инструкций, которые выполняются при запуске программы и, так называемых, комментариев. KTurtle не выполняет комментарии, а просто игнорирует их при выполнении программы. Комментарии нужны для других программистов, чтобы они лучше могли понять вашу программу. Всё, что следует после символа # считается в TurtleScript комментарием. Вот пример маленькой программы, которая ничего не делает:

```
# Это маленькая программа ничего не делает. Это всего лишь комментарий!
```

Это в некоторой степени бесполезный пример, но он хорошо объясняет суть комментариев.

Комментарии очень полезны в более сложных программах. Они могут дать какие-то советы другим программистам. В следующей программе вы можете видеть совместное использование комментариев и команды напиши.

```
# эта программа создана Cies Breijs.
напиши "этот текст будет написан на холсте"
# предыдущая строка - не комментарий, а следующая - комментарий:
# напиши "этот текст не будет написан"
```

Первая строка описывает программу. Вторая строка выполняется KTurtle и печатает на холсте `этот текст будет написан на холсте`. Третья строка является комментарием. Четвёртая строка — это тоже комментарий, содержащий строку TurtleScript. Если удалить символ `#` из четвёртой строки, команда печати будет выполнена KTurtle. Программисты говорят: оператор печати в четвертой строке «закомментирован».

В редакторе кода комментарии выделяются светло-серым.

4.1.2 Команды

С помощью команд вы говорите Черепашке или KTurtle, что необходимо выполнить какие-то действия. Некоторым командам нужны входные данные, некоторые дают что-то на выходе.

```
# команде вперёд нужны входные данные, в этом примере - число 100:
вперёд 100
```

Первая строка это комментарий. Вторая строка содержит команду `вперёд` и число `100`. Число не является частью команды, он считается «входом» для команды.

Некоторым командам, например, команде `иди` требуется более чем одно входное значение. Несколько значений отделяются друг от друга символом `,` (запятая).

Подробное описание всех команд, поддерживаемых KTurtle, находится здесь. Встроенные команды выделяются тёмно-синим

4.1.3 Числа

Скорее всего, вы уже знаете немного о числах. То, как используются числа в KTurtle не сильно отличается от разговорного языка или математики.

Существуют так называемые натуральные числа: `0`, `1`, `2`, `3`, `4`, `5` и т.д., отрицательные числа: `-1`, `-2`, `-3` и т.д. и рациональные (дробные) числа, например: `0.1`, `3.14`, `33.3333`, `-5.05`, `-1.0`. Символ `.` (точка) используется для отделения дробной части.

Числа могут использоваться с математическими операторами и с операторами сравнения. Также они могут быть сохранены в переменных. Числа выделяются тёмно-красным.

4.1.4 Строки

Пример:

```
напиши "Привет, я строка."
```

В этом примере `напиши` — команда, которой передаётся строка `"Привет, я строка."`. Строки должны начинаться и заканчиваться символом `"`, чтобы KTurtle смог определить, что это строка.

Строки можно помещать в переменные, так же как и числа. Тем не менее, в отличие от числа, строки не могут быть использованы в математических операторах или операторах сравнения. Строки выделяются красным.

4.1.5 Логические (истина/ложь) значения

Существует только два логических значения: `истина` и `ложь`. Иногда они также называются: включено и выключено, да и нет, один и ноль. Но в TurtleScript мы всегда называем их `истина` и `ложь`. Посмотрите на этот фрагмент кода TurtleScript:

```
$a = истина
```

Если вы посмотрите в инспектор, то увидите, что переменная `$a` имеет значение `истина` и логический тип.

Часто логические значения появляются в результате операций сравнения, например, как в следующем фрагменте кода TurtleScript:

```
$ответ = 10 > 3
```

Переменная `$ответ` установлена в значение `истина` потому, что `10` больше чем `3`.

Логические значения `истина` и `ложь` выделяются тёмно-красным.

4.2 Математические, логические операторы и операторы сравнения

Название этого раздела может показаться очень трудным, но это не так сложно, как кажется.

4.2.1 Математические операторы

Вот основные математические символы: сложение (`+`), вычитание (`-`), умножение (`*`), деление (`/ `) и возведение в степень (`^`).

Небольшой пример использования математических операторов в TurtleScript:

```
$сумма        = 1 + 1
$разница      = 20 - 5
$произведение = 15 * 2
$частное      = 30 / 30
$степень      = 2 ^ 2
```

Результаты выполнения математических операций сохранены в различных переменных. Вы можете увидеть их значения в инспекторе.

Если в программе вам нужно вычислить простое выражение, вы можете поступить следующим образом:

```
напиши 2010-12
```

Вот пример с круглыми скобками:

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

Выражение в скобках будет вычислено первым. В данном примере сначала будет получена разность 20 – 5, затем полученное значение будет умножено на 2, поделено на 30 и, напоследок, будет добавлена единица (результат равен 2). Скобки также могут быть использованы и в других случаях.

В KTurtle также есть дополнительные математические функции в виде команд. Обратите внимание на следующие команды, но знайте, что речь идёт о сложных операциях: округли, случайное, sqrt , pi, sin, cos, tg, arcsin, arccos, arctg.

4.2.2 Логические (истина/ложь) операторы

В то время как математические операторы используют числа, логические операторы используют логические значения (`истина` и `ложь`). Существует только три логических оператора, а именно: `и`, `или` и `не`. Следующий фрагмент кода TurtleScript демонстрирует их использование:

```
$и_1_1 = истина и истина  # -> истина
$и_1_0 = истина и ложь    # -> ложь
$и_0_1 = ложь и истина    # -> ложь
$и_0_0 = ложь и ложь      # -> ложь

$или_1_1 = истина или истина  # -> истина
$или_1_0 = истина или ложь    # -> истина
$или_0_1 = ложь или истина    # -> истина
$или_0_0 = ложь или ложь      # -> ложь

$не_1 = не истина  # -> ложь
$не_0 = не ложь    # -> истина
```

С помощью инспектора можно увидеть значения, но мы также указали их в комментариях в конце каждой строки. `и` вернёт `истина`, только если обе стороны содержат `истина`. `или` вернёт `истина`, если хотя бы одна из сторон содержит `истина`, и `не` превращает `истина` в `ложь`, а `ложь` в `истина`.

Логические операторы выделяются розовым.

4.2.2.1 Несколько более сложных примеров

Рассмотрим следующий пример с `и`:

```
$a = 1
$b = 5
если (($a < 10) и ($b == 5)) и ($a < $b) {
  напиши "привет"
}
```

В этом фрагменте кода TurtleScript результаты трёх операторов сравнения объединены операторами `и`. Это означает, что для печати «привет» все три сравнения должны вернуть «истина».

Пример с `или`:

```
$n = 1
если ($n < 10) или ($n == 2) {
  напиши "привет"
}
```

В этом фрагменте кода TurtleScript левая часть от `или` возвращает «истина», а правая часть — «ложь». Так как одна из сторон оператора `или` — «истина», этот оператор вернёт «истина» и будет напечатано «привет».

И, наконец, пример с `не`, меняющим «истина» на «ложь» и «ложь» на «истина»:

```
$n = 1
если не ($n == 3) {
  напиши "привет"
} иначе {
  напиши "не привет ;-)"
}
```

4.2.3 Операторы сравнения

Рассмотрим простое сравнение:

```
$ответ = 10 > 3
```

Здесь 10 сравнивается с 3 оператором «больше чем». Результатом сравнения является логическое значение истина, которое сохраняется в переменной $ответ.

Все числа и переменные (содержащие числа) можно сравнивать друг с другом при помощи операторов сравнения.

Здесь приведены все операторы сравнения:

$A == $B	равно	ответ «истина» если $A равно $B
$A != $B	не равно	ответ «истина» если $A не равно $B
$A > $B	больше чем	ответ «истина» если $A больше чем $B
$A < $B	меньше чем	ответ «истина» если $A меньше чем $B
$A >= $B	больше или равно	ответ «истина» если $A больше или равно $B
$A <= $B	меньше или равно	ответ «истина» если $A меньше или равно $B

Таблица 4.1: Типы вопросов

Обратите внимание, что $A и $B должны быть числами или переменными с числами.

4.3 Команды

Используя команды, вы указываете Черепашке или KTurtle выполнить какое-то действие. Некоторым командам необходимы входные данные, некоторые выводят данные. В этом разделе мы опишем все встроенные команды KTurtle. Используя команду выучи, вы можете создавать собственные команды. Описываемые здесь встроенные команды выделяются тёмно-синим.

4.3.1 Двигаем Черепашку.

Существует несколько команд для перемещения Черепашки по экрану.

вперёд (вп)

```
вперёд X
```

вперёд перемещает Черепашку на X пикселов вперёд. Когда перо опущено Черепашка будет оставлять за собой след. вперёд также может записываться как вп.

назад (нд)

```
назад X
```

назад перемещает Черепашку назад на X пикселов. Когда перо опущено Черепашка будет оставлять за собой след. назад также может записываться как нд.

налево (лв)

```
налево X
```

`налево` предписывает Черепашке повернуть на X градусов налево. `налево` также может записываться как `лв`.

направо (пр)

```
направо X
```

`направо` предписывает Черепашке повернуть на X градусов направо. `направо` также может записываться как `пр`.

направление (нпр)

```
направление X
```

`направление` устанавливает направление Черепашки на X градусов относительно 0, а не относительно предыдущего направления. `направление` также может записываться как `нпр`.

получить_направление

```
получить_направление
```

`получить_направление` возвращает направление Черепашки в градусах относительно нуля, где нулём считает положение Черепашки смотрящей вверх.

центр

```
центр
```

`центр` перемещает Черепашку в центр холста.

иди

```
иди X,Y
```

`иди` предписывает Черепашке занять определённое место на холсте. Это место отстоит на X пикселей от левой границы и на Y пикселей от верхней границы холста.

иди_гор

```
иди_гор X
```

`иди_гор` используется для перемещения Черепашки на X пикселов от левой границы холста, высота остаётся неизменной.

иди_верт

```
иди_верт Y
```

`иди_верт` предписывает Черепашке переместиться на Y пикселей от верхней границы холста. Положение относительно левой границы остаётся неизменным.

> **Замечание**
> При использовании команд `иди`, `иди_гор`, `иди_верт` и `центр` Черепашка не будет рисовать линию вне зависимости от того, поднято перо или опущено.

4.3.2 Где Черепашка?

Существует две команды, возвращающие положение Черепашки на экране.

получить_гор

`получить_гор` возвращает количество пикселей от левого края холста до текущей позиции Черепашки.

получить_верт

`получить_верт` возвращает количество пикселей от верхнего края холста до текущей позиции Черепашки.

4.3.3 У Черепашки есть перо

У Черепашки есть перо, которым она рисует линию во время перемещения. Есть несколько команд для управления пером. В данном разделе они будут описаны подробно.

перо_подними (пп)

```
перо_подними
```

`перо_подними` отрывает перо от холста. Пока перо оторвано, Черепашка не будет рисовать линию во время перемещений. См. также `перо_опусти`. `перо_подними` также может записываться как `пп`.

перо_опусти (по)

```
перо_опусти
```

`перо_опусти` опускает перо на холст. Когда перо опущено, Черепашка при перемещениях рисует линию. См. также `перо_подними`. `перо_опусти` также может записываться как `по`.

толщина_пера (тп)

```
толщина_пера X
```

`толщина_пера` устанавливает толщину пера (толщину линии) в X пикселов. `толщина_пера` также может записываться как `тп`.

цвет_пера (цп)

```
цвет_пера R,G,B
```

`цвет_пера` устанавливает цвет пера. В качестве параметров указывается RGB комбинация. `цвет_пера` также может записываться как `цп`.

4.3.4 Команды для работы с холстом.

Существует несколько команд для работы с холстом.

размер_холста (рх)

```
размер_холста X,Y
```

`размер_холста` устанавливает размер холста. В качестве входных параметров задаются ширина X и высота Y в пикселах. `размер_холста` также может записываться как `рх`.

цвет_холста (цх)

```
цвет_холста R,G,B
```

`цвет_холста` устанавливает цвет холста. Входным параметром является RGB комбинация. `цвет_холста` также может записываться как `цх`.

4.3.5 Команды очистки

Существуют две команды очистки холста.

очисти (очс)

```
очисти
```

`очисти` удаляет с холста все рисунки. Всё остальное останется по-прежнему: позиция и угол направления Черепашки, цвет холста, видимость Черепашки и размер холста.

сброс

```
сброс
```

`сброс` очищает больше, чем команда `очисти`. После выполнения команды `сброс` всё будет выглядеть так, как будто вы только что запустили KTurtle. Черепашка будет расположена в центре экрана, цвет холста будет белым, и Черепашка будет рисовать чёрные линии на холсте размером 400 x 400 пикселей.

4.3.6 Черепашка — это спрайт.

Большинство людей и понятия не имеют, что такое спрайты, так вот, спрайты — это маленькие картинки, которые можно перемещать по экрану. Так что Черепашка — это спрайт.

Ниже будет дано подробное описание всех команд работы со спрайтами.

[Текущая версия KTurtle пока не поддерживает использование спрайтов, отличных от Черепашки. В ближайшем будущем вы сможете заменить Черепашку на любой другой персонаж по собственному вкусу]

покажи (пж)

```
покажи
```

`покажи` делает Черепашку видимой после того, как она была скрыта. `покажи` также может записываться как `пж`.

спрячь (сч)

```
спрячь
```

`спрячь` скрывает Черепашку. Это полезно, когда Черепашка не уместна в ваших рисунках. `спрячь` также может записываться как `сч`.

4.3.7 Черепашка может писать?

Черепашка может написать всё, что вы ей прикажете.

напиши

```
напиши X
```

`напиши` используется для указания Черепашке написать что-либо на холсте. В качестве входных параметров можно передавать строки или числа. Вы можете печатать различные числа и строки, комбинируя их вместе оператором «+». Вот маленький пример:

```
$год = 2003
$автор = "Cies"
напиши $автор + " начал проект KTurtle в " + $год + " и до сих пор  ↩
    получает удовольствие от работы над ним!"
```

размер_шрифта

```
размер_шрифта X
```

`размер_шрифта` устанавливает размер шрифта, используемый для печати. Входной параметр один. Он должен быть числом. Размер задаётся в пикселах.

4.3.8 Математические команды

Далее перечислены дополнительные математические команды KTurtle.

округли

```
округли(x)
```

`округли` заданное число до ближайшего целого.

```
напиши округли(10.8)
вперёд 20
напиши округли(10.3)
```

Выполняя этот код, Черепашка напечатает числа 11 и 10.

случайное (слч)

```
случайное X,Y
```

`случайное` — команда, которая имеет входные и выходные параметры. На входе требуются два числа. Первое (X) задаёт нижний порог получаемых чисел, второе (Y) задаёт верхний порог. На выходе получается псевдослучайное число, которое не меньше минимума и не больше максимума. Вот маленький пример:

```
повтори 500 {
  $x = случайное 1,20
  вперёд $x
  налево 10 - $x
}
```

Используя команду `случайное`, вы можете привнести немного хаоса в вашу программу.

остаток

```
остаток X,Y
```

`остаток` возвращает остаток после деления первого числа на второе.

sqrt

```
sqrt X
```

`sqrt` используется для поиска квадратного корня от числа X.

pi

```
pi
```

Эта команда возвращает константу Пи, `3.14159`.

sin, cos, tg

```
sin X
cos X
tg X
```

Эти три команды представляют всемирно известные тригонометрические функции `sin`, `cos` и `tg`. Входной аргумент этих команд, X, является числом.

arcsin, arccos, arctg

```
arcsin X
arccos X
arctg X
```

Вот список обратных тригонометрические функций для sin, cos и tg. Входной аргумент этих функций, X, является числом.

4.3.9 Взаимодействие через диалоги

Диалог — маленькое окно, которое обеспечивает обратную связь или запрашивает ввести что-либо. В KTurtle есть две команды для диалога: `сообщение` и `спроси`

сообщение

```
сообщение X
```

Команде `сообщение` требуется передать строку, она будет показана в появившемся окне.

```
сообщение "Cies начал KTurtle в 2003 сих пор получает удовольствие ↩
    от работы над ним!"
```

спроси

```
спроси X
```

`спроси` принимает на вход строку. Она показывает эту строку во всплывающем диалоге (как и команда сообщение), а также она показывает поле ввода. После того, как пользователь ввёл число или строку, это значение может быть сохранено в переменной или передано как аргумент команде. Например:

```
$вход = спроси "В каком гору ты родился?"
$выход = 2003 - $вход
напиши "В 2003 тебе было " + $выход + " лет."
```

Если пользователь ничего не введёт или просто закроет диалог, то переменная будет пустой.

4.4 Назначение переменных

Вначале рассмотрим переменные, а затем присвоение значений переменным.

Переменные — это слова, начинающиеся с символа «$». В редакторе кода они выделяются фиолетовым.

Переменные могут содержать числовые, строковые или логические (истина/ложь) значения. Значение присваивается переменной с использованием символа `=`. Переменные хранят значения до завершения программы или до того момента, как в переменную будет записано новое значение.

После присвоения значения переменной, вы сможете использовать это значение. Например, в следующем фрагменте кода TurtleScript:

```
$x = 10
$x = $x / 3
напиши $x
```

Сперва переменной `$x` присваивается значение `10`. Далее переменной `$x` присваивается новое значение её самой, разделённой на `3`. Фактически это означает, что переменной `$x` будет присвоен результат `10 / 3`. В конце значение переменной `$x` распечатывается. В строках используется содержимое переменной `$x`.

Для того чтобы использовать переменную, её надо предварительно назначить. Например:

```
напиши $n
```

Даст сообщение об ошибке.

Рассмотрим следующий фрагмент кода TurtleScript:

```
$a = 2004
$b = 25

# следующая команда напечатает "2029"
напиши $a + $b
назад 30
# следующая команда напечатает "2004 плюс 25 равно 2029"
напиши $a + " плюс " + $b + " равно " + ($a + $b)
```

В первых двух строках переменные `$a` и `$b` устанавливаются равными 2004 и 25. Далее выполняются две команды `напиши` и, между ними, команда `назад 30`. Комментарии перед `напиши` объясняют, что будет напечатано. Команда `назад 30` используется здесь для уверенности в том, что каждая новая печать на холст будет на новой строке. Как видите, переменные используются точно также, как и их содержимое — совместно с любыми операторами или как входные данные при вызове команд.

Ещё пример:

```
$имя = спроси "Как тебя зовут?"
напиши "Привет " + $имя + "! Удачи в изучении искусства программирования ↩
    ..."
```

Простой и понятный пример. Переменная `$имя` рассматривается как строка.

Инспектор очень сильно помогает при использовании переменных. Он отображает содержимое всех используемых в данный момент переменных.

4.5 Контроль над выполнением

Управляющие операторы позволяют вам контролировать процесс выполнения (о чём говорит само их название).

Управляющие операторы выделяются тёмно-зелёным цветом и жирным начертанием шрифта. Скобки, используемые вместе с ними, выделяются чёрным.

4.5.1 Может ли Черепашка ждать?

Если вы уже немного попрактиковались в программировании в KTurtle, вы могли заметить, что Черепашка может рисовать чересчур быстро. Следующая команда позволяет избежать этого.

жди

```
жди X
```

Команда **жди** указывает Черепашке подождать X секунд.

```
повтори 36 {
  вперёд 5
  направо 10
  жди 0.5
}
```

Данный код рисует круг, при этом после каждого шага Черепашка будет ждать пол секунды. Это создаёт впечатление неторопливого движения.

4.5.2 Условное выполнение: «если»

если

```
если логическое значение { ... }
```

Код, расположенный между скобками, будет выполнен только `если` в результате вычисления логического значения получится «истина».

```
$x = 6
если $x > 5 {
  напиши "$x больше пяти!"
}
```

В первой строке контейнеру `$x` присваивается значение 6. Во второй строке оператор сравнения используется для вычисления `$x > 5`. Если ответом на него будет «истина», 6 больше чем 5, управляющий оператор `если` позволит выполнить код, расположенный между скобками.

4.5.3 Если нет, другими словами: «иначе»

иначе

```
если логическое значение { ... } иначе { ... }
```

`иначе` может быть дополнением к оператору условного выполнения `если`. Код между скобками, расположенными после `иначе` будет выполнен тогда, когда логическое выражение будет «ложь».

```
сброс
$x = 4
если $x > 5 {
  напиши "$x больше пяти!"
}
иначе
{
  напиши "$x меньше пяти!"
}
```

Оператор сравнения вычисляет выражение `$x > 5`. Так как 4 не больше, чем 5, результатом будет «ложь». Это означает, что будет выполнен код между скобками после `иначе`.

4.5.4 Цикл «пока»

пока

```
пока логическое значение { ... }
```

Управляющий оператор `пока` очень похож на `если`. Разница в том, что `пока` будет повторять код, расположенный между скобками, до тех пор, пока логическое выражение не выдаст значение «ложь».

```
$x = 1
пока $x < 5 {
  вперёд 10
  жди 1
  $x = $x + 1
}
```

В первой строке `$x` присваивается 1. На второй строке выполняется сравнение `$x < 5`. Так как получаемое значение — «истина», оператор `пока` начнёт выполнять код между скобками, пока не будет получена «ложь». В данном случае код между скобками будет выполнен 4 раза, потому что на каждом прогоне в пятой строке `$x` будет увеличиваться на 1.

4.5.5 Цикл «повтори»

повтори

```
повтори число { ... }
```

Управляющий оператор `повтори` похож на `пока`. Разница в том, что `повтори` повторяет код в скобках заданное число раз.

4.5.6 Считающий цикл «для»

для

```
для переменная = число до число { ... }
```

Цикл `для` — это цикл «со счётчиком», то есть он сохраняет счётчик для вас. Первое число задаёт значение для первого прохода. С каждым проходом это значение увеличивается до того момента, пока не достигнет второго числа.

```
для $x = 1 до 10 {
  напиши $x * 7
  вперёд 15
}
```

Каждый раз, когда выполняется код в скобках, значение `$x` увеличивается на 1, и так до тех пор, пока `$x` не станет равным 10. Код в скобках выводит на печать произведение `$x` и 7. После завершения выполнения программы вы увидите на холсте таблицу умножения на 7.

По умолчанию, шаг цикла равен 1. Вы можете задать другой шаг цикла с помощью

```
для переменная = число до число шаг шаг { ... }
```

4.5.7 Выход из цикла

прекрати

```
прекрати
```

Немедленно завершает выполнение текущего цикла и передаёт управление оператору, следующему сразу после этого цикла.

4.5.8 Остановка выполнения программы

закончить

```
закончить
```

Заканчивает выполнение программы.

4.6 Создавайте свои собственные команды!

Команда `выучи` — это особенная команда, потому что она предназначена для создания ваших собственных команд. Создаваемые вами команды могут принимать входные параметры и возвращать различные значения. Давайте посмотрим, как создаются собственные команды:

```
выучи круг $x {
  повтори 36 {
    вперёд $x
    налево 10
  }
}
```

Новая команда называется `круг`. Команда `круг` принимает один входной аргумент для задания размера круга. Команда `круг` ничего не возвращает. Теперь команду `круг` можно использовать как же, как и обычные команды:

```
выучи круг $X {
  повтори 36 {
    вперёд $X
    налево 10
  }
}

иди 200,200
круг 20

иди 300,300
круг 40
```

В следующем примере будет создана команда, возвращающая значение.

```
выучи факториал $x {
  $r = 1
  для $i = 1 до $x {
    $r = $r * $i
  }
  верни $r
}

напиши факториал 5
```

В данном примере создана новая команда с именем `факториал`. Если на вход этой команде подать `5`, то на выходе будет произведение `5*4*3*2*1`. При использовании `верни` будет задано выходное значение и оно будет возвращено при выполнении.

У команды может быть более одного входа. В следующем примере создаётся команда, рисующая прямоугольник:

```
выучи прямоугольник $x, $y {
  вперёд $y
  направо 90
  вперёд $x
  направо 90
  вперёд $y
  направо 90
  вперёд $x
  направо 90
}
```

Теперь вы можете дать команду `прямоугольник 50, 100` и Черепашка нарисует на холсте прямоугольник.

Глава 5

Глоссарий

В данной главе вы найдёте объяснение большинства «непонятных» слов, встречающихся в данном руководстве.

градусы

Градусы — единицы измерения углов или поворотов. Полный разворот — это 360 градусов, половина разворота — это 180 градусов, четверть разворота — 90 градусов. Входными параметрами команд `налево`, `направо` и `направление` являются углы в градусах.

входные параметры и возвращаемые значения команд

Некоторым командам необходимы входные параметры, некоторые возвращают значения. Есть такие, которые имеют *и* вход, *и* выход, а есть, наоборот, не имеющие ни входных параметров, ни возвращаемых значений.

Вот несколько команд, имеющих только входные параметры:

```
вперёд 50
цвет_пера 255,0,0
напиши "Привет!"
```

Команда `вперёд` принимает в качестве входного параметра число `50`. Данный параметр указывает команде `вперёд` на сколько пикселов вперёд должна продвинуться Черепашка. Входным параметром для `цвет_пера` является цвет, а для `напиши` — строка. И не забывайте, что входным параметром также может являться контейнер. Следующий пример продемонстрирует это:

```
$x = 50
напиши $x
вперёд 50
$стр = "привет!"
напиши $стр
```

Теперь приведём примеры команд, возвращающих значения:

```
$x = спроси "Введите что-нибудь и нажмите OK... спасибо!"
$r = случайное 1,100
```

Команда `спроси` принимает в качестве входного параметра строку, а возвращает введённое число или строку. Как вы можете заметить, возвращаемое `спроси` значение помещается в контейнер `x`. Команда `случайное` также возвращает значение. В данном случае это будет число от 1 до 100. Как и в случае с предыдущей командой, выходное значение `случайное` также помещается в контейнер, имеющий имя `r`. Надо заметить, что контейнеры `x` и `r` нигде до этого в коде примера не использовались.

Упомянем и команды, которые ничего не принимают и ничего не возвращают. Вот несколько примеров:

```
очисти
перо_подними
```

подсветка синтаксиса

Это особенность KTurtle делает программирование очень простым. С интуитивной подсветкой синтаксиса весь код в редакторе выводится разными цветами, в зависимости от того, для чего предназначена та или иная часть программы. В следующем списке вы найдёте описание разных типов кода и цветов, которые они получают в редакторе кода.

команды	тёмно-синий	Обычные команды описаны здесь.
команды контроля выполнения	чёрный (жирный)	Эти специальные команды контролируют выполнение. Узнать больше можно здесь.
комментарии	серый	Строки комментария начинаются со знака комментария (#). Они игнорируются при выполнении программы. Комментарии необходимы для пояснения программистом того, что он делает в том или ином фрагменте кода, а также для того, чтобы временно не выполнять какие-либо команды.
скобки {, }	тёмно-зелёный (жирный)	Скобки используются для группировки фрагмента программы. Зачастую скобки используются совместно с командами контроля выполнения.
команда выучи	светло-зелёный (жирный)	Команда выучи используется для создания новых команд.
строки	красный	Единственное, что мы скажем о строках — они должны начинаться и заканчиваться двойными кавычками (″).
числа	тёмно-красный	Числа..., да вроде бы говорить о них нечего.
логические значения	тёмно-красный	Существует только два логических значения: истина и ложь.
переменные	фиолетовый	Начинается с символа «$» и может содержать цифры, строки и логические значения.

математические операторы	серый	Математические операторы: `+`, `-`, `*`, `/` и `^`.
операторы сравнения	светло-синий (жирный)	Операторы сравнения: `==`, `!=`, `<`, `>`, `<=` и `>=`.
логические операторы	розовый (жирный)	Логические операторы: `и`, `или` и `не`.
обычный текст	чёрный	

Таблица 5.1: Разные типы кода и их раскраска

пикселы

Пиксел — точка на экране. Если вы посмотрите на экран с очень близкого расстояния вы увидите, что ваш монитор использует пикселы. Пиксел — наименьшая частица, которая может быть нарисована на экране.

Множеству команд требуется количество пикселов в качестве входных параметров. Вот эти команды: `вперёд`, `назад`, `иди`, `иди_гор`, `иди_верт`, `размер_холста` и `толщина_пера`.

В ранних версиях KTurtle холст был растровым изображением, в более поздних версиях холст — это векторный рисунок. Это означает, что холст можно увеличить или уменьшить, так как один пиксел не обязательно должен соответствовать одной точке на экране.

RGB комбинации (коды цветов)

RGB комбинации используются для описания цветов. «R» отвечает за «красный», «G» за «зелёный» и «B» за «синий» цвета. Например, рассмотрим комбинацию `255,0,0`: первое число, отвечающее за «красный», равно 255, а два остальных равны 0. Это говорит о том, что данная комбинация передаёт чистейший красный цвет. Каждая составляющая комбинации лежит в диапазоне от 0 до 255. Ниже приведён пример нескольких часто используемых цветов:

`0,0,0`	чёрный
`255,255,255`	белый
`255,0,0`	красный
`150,0,0`	тёмно-красный
`0,255,0`	зелёный
`0,0,255`	голубой
`0,255,255`	светло-голубой
`255,0,255`	розовый
`255,255,0`	жёлтый

Таблица 5.2: Часто используемые RGB комбинации

RGB комбинации в качестве входных параметров используются в двух командах: `цвет_холста` и `цвет_пера`.

спрайт

Спрайт — это небольшая картинка, перемещаемая по экрану. Наша Черепашка, к слову, является спрайтом.

Примечание: в данной версии KTurtle спрайт не может быть заменён с Черепашки на что-либо другое. В следующих версиях KTurtle такая возможность будет предусмотрена.

Глава 6

Руководство переводчика KTurtle

Как вы уже, вероятно, знаете, язык программирования TurtleScript, используемый в KTurtle, можно перевести. Это устраняет барьер для некоторых (в особенности юных) учеников в их трудах по освоению основ программирования.

При переводе KTurtle на новый язык, в стандартных .pot файлах, используемых при переводе KDE, помимо строк графического интерфейса вы обнаружите ещё и команды программирования, примеры и сообщения об ошибках. Всё это переводится с использованием стандартных способов, принятых в KDE, однако мы настоятельно рекомендуем немного ознакомиться с тем, как переводить такие строки и как понимать комментарии для переводчиков.

За дополнительной информацией о процессе перевода обратитесь, пожалуйста, к http://edu.kde.org/kturtle/translator.php. Спасибо большое за вашу работу! Переводы очень важны для KTurtle.

Глава 7

Авторские права и лицензия

Глава 8

Предметный указатель