Томас Тулка

Цифровые приключения

Как программируют компьютеры



ДМК
издательство

Томас Тулка

Цифровые приключения: как программируют компьютеры

Computer Science for Kids

Discover how computers work
and learn the power of programming!

Tomas Tulka



Leanpub

Цифровые приключения: как программируют компьютеры

Томас Тулка

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
Москва, 2025

УДК 087.5:004.4
ББК 32.973
Т82

Тулка Т.
Т82 Цифровые приключения: как программируют компьютеры / пер. с англ. А. Н. Киселева. – М.: ДМК Пресс, 2025. – 142 с.: ил.

ISBN 978-5-93700-420-8

Благодаря забавным иллюстрациям и увлекательным заданиям эта книга о приключениях маленького робота сделает сложные идеи простыми для понимания. По мере развития сюжета дети овладеют основами программирования и приобретут навыки решения задач с помощью кода. Увлекательное интерактивное приключение пробудит в них интерес к информатике и языкам программирования и покажет, насколько безграничны возможности современных технологий.

Книга адресована любознательным детям и их родителям, желающим превратить знакомство с программированием компьютеров в веселую игру.

УДК 087.5:004.4
ББК 32.973

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-5-93700-420-8 (рус.)

© 2025 Tomas Tulka
© Перевод, оформление, издание, ДМК Пресс, 2025

Содержание

Вступление.....9

Послание из космоса – *история начинается*.....10

КАК РАЗГОВАРИВАТЬ С РОБОТАМИ

Электронные слова – *цифровая связь*.....15

Универсальные роботы.....16
Сигналы.....17
Кодирование.....18
Команды.....21
Бинарный алфавит.....23

Механический язык – *языки программирования*.....27

Компиляторы.....30
Программы.....31

Сверкающие глаза – *устройства отображения*.....34

Пиксели.....35
Цвета.....38

Кремниевый разум – *из чего сделаны компьютеры*.....42

Роботы.....43
Компьютеры.....43
Интегральные схемы.....44
Ввод и вывод.....45
Процессор.....46
Память.....48

КАК ПРОГРАММИРОВАТЬ РОБОТОВ

Странный мир – *введение в программирование* 51
Языки программирования 52
LudolfC 53

Понемногу обо всём – *типы данных* 54
Числа 57
Текст 58
Истинность 59
Составные типы данных 62
Порядок операций 62
Упражнение 64

И то, и это – *переменные* 65
Что такое переменные? 66
Как создаются переменные 67
Имена переменных 68
Значения переменных 69
Использование переменных 70
Указатели 72
Упражнение 74

Шаг за шагом – *массивы* 75
Что такое массивы 76
Как записываются массивы 77
Многомерные массивы 78
Как работать с массивами 78
Размер массива 80
Массивы в памяти компьютера 81
Упражнение 82

Так или иначе – *условные инструкции* 83
Что такое условные инструкции? 84
Как записываются условные инструкции 84
А если условие не соблюдается? 85
Проверка второго условия в инструкции else 86
Упражнение 87

Снова и снова – *циклы* 89
Что такое цикл? 89
Как записываются циклы 90
Вложенные циклы 91
Упражнение 92

Выполнение по требованию – *функции* 93
Что такое функции 94
Как записываются функции 95
Как вызывать функции 95
Функции, которые ничего не возвращают 96
Функции без параметров 96
Использование функций 97
Область видимости функции 97
Замыкания 99
Упражнение 101

Я робот – *объекты* 102
Что такое объекты 103
Как записываются объекты 103
Как работать с объектами 105
Использование объектов 105
Внутри объекта 106
Всё является объектом 108
Фабрики объектов 109
Упражнение 110

НАЗАД ДОМОЙ

Возвращение – *решение* 112
Первая попытка 113
Обход кратеров 116
Переход в начало 117
Энергия 119
Лучшее решение 120
К ракете 123
Упражнение 124

Прощай, комета! – *эпилог* 125

ПРИЛОЖЕНИЕ

Решения 128
Задача № 1 128
Задача № 3 128
Задача № 4 128
Задача № 5 129
Задача № 6 129
Задача № 7 129

Задача № 8 129
Задача № 9 130
Задача № 10 130
Задача № 11 130
Задача № 13 131
Задача № 14 131
Задача № 15 131
Задача № 16 131
Задача № 17 132
Задача № 18 132
Задача № 19 132
Задача № 20 132
Задача № 21 133
Задача № 22 133
Задача № 23 133
Задача № 24 134
Задача № 25 134
Задача № 26 134
Задача № 27 134
Задача № 28 135
Задача № 29 135
Задача № 30 135
Задача № 31 135
Задача № 33 135
Задача № 34 136
Задача № 35 136
Задача № 36 136
Задача № 37 137
Задача № 38 137
Задача № 39 138
Задача № 40 138
Задача № 41 138
Задача № 42 138
Задача № 43 138
Задача № 44 139
Задача № 45 139
Задача № 47 139
Задача № 48 139
Задача № 49 139
Задача № 50 139
Задача № 56 140

Благодарности 141

Вступление

В небе комета сияет
Таинственным ярким светом.
Звёзды стыдливо тают,
Будто завидуют ей.

Как пламя комета горит,
Сквозь космос стремительно мчит.
Увидеть её – это дар,
Мгновение, что не повторится.

– написано ChatGPT

Послание из космоса – *история начинается*

За окном сверкнула вспышка и раздался оглушительный грохот! Что там?

Ты подбегаешь к окну и выглядываешь наружу. Из кустов в дальнем конце сада вьётся струйка синеватого дыма. Что случилось? Явно что-то упало с неба!

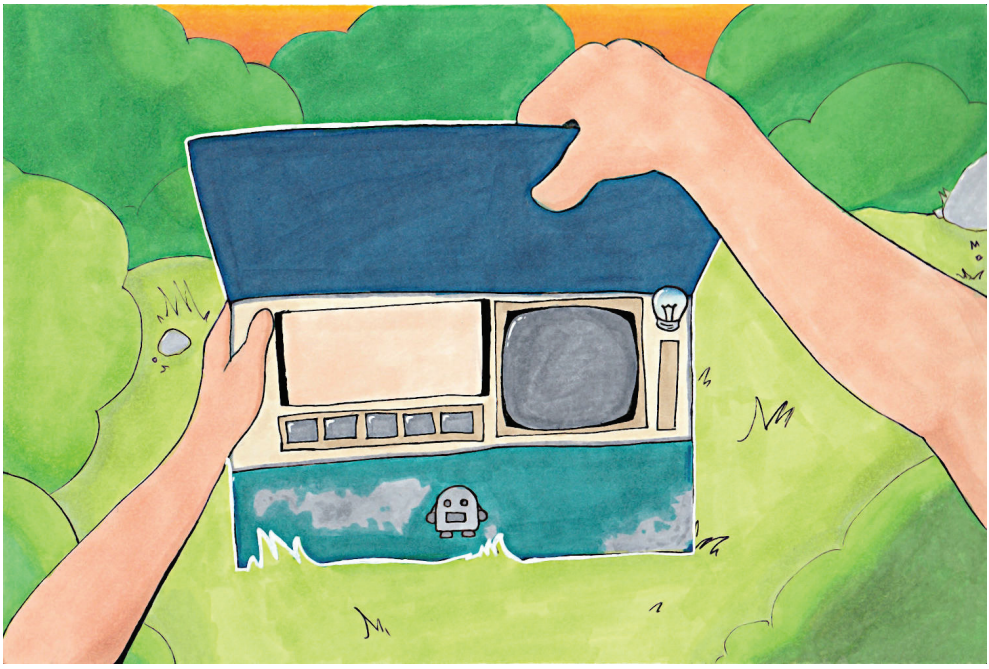
Ты выскакиваешь наружу, чтобы взглянуть поближе. Дым рассеялся, и становятся видны сломанные ветки. Ты подходишь ближе и с любопытством всматриваешься в кусты. На земле среди обломков ветвей и обгоревшей травы лежит маленькая жестяная коробка. Ты прикасаешься к ней и тут же отдёргиваешь руку. Ой! Она горячая! Похоже, она летела на очень большой скорости, возможно, даже из космоса.

Ты смотришь вверх, надеясь увидеть след пролетевшего космического корабля, но синее небо абсолютно чистое. Немного подождав, ты быстро касаешься крышки коробки одним пальцем и на всякий случай тут же отдёргиваешь руку. Но на этот раз ты не обжёгся. Похоже, коробка остыла. Ты пробуешь дотронуться до неё ещё раз, уже медленнее. Затем осмеливаешься положить на неё всю ладонь. Она ещё тёплая, но уже не обжигает. Ты смахиваешь пепел с крышки рукой и видишь изображение маленького робота с надписью «Экспедиция Когоутек II».

Ты немного разочарован, потому что посылку прислали не зелёные инопланетяне с большими рыбьими глазами, какими их показывают в научно-фантастических комиксах.

После осмотра коробки со всех сторон обнаруживается кнопка, после нажатия которой крышка со щелчком открывается, и наружу вылетает облачко пыли. Ты вздрагиваешь от неожиданности и на секунду задерживаешь дыхание. А что, если внутри спрятано что-то опасное?

Но ничего не происходит, и ты медленно открываешь коробку. Внутри оказывается небольшой блокнот, панель с кнопками, оранжевая лампочка и небольшой экран.



Там же лежит пластиковый конверт с надписью «Инструкция». Ты открываешь его и начинаешь читать. Письмо написано красивым почерком. Похоже, что его писал кто-то аккуратный и старательный. В письме говорится:

> Приветствую тебя, друг!
>
> Я – универсальный робот 16-й серии,
> созданный для поиска внеземной жизни.
>
> Моей задачей было исследование кометы
> КОГОУТЕК II.
>
> Однако неожиданная авария привела
> к повреждению моих управляющих программ
> и практически полной потере энергии.
>
> К сожалению, я не могу вернуться домой
> без перепрограммирования и подзарядки
> аккумуляторов.

```
Я прилагаю руководство по управлению со
списком необходимых шагов и прошу помочь мне
решить эту критически важную задачу.

Моя судьба в твоих руках.

Спасибо!
```

Ты стоишь, не веря своим глазам. Что всё это значит? Ты снова и снова перечитываешь письмо. О каких аккумуляторах и управляющих программах в нём говорится? Ты ничего не знаешь ни о комете Когоутек II, ни о роботах, отправляемых в космические экспедиции. И что такое *универсальный робот*? Возможно, это лишь одно большое недоразумение. Но что, если какой-нибудь бедный робот действительно застрял в космосе и нуждается в помощи? Как ему помочь? Как запрограммировать такого робота?

Ты кладёшь письмо и берёшь потрёпанный блокнот, надеясь найти хоть какую-то подсказку. На обложке блокнота написано:

```
== Руководство по управлению ==
робот 16-й серии,

серийный номер 256
```

Ладно, пока понятно. На первой странице руководства ты читаешь:

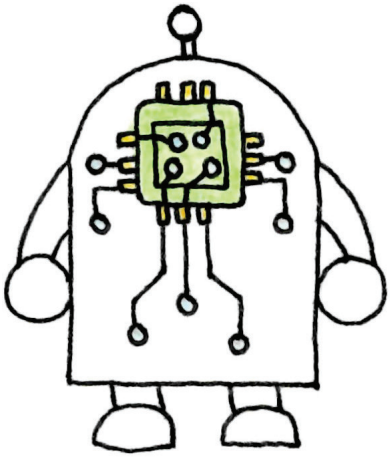
```
Шаг 1. Включите панель управления.
```

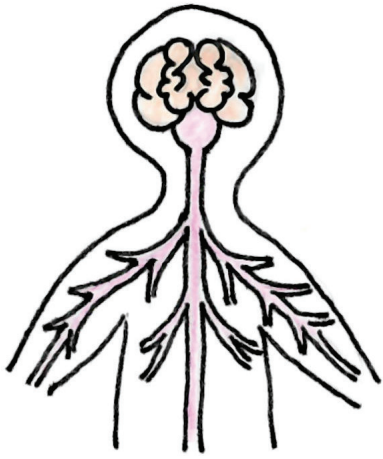
Ты снова смотришь на странное устройство и замечаешь зелёный тумблер с надписью «Пуск». Достаточно щёлкнуть им – и панель управления включится. Спасение может начаться сейчас. Ты уже протягиваешь руку, и тут тебя осеняет: а ведь действительно судьба маленького робота, затерявшегося в просторах космоса, кажется, в твоих руках. Сможешь ли ты ему помочь?

Тебя ждёт множество препятствий и головоломок, которые нужно будет решить, проявив упорство и смекалку. Кажется, у тебя есть всё необходимое, но не хватает самого главного: ты должен научиться работать с роботом и разговаривать с ним. Только поняв его, ты сможешь помочь, а может быть, даже вернуть его домой.

Робот кажется невообразимо сложной машиной, но всякое сложное устройство состоит их простых деталей, работающих вместе. Даже самый сложный дом построен из простых кирпичиков, и самая сложная машина состоит из простых компонентов. Робот – лишь одна их таких машин. Тело робота сделано из

листов металла и кремния, а электрические сигналы циркулируют в его электронном мозгу так же, как в твоём. На самом деле отличий не так уж и много.





Эта книга расскажет тебе, как работают роботы и как их *понимать*: от электрических сигналов до сложных языков программирования.

Робот – это машина, действующая абсолютно логично. Поэтому не полагайся на удачу, а руководствуйся здравым смыслом. У тебя всё получится, если не спешить и последовательно переходить **от простых задач к более сложным**.

Как разговаривать с роботами

Электронные слова – *цифровая связь*

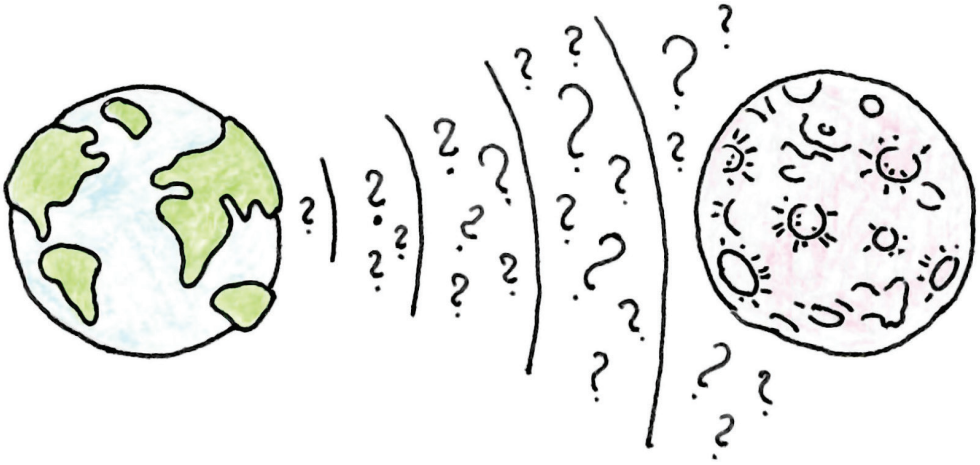
Серебристая ракета, сверкающая в лучах солнца, с маленьким роботом на борту отправляется с родной планеты в дальнюю экспедицию. При взгляде через иллюминаторы Земля выглядит сначала как глубокий океан с плоскими островами, затем как черничный пирог со взбитыми сливками и, наконец, как голубоватое пятнышко размером с грецкий орех. Звёзды кажутся большими и яркими, как кошачьи глаза. Ракета дрожит, проходя сквозь облако звёздной пыли, но робот остаётся спокойным. Цель уже недалеко.

Комета Когоутек II приближается к Солнечной системе с огромной скоростью. Она начинает нагреваться в ярких лучах солнца, а её хвост, состоящий из истекающих испарений, красиво светится.

Кометы имеют очень вытянутые орбиты. Они пролетают через далёкие и неизведанные уголки космоса, прежде чем вернуться обратно. Чтобы сделать один оборот вокруг Солнца, им могут потребоваться сотни или даже тысячи лет. Находиться на комете – всё равно что сесть в межзвёздный экспресс. Но на ней нельзя долго оставаться, иначе можно улететь слишком далеко и не вернуться. Действовать нужно быстро.

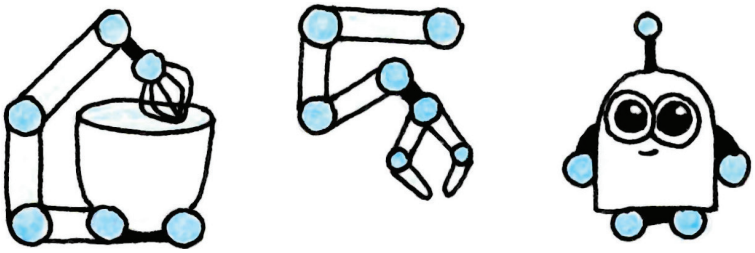
Ты щёлкаешь зелёным тумблером на панели управления, переводя его в положение «Пуск». Внутри что-то начинает гудеть, но потом звук неожиданно обрывается. Даже со второй попытки ничего не происходит. Панель явно повреждена. Возможно, она повредилась при полёте в атмосфере или при ударе о землю. В любом случае она не работает.

Как же быть? Панель управления была единственным способом поговорить с потерявшимся роботом. Чтобы ему помочь, нужно сначала понять, как работает такая связь.



Универсальные роботы

Наверное, тебе приходилось видеть кухонного робота, замешивающего тесто, или робота, пылесосящего пол, или даже более крупных роботов, собирающих автомобили на заводе. Робот – это просто набор деталей, таких как миксер, колёса, пылесос или механическая рука, собранных и действующих вместе.



Универсальные роботы (например, те, которые посылаются в исследовательские экспедиции) отличаются от кухонных роботов – они не специализируются на выполнении какой-то конкретной работы и способны решать широкий спектр задач – от сбора проб до выполнения сложных математических расчётов.

Мозг универсального робота – это мощный компьютер, управляющий всеми остальными компонентами. Поскольку мозг робота – это компьютер, то и роботы, по сути, являются всего лишь большими компьютерами. Робот может функционировать без рук и без ног, но ничего не сможет сделать без своего центрального компьютера. Без него он – просто куча бесполезных запчастей.



Задача № 1

Подобное универсальное существо уже имеется в живой природе. Догадайся, кто это?

(Решение этой и всех других задач ты найдёшь в конце книги.)

Сигналы

Как заставить руку почесать ухо или написать домашнее задание по математике? Ты просто думаешь об этом, а мысль затем передаётся из мозга в руку в виде электрических сигналов, распространяющихся по нервным волокнам.

В этом отношении роботы мало отличаются от людей. Когда робот хочет пошевелить рукой, его мозг – центральный компьютер – посылает электронный сигнал механической руке, которая затем трёт ухо или пишет домашнее задание по математике (если, конечно, у робота есть уши и он ходит в школу).

Сигналы – это основные элементы коммуникации, поэтому они должны быть предельно простыми: *сигнал либо есть, либо его нет* – нет ничего проще!

Сигнал можно представить как последовательность простых событий. Например, мигание лампочки или постукивание по стене. Слышал ли ты об азбуке Морзе? Её суть как раз заключается в подаче таких сигналов. Древние индейцы посылали дымовые сигналы, закрывая и открывая огонь. Вспышка, клуб дыма или постукивание – всё это сигналы.



Для любознательных

Сигналы передаются либо по кабелям и проводам, либо по воздуху. В кабелях обычно используется электрический ток или свет (да, некоторые действительно передают свет – такие кабели называются оптоволоконными).

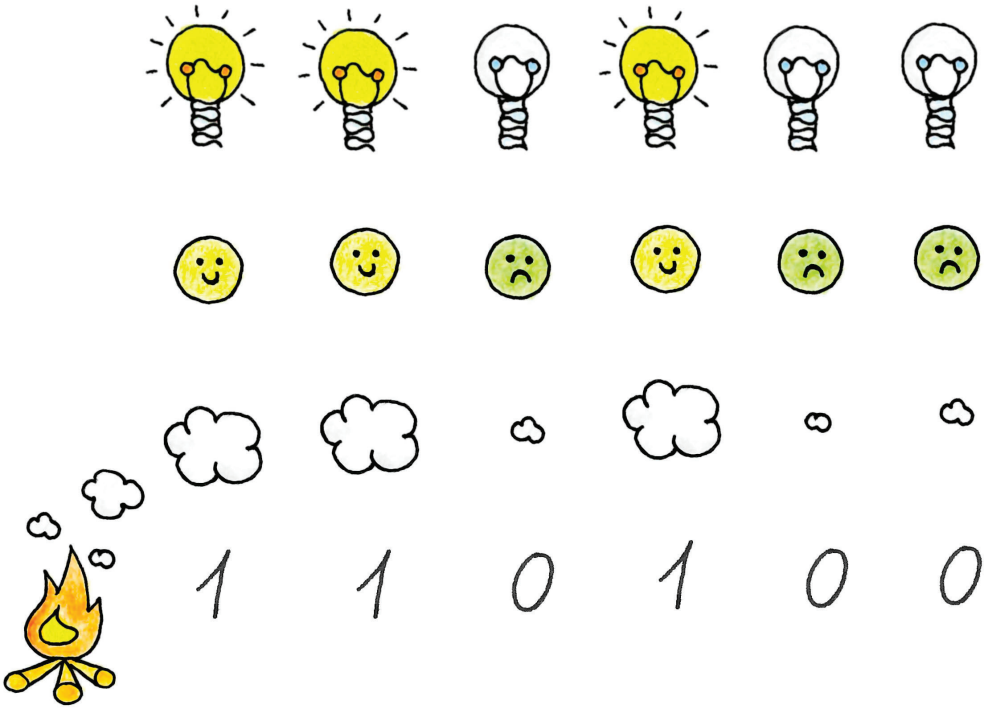
По воздуху сигналы передаются с помощью электромагнитных волн, которые можно представить как разновидность света, невидимого невооружённым глазом. Антенны и спутниковые тарелки на крышах домов принимают электромагнитные волны.

Кодирование

Сигналы можно записывать на бумаге с помощью символов. Там, где нет вспышки света, клуба дыма или звука, записываются пробелы. Например, в азбуке Морзе точки используются для представления сигнала, а тире – для представления пустых мест там, где сигнала нет.

Роботы используют для записи сигналов **нули и единицы**. Однако ничто не мешает записывать сигналы, например, с использованием двух разных смайликов. К сожалению, у роботов почти нет чувства юмора.

Ты также скоро узнаешь, что у их любимых нулей и единиц есть одно важное преимущество: с их помощью можно производить *математические вычисления*. А поскольку роботы – это компьютеры, вычисления – их любимое занятие.



Процесс записи сигналов с помощью нулей и единиц называется **кодированием** или **оцифровкой**, а записанный таким способом сигнал известен как *цифровой* или *числовой* код. Например, код `110100` – это оцифровка следующего постукивания по стене: *стук*, *стук*, *пауза*, *стук*, *пауза*, *пауза*.

Задача № 2

Попробуй отстучать этот цифровой код костяшкой пальца по столу: `010101101`.

Попроси кого-нибудь из членов семьи или друзей попробовать записать твои звуковые сигналы в виде цифрового кода (используя единицы и нули).

Для любознательных

Оцифровка – это процесс преобразования сигналов из *несовершенного представления* (свет, звук, электричество) *в физическом мире людей* в *совершенное представление* (единицы и нули) *в мире компьютеров* и чисел.

Это не самый простой процесс, потому что в разные моменты лампочка может мигать с разной яркостью. То же относится к постукиванию и электричеству. А оцифрованный сигнал свободен от таких погрешностей, и с ним гораздо проще работать, чем с исходным несовершенным физическим сигналом.

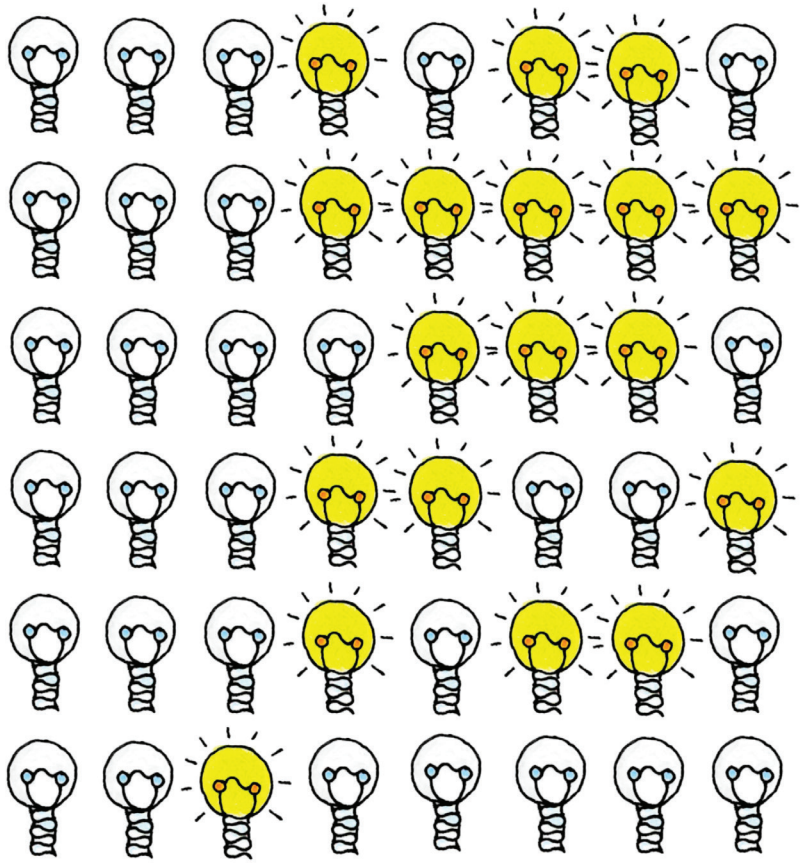
Сравни следующий электрический сигнал и его цифровой код. Обрати внимание, что единицы и нули в физическом сигнале немного отличаются друг от друга, тогда как в цифровом коде они абсолютно одинаковы:



Узнав, как работает электронная связь, ты быстро понял, почему панель управления не работает: лампочка разбилась при падении и произошёл разрыв электрической цепи.

Ты мчишься в свою комнату, выкручиваешь лампочку из настольного светильника и бежишь обратно. Затем осторожно меняешь разбитую лампочку и снова поворачиваешь зелёный тумблер на панели управления в положение «Пуск».

Панель тихо щёлкает, а лампочка выдаёт такую серию сигналов:





Задача № 3

Попробуй закодировать этот световой сигнал, используя нули и единицы как цифровой код.

Ты только что получил первое сообщение из далёкого мира. Теперь ты знаешь, как можно разговаривать с роботом! Но что означают все эти единицы и нули? К сожалению, описание второго шага в руководстве мало чем помогло:

```
Шаг 2. Прочитайте сообщение.
```

К счастью, эту задачу всё же можно решить!

Для любознательных

Кодирование – это не что-то особенное, характерное только для машин. Живые организмы используют молекулярный код ДНК, в котором закодированы инструкции по построению клеток и тканей. Вся информация о нас и наших особенностях закодирована в нашей ДНК.

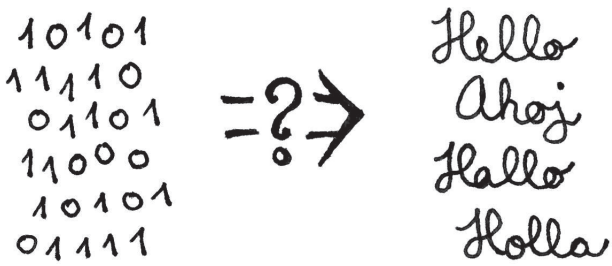
Величина носа, цвет глаз и волос и даже чувство юмора и кулинарные предпочтения – всё это закодировано в ДНК.

Команды

Сигналы подобны буквам, из которых складываются слова. Но как составить слово всего из двух букв (`0` и `1`), если человеческий алфавит содержит их гораздо больше?

Цифровой код, состоящий только из двух значений (`0` и `1`), называется **бинарным кодом** (на латыни *бинарный* означает двоичный). Из двух букв можно составить только два слова. Например, `0` может означать *«выкл»*, а `1` – *«вкл»*.

Для простых роботов двух слов может быть вполне достаточно, но определённо недостаточно для универсального робота. Нужен какой-то способ, который позволит с помощью всего двух букв написать столько же слов, сколько используют люди.



Итак, как написать больше двух слов в бинарном коде? Просто использовать *более длинный код*!

Как и в человеческой речи, для выражения более длинной мысли необходимо больше слов. Использование кода с длиной в три символа позволит создать больше слов, чем использование кода с длиной в один символ.

Например:

Код	Команда
000	выключить
001	включить
010	петь
011	танцевать
100	сделать бутерброд
101	вынести мусор
110	прибрать в комнате
111	решить домашнее задание

Как оказалось, используя бинарный код с длиной в *три символа*, можно выразить восемь различных команд. И для этого не потребовалось ничего, кроме нулей и единиц!

Чтобы выразить больше восьми команд, нужно лишь использовать чуть более длинный код, и так можно продолжать до бесконечности. Чем длиннее код, тем больше команд можно выразить с его помощью.

Однако такой способ довольно сложен. Чтобы универсальный робот мог понимать все возможные команды, ему потребовалась бы чрезвычайно длинная таблица. Это всё равно, как если бы каждое человеческое слово выражалось своей буквой. Тогда у нас было бы невообразимое количество букв. Ещё одна проблема – создание новых слов. Каждое новое слово потребовало бы создания совершенно другой буквы. Это было бы крайне неудобно.

Нужно какое-то другое решение, чтобы не пришлось писать бесконечно длинные таблицы команд. Можно, например, пойти таким путём: вместо кодирования конкретных команд можно сначала закодировать простые буквы, затем из букв составлять слова и, наконец, из слов – команды, как мы делаем это в повседневной жизни.

Поскольку человеческий алфавит содержит больше восьми букв и цифр, необходим код с длиной больше трёх символов. Для начала будет достаточно восьми.

Отдельные символы кода называются **битами** (*«двоичными цифрами»*). То есть предыдущая кодировка команд была трёхбитной, а теперь она будет восьмибитной. Группы по восемь бит часто называют **байтом** или *октетом* (на латыни *окто* означает восемь).



Роботы обычно начинают счёт битов *с нуля*, то есть первый бит имеет порядковый номер `0`, второй бит имеет порядковый номер `1`, третий – `2` и т. д.

Теперь запишем *символы* (буквы и цифры) в пустую таблицу и присвоим им уникальные коды. Таблицу можно отсортировать по алфавиту и начать с цифр (от `0` до `9`), потому что цифр меньше, чем букв (от `A` до `Я`):

Символ	**Код**
0	?
1	?
2	?
3	?
…	
9	?
А	?
Б	?
В	?
…	
Я	?

И какие коды мы должны использовать? Было бы идеально, если бы коды можно было сортировать и по величине. Но можно ли сортировать бинарный код? Как оказывается, можно. Бинарный код – это просто числа (нули и единицы), а числа можно сортировать по их величине!

Бинарный алфавит

В повседневной жизни числа записываются десятью символами: `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9` и `0`. Этот тип кода известен как **десятичная** система счисления (то есть использующая десять цифр). Например, число двести пятьдесят шесть записывается как `256`.

В десятичной системе счисления нет ничего особенного. Она была выбрана просто потому, что людям было проще считать на своих десяти пальцах. Если бы у людей было по одному пальцу на каждой руке, то есть всего два пальца, то мы бы с вами считали в двоичной системе.

Для счёта в пределах десяти можно считать на пальцах. Однако, когда счёт будет переваливать за число десять, нам придётся где-то одолжить дополнительные пальцы. Люди могли бы считать на пальцах только до двух, если бы у них было по одному пальцу на каждой руке.

Число два является *основанием* для двоичной системы счисления, так же как число десять является основанием для десятичной системы счисления (по количеству пальцев).

Для любознательных

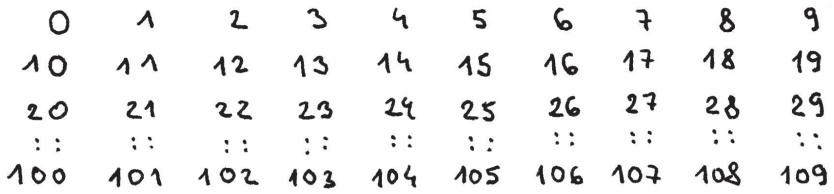
Роботы могут также использовать другие коды для вычислений внутри своих железных голов, например *восьмеричную* систему счисления с основанием восемь или *шестнадцатеричную* – с основанием шестнадцать.

Это похоже на то, как записываются (кодируются) числа. Чтобы записать число меньше десяти в десятичной системе счисления, нужен всего один бит: со значением от нуля до девяти (`0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`). Однако для больших чисел (десять, одиннадцать, двенадцать или, например, двести пятьдесят шесть) требуется два бита и более (`10`, `11`, `12`, `256`).

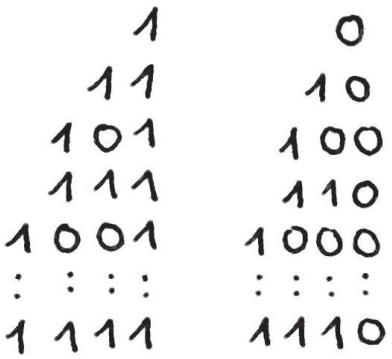
Для любознательных

Почему на пальцах можно считать до десяти, а одним десятичным знаком можно записать только числа до девяти? Причина этого – число ноль. Чтобы выразить число ноль на руках, пальцы не нужны, но для записи нуля на бумаге нужен *специальный символ*: `0`.

То же относится к бинарному коду, за исключением того, что в качестве основания используется два, а не десять.



На предыдущем рисунке старший бит (разряд) увеличивается, только когда предшествующий ему младший бит достигает десяти. То же справедливо и для двоичного счёта: когда счёт достигает двух, просто прибавляй единицу к старшему биту, возвращайся к началу и повторяй.



Теперь становится понятнее, как назначать коды в бинарном алфавите, чтобы иметь возможность сортировки:

	Код		**Код**		**Код**
0	00000000	Е	00001111	У	00011111
1	00000001	Ё	00010000	Ф	00100000
2	00000010	Ж	00010001	Х	00100001
3	00000011	З	00010010	Ц	00100010
4	00000100	И	00010011	Ч	00100011
5	00000101	Й	00010100	Ш	00100100
6	00000110	К	00010101	Щ	00100101
7	00000111	Л	00010110	Ъ	00100110
8	00001000	М	00010111	Ы	00100111
9	00001001	Н	00011000	Ь	00101000
А	00001010	О	00011001	Э	00101001
Б	00001011	П	00011011	Ю	00101010
В	00001100	Р	00011100	Я	00101011
Г	00001101	С	00011101		
Д	00001110	Т	00011110		



Для любознательных

Обрати внимание, что в этой таблице отсутствуют некоторые символы. Например, нет немецкой буквы Ü, испанской Ñ или чешской Ř. Также отсутствуют специальные символы, такие как *точка*, *плюс* и *минус*.

Наш бинарный алфавит очень прост, но его нетрудно расширить позже. Используя восьмибитный код, можно закодировать 256 символов, и если повезёт, то даже найдётся место для нескольких смайликов!

Теперь, наконец, ты сможешь расшифровать сообщение, посланное роботом-исследователем!



Задача № 4

Сверь отправленный код с таблицей и запиши символы рядом друг с другом.

В результате должно получиться имя робота, попавшего в беду.

Механический язык – *языки программирования*

Внимание, подготовка к посадке! На передней панели кабины загорается индикатор обратного отсчёта. *Десять!* Все системы работают нормально. *Девять!* Комету Когоутек II окружает очень разреженная атмосфера, поэтому нет необходимости беспокоиться о перегреве корпуса из-за трения. *Восемь!* Навигационная система рассчитывает наилучшее место для посадки и безопасный угол падения. *Семь!* Комета сложена блестящими породами из металлов и кремния. *Шесть!* Кремний – полуметаллический элемент, используемый для изготовления так называемых полупроводниковых компонентов, а также стекла и фарфора. Кстати, встречающийся в природе красивый фиолетовый минерал аметист – это тоже кремний. *Пять!* Включены посадочные двигатели. Удар о поверхность каменной кометы будет жёстким. *Четыре!* Двигатели замедляют скорость падения, чтобы смягчить посадку. По корпусу ракеты прокатываются волны вибрации. *Три!* Несколько метров до поверхности. *Два!* Сейчас или никогда! *Один!*

Удар! Ракета проваливается в образовавшуюся расщелину. Раздаётся сильный грохот, словно в стальном туннеле. Повсюду искрящаяся пыль и мутный пар. Ракета разваливается на три части. Неужели экспедиция на этом и закончилась? К счастью, нет! Через несколько минут пыль оседает. Вспыхивает зелёный свет. Посадка прошла успешно, и дерзкая экспедиция в места, куда не ступал ни один робот, продолжается!

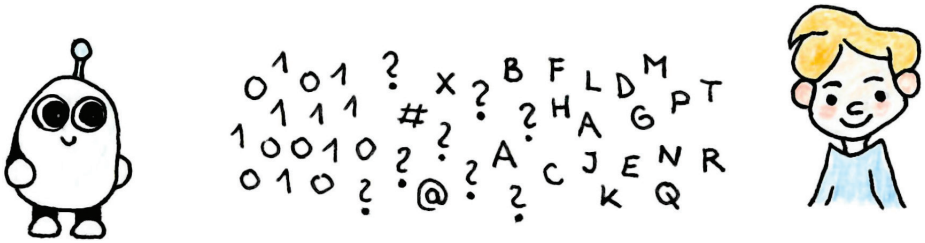
В предыдущей главе ты расшифровал первое сообщение от пропавшего робота Лудолфа и теперь знаешь, как говорит Лудолф и как читать его слова. Ты переворачиваешь страницу блокнота и читаешь следующую инструкцию в руководстве:

```
Шаг 3. Подтвердите получение сообщения.
```

Закодировать сообщение с использованием нулей и единиц и затем отправить его несложно, но неудобно. Например, вот как могло бы выглядеть меню на обед:

```
00011011 00010011 00100010 00100010
00001010
00011100 00011001 00010110 00010110
00100111
00011011 00001010 00011101 00011110
00001010
```

Поскольку общаться таким способом затруднительно, было бы разумнее найти более удобный способ поговорить с Лудолфом.

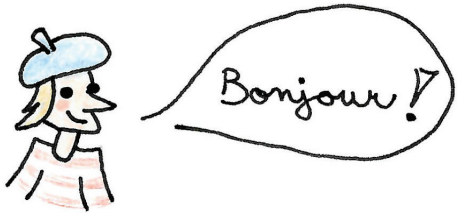


Робот, понимающий только двоичный код, подобен иностранцу, говорящему на непонятном языке. Посмотри, как два таких незнакомца общаются друг с другом.

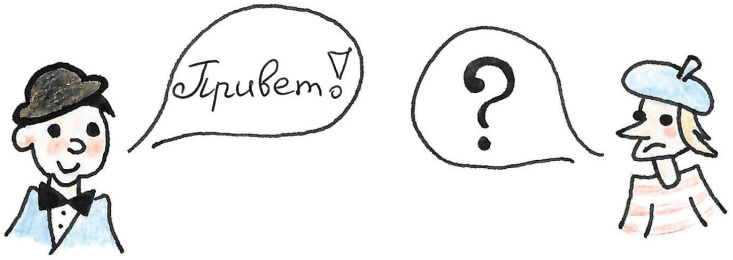
Это Женя. Он – русский и говорит по-русски:



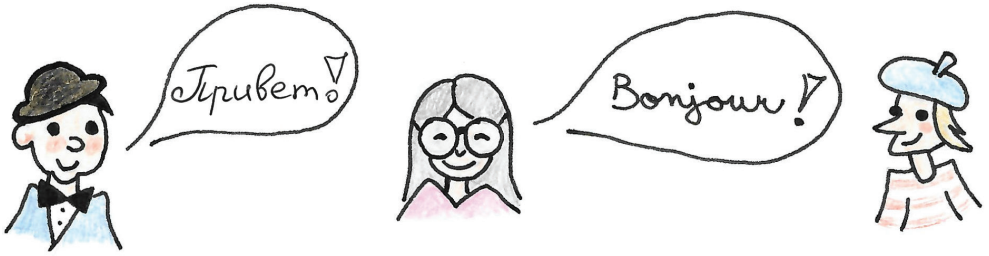
А это – Пьер. Он – француз и говорит по-французски.



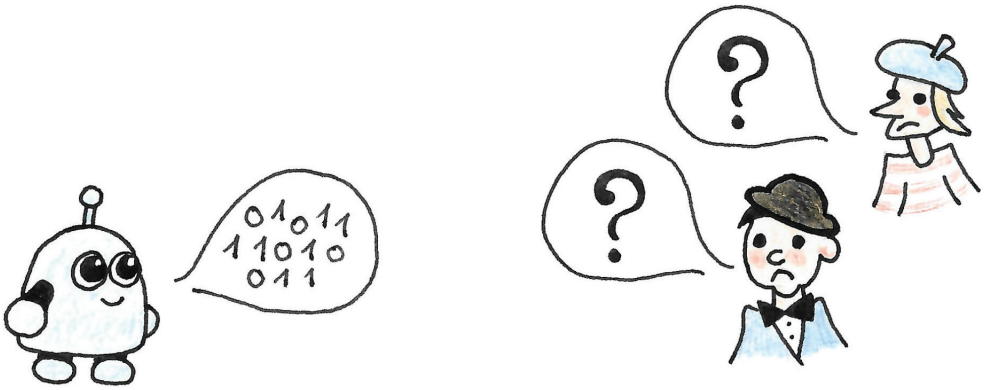
Когда Женя приветствует Пьера по-русски, Пьер его не понимает, потому что они говорят на разных языках:



Чтобы понять друг друга, Женя и Пьер должны выучить язык друг друга или попросить того, кто говорит на обоих языках, перевести их слова. Такого умного человека называют *переводчиком*:



Лудолф – робот, говорящий на языке нулей и единиц. Поэтому Жене и Пьеру будет слишком сложно научиться общаться с ним.



Хорошо бы и в этом случае найти переводчика.



Но где найти такого умного переводчика?

Компиляторы

К счастью, такие переводчики существуют. Иногда их называют *компиляторами*.

Компилятор принимает команды, написанные на языке, похожем на человеческий, и переводит их в нули и единицы. По сути, компилятор – это механизм, который помнит всю таблицу алфавита и преобразует буквы в двоичный код.

Конечно, человек тоже сможет выполнять работу компилятора, как ты это уже делал в предыдущей главе. Но люди не любят скучную работу, предпочитая перекладывать её на плечи машины, поэтому работу компилятора обычно выполняет *компьютерная программа*.



Для любознательных

Компилятор принимает команды, написанные на *языке программирования*, переводит их в двоичный код и сохраняет в новом файле. Этот файл затем загружается и запускается на машине, для которой он был написан. Однако некоторые языки позволяют пропустить шаг перевода и запускать программы в том виде, в каком они были написаны. Делается это с помощью упрощённых компиляторов, которые называются *интерпретаторами*.

Интересный факт: один такой интерпретатор существует в каждом интернет-браузере, а язык, который он интерпретирует, называется *JavaScript*.

Ты можешь найти такой простой интерпретатор в интернете по адресу **ludolf.github.io/abc** – используй кнопку с изображением робота, чтобы закодировать простой текст в двоичный код, и кнопку с изображением человека, чтобы раскодировать двоичный код в текст, понятный человеку.

Программы

Как оказывается, один такой компилятор уже встроен в панель управления, поэтому тебе больше не придётся вручную преобразовывать буквы в слова! Это здорово облегчит работу. Однако есть одна загвоздка.

Хотя роботы очень умны (например, им достаточно одной секунды, чтобы выполнить домашнее задание по математике), с другой стороны, иногда они бывают настолько невежественны, что начинают вести себя глупо. Стоит лишь чуть-чуть неточно выразиться, допустить двусмысленность или ошибку в задании, и робот не сможет понять, что он должен сделать, или поймёт задание неправильно. Ему всё должно быть разъяснено чётко, подробно и без какой-либо путаницы. Достаточно лишь пропустить одну букву, и робот уже не разберётся.

К сожалению для Лудолфа, человеческая речь полна неточностей и двусмысленностей. Люди могут легко догадываться о чём-то и исправлять мелкие ошибки, опираясь на свой опыт или исходя из ситуации. Роботы пока этого не умеют.

Например, когда папа или мама даёт задание прибраться, ты прекрасно понимаешь, что речь идёт о твоей комнате, потому что ты уже знаешь, как сильно им не нравится беспорядок. Но Лудолф вряд ли понял бы, где он должен прибраться – в комнате, на кухне или, может быть, в кабинете природоведения в школе.

Чтобы избежать подобной путаницы, в мире существуют языки, которые хотя и основаны на человеческих языках, но гораздо проще и точнее. Ведь ошибиться в чём-то простом гораздо сложнее, чем в чём-то сложном. Некоторые из этих языков настолько просты, что состоят всего из нескольких слов. Попробуй представить целую речь, состоящую, скажем, всего из восьми слов. Звучит невероятно, но далее в этой книге ты увидишь, что это возможно.

Такие языки называются **языками программирования**, и в мире их довольно много, так что каждый может выбрать себе наиболее подходящий. Например, Женя является поклонником языка программирования *JavaScript*, а Пьер предпочитает язык программирования со странным названием *Python*.



Задача № 5

Найди в интернете несколько языков программирования и перечисли несколько различий между ними.

Какой из них кажется более простым и понятным, а какой – более сложным?

Команды, написанные на некоторых языках программирования, транслируются компилятором на язык нулей и единиц. Результатом является *программа*, которую робот сможет понять и *выполнить*. Написание кода на языке программирования называется **программированием**. Роботы и компьютеры, как правило, *программируются*. Человек, умеющий программировать на языке программирования, называется *программистом*.

Программа – это просто *последовательность команд*, понятных компьютеру. Например, вот программа твоего вечернего распорядка, написанная на упрощённом человеческом языке:

1) умыться,
2) почистить зубы,
3) лечь в постель,
4) сон.



Для любознательных

В программировании часто приходится сталкиваться со словом «алгоритм». *Алгоритм* – это абстрактное описание программы на простом человеческом языке или с использованием рисунков.

Например, *рецепт* приготовления торта – это алгоритм. Программа – это *код, выполняющий некоторый алгоритм*. Например, код из нулей и единиц для кухонного робота, который испечёт торт.

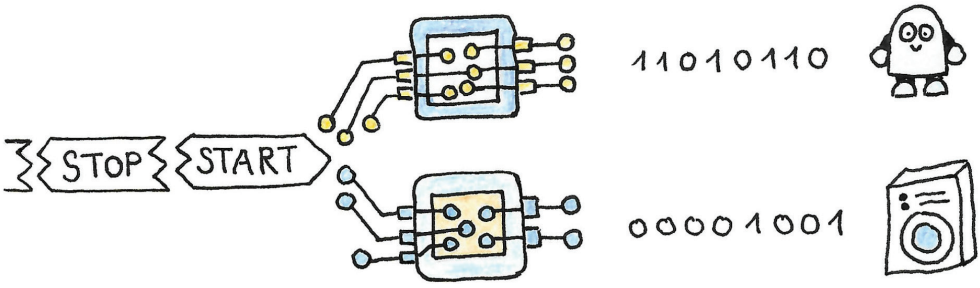


Задача № 6

Напиши простой алгоритм завязывания шнурков.

К сожалению, компилятор может транслировать программы не на всех языках программирования. Большинство компиляторов специализируются на каком-то одном языке. Точно так же не все компиляторы могут транслировать программы на язык нулей и единиц, понятный всем компьютерам. Некоторые специализируются на кухонных роботах, другие – на персональных компьютерах или на стиральных машинах (да, в наши дни даже в стиральных машинах есть программируемые компьютеры). Поэтому необходимо иметь компилятор, понимающий твой любимый язык программирования и способный транслировать программы на язык нулей и единиц твоего компьютера.

Но почему скомпилированная программа может выполняться не на любом компьютере? Вспомни таблицу с бинарным алфавитом. В ней ты записал буквы от А до Я и присвоил им определённые двоичные коды. Буквы важны для универсальных роботов, таких как Лудолф. Другим роботам, например домашней стиральной машине, буквы совершенно не нужны. Для такой стиральной машины нужны только команды включения, выключения, стирки и отжима. Это означает, что таблица для стиральной машины выглядит совершенно иначе, чем для Лудолфа. Например, код `00001010` для Лудолфа означает букву `А`, а для стиральной машины – команду остановки отжима. По этой причине компилятор должен не только понимать язык программирования, но и уметь правильно его перевести для конкретного компьютера.



Но если в панели управления имеется компилятор, то почему на синем экране ничего не отображается? Ответ ты узнаешь в следующей главе.

Сверкающие глаза – *устройства отображения*

Солнце на комете светит так же ярко, как на Земле. Однако условия там более жёсткие и суровые. Из-за разреженной атмосферы бóльшая часть тепла моментально улетучивается в космос. Кроме того, на поверхности кометы периодически бушуют мощные бури и ураганы. Мало кому захотелось бы жить там.

Но самое чуждое – это небо. Даже днём, когда солнце находится в зените, небо остаётся чёрным как смоль. Белый солнечный свет на Земле преломляется, искривляется и рассеивается молекулами воздуха в атмосфере, из-за чего небо имеет синий цвет. Но на Когоутеке II почти нет воздуха, поэтому даже в полдень небо остаётся чёрным.

Но нехватка цвета в небе с лихвой компенсируется поверхностью кометы. Кристально белые горы и изумрудные озёра покрывают её. Между ними лежат сверкающие пустыни, а в фиолетовых скалах скрываются туннели и пещеры. Потрясающе красиво!

Экран кажется выключенным, но на панели нет кнопки для его включения. После осмотра его со всех сторон выясняется причина проблемы.

Экран состоит из нескольких рядов мягких крошечных стеклянных линз, расположенных друг над другом. Прозрачные линзы соединены спутанными тонкими проводами. В момент удара при приземлении несколько проводов оборвались, из-за чего экран перестал работать и оставался тёмным, как небо поздним вечером.

Чтобы воспользоваться компилятором, нужно соединить провода и оживить экран.

Эта работа требует немного терпения, но на самом деле она совсем не сложная.

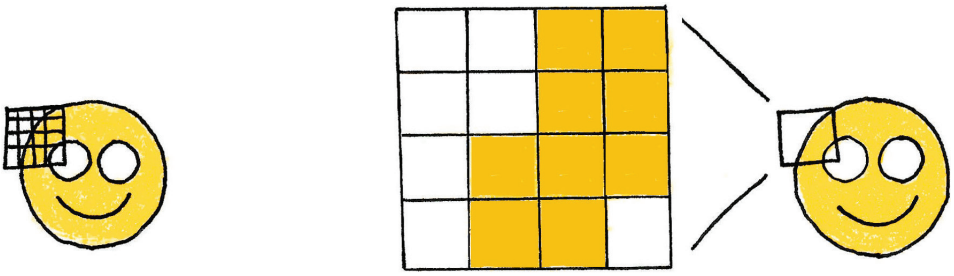
Экраны, как и все другие *устройства связи* (клавиатура, наушники, мышь и джойстик), – это всего лишь *потребители двоичного кода*. Представьте себе линзы экрана как клетки на шахматной доске. У каждой клетки есть свой уникальный номер. Чтобы что-то отобразить, программа отправляет соответствующий двоичный код с тем же количеством битов, что и количество линз. Если бит с тем же номером равен единице, линза загорается; если равен нулю, линза гаснет.

Пиксели

Линзы экрана обычно называются *пикселями*.

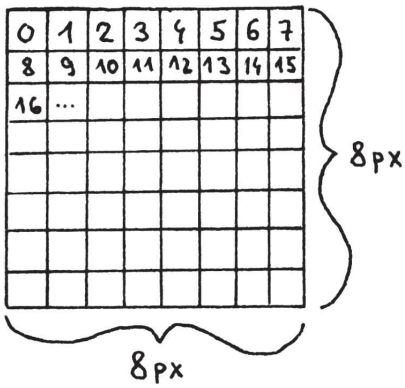
Пиксель – это самая маленькая точка, которую может отобразить экран. Пиксели можно считать *мельчайшими цветными точками*, из которых состоят изображения и видео.

Попробуй сильно увеличить масштаб изображения или фотографии на твоём мобильном телефоне, и ты заметишь маленькие квадратные точки – пиксели.



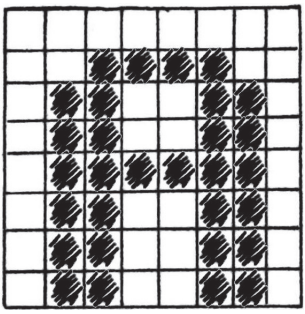
Пиксели бывают разных размеров. Пиксель на экране смартфона меньше миллиметра, но на больших экранах, таких как те, что используются на стадионах, он может достигать нескольких сантиметров.

Однако код изображения не зависит от типа экрана, на котором он отображается. Поэтому имеет смысл указывать размер изображения не в сантиметрах, а в пикселях. Для экранов важно количество пикселей, которые он может отобразить. Это количество называется **разрешением**. Например, следующий экран имеет разрешение 8×8 пикселей:



Букву A можно отобразить на этом экране с помощью следующего кода:

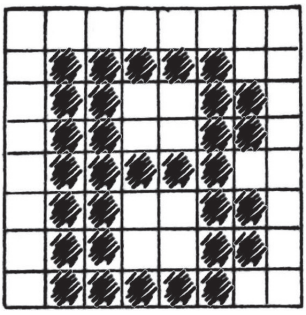
```
0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
```





Задача № 7

Попробуй написать код, который отобразит букву B, как показано на изображении ниже:



Простой экран с разрешением 8×8 пикселей можно найти на **ludolf.github.io/display** – просто введи двоичный код и отобрази его, нажав кнопку с изображением глаз.



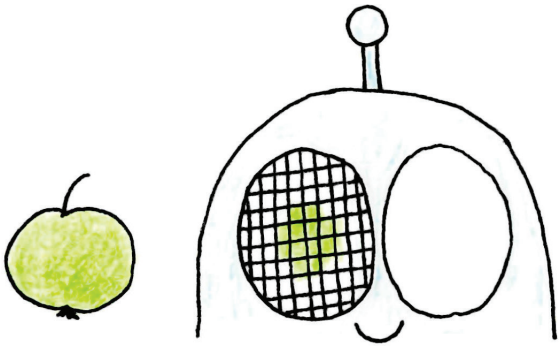
Для любознательных

Возможно, вам интересно, как компьютер распознаёт изображение, созданное программой. Оба кода на первый взгляд выглядят одинаково – как набор нулей и единиц. Разница заключается в так называемом *заголовке* – первых нескольких битах, определяющих *тип* кода. Например, если код начинается с битов `01`, значит это изображение, а если с битов `10`, то это выполняемая программа.

Различные типы файлов часто различаются по *расширению*: например, расширение `.png` используется для изображений, а `.exe` – для выполняемых программ. Расширения помогают людям понять, что находится в файле перед ними. Однако благодаря заголовкам компьютеры могут не смотреть на расширения.

Всё то же самое верно и в обратную сторону. Как и экран, глаза Лудолфа состоят из сетчатых линз. Линзы светочувствительны и могут определять, падает ли на них свет. Если свет падает на линзу, она генерирует единицу. Если на линзу не падает свет или падает очень мало света, она генерирует ноль.

Вот как устроены цифровые камеры и другие устройства, которые позволяют машинам *видеть*.





Для любознательных

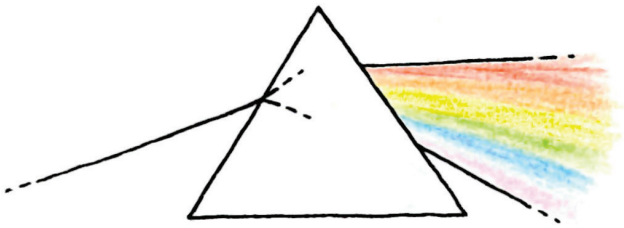
Устройства, чувствительные к окружающей среде (свету, звуку, температуре или, может быть, влажности), называются *датчиками* или *сенсорами*.

Цвета

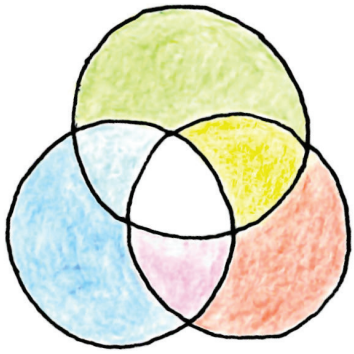
Если каждому пикселю будет соответствовать только один бит, то экран получится *чёрно-белым*. То есть он сможет отображать только два цвета: чёрный, соответствующий единице, и белый, соответствующий нулю. Конечно, экран не обязательно должен быть строго чёрно-белым. Синий и, возможно, розовый тоже возможны, но не более *двух цветов*, поскольку один пиксель соответствует одному биту, а один бит может иметь только два значения (ноль и единица).

Чтобы экран отображал больше цветов, можно воспользоваться тем же решением, что и для большего количества букв в алфавите, – использовать более длинный код, с бóльшим числом битов на пиксель. Можно, к примеру, назначить номера цветам и создать таблицу кодов и цветов, как с буквами и числами в таблице бинарного алфавита.

Проблема в том, как расположить эти цвета. Можно ли сказать, что красный больше синего? А как упорядочить тёмно-синий или тыквенно-оранжевый? Может быть, цвета можно расположить в том же порядке, что и в радуге? Да! И радуга покажет ещё более удачный способ.



Знаешь ли ты, как образуется радуга? Радуга – это преломлённый дневной свет в каплях дождя. Да, белый дневной свет состоит из всех цветов радуги. На самом деле он даже проще, потому что было обнаружено, что все оттенки цветов можно получить всего из трёх цветов: *красного*, *зелёного* и *синего*.



Это открытие было использовано при создании цветных экранов. Достаточно лишь, что каждый пиксель может светиться красным, зелёным и синим одновременно. Комбинируя эти цветные источники света, можно получить любой цвет, включая белый и чёрный.

Как создать белый свет из трёх цветных лучей? Вспомни, как образуется радуга: разные цвета получаются в результате преломления обычного белого света, а значит, белый свет можно получить комбинированием цветных лучей. А чёрный? Нужно лишь выключить весь свет, и станет темно.



Для любознательных

Возможно, ты уже пробовал смешивать цвета подобным способом, рисуя цветными мелками на асфальте или красками на листе бумаге, только при смешивании всех трёх цветов ты получал не белый, а некрасивый коричневый цвет.

Причина в том, что краски на бумаге не *излучают* свет, а *поглощают* его. Без краски бумага *белая*, а экран без света – *чёрный*.

Но краски на бумаге тоже можно смешивать. Было обнаружено, что любой цвет можно создать, смешивая *красную*, *жёлтую* и *синюю* краски. Художнику не нужны другие цвета, потому что из красного, жёлтого и синего он может получить все цвета радуги.

Смешение цветов света обычно кодируется в формате RGB (от английских названий цветов *Red*, *Green* и *Blue* – красный, зелёный и синий). *Интенсивность* отдельных цветов, то есть интенсивность свечения данного цветного света, кодируется числом. Для простых цветов можно использовать значения от нуля до единицы, например, где ноль означает, что свет отсутствует, а единица – светится с максимальной интенсивностью.

Итак, выделив всего один бит на каждый из трёх цветов, то есть три бита на пиксель, получим:



Чтобы получить больше цветов, потребуется больше битов. Современные экраны обычно используют по восемь бит для каждого основного цвета, то есть трижды восемь – *24 бита на пиксель*. Соответственно, один пиксель может отображать более 16 миллионов разных цветов!

Простой цветной экран с разрешением 2×2 пикселя можно найти на **ludolf.github.io/rgb** – просто введи двоичный код цвета и нажми кнопку с изображением глаз, а экран отобразит соответствующий цвет.



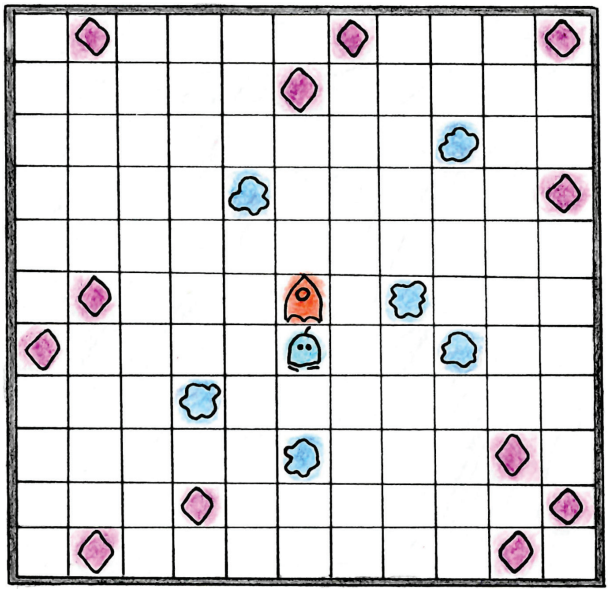
Для любознательных

Для экономии места 24-битные цвета RGB обычно записываются не в двоичном, а в *шестнадцатеричном коде*. Например, значение приятного оранжевого оттенка записывается как `FF9225`.

Цвета, записанные в форме шестнадцатеричных чисел, часто обозначаются знаком # в начале, чтобы было понятно, что это шестнадцатеричная запись, а не десятичная. Поэтому код оранжевого цвета будет записан как `#FF9225`.

Как только ты узнаешь, как работают экраны, станет понятно, как нужно подключить провода к линзам.

Подключив провода, ты вновь переводишь зелёный тумблер в положение «Пуск». Снова мигает лампочка, и на маленьком экране появляется загадочное изображение:



Что это значит? Ты смотришь в руководство, надеясь понять, что изображено на этой странной карте.

```
Шаг 4. Запрограммируйте маршрут.
```

Ничего не понятно. Все эти цветные квадратики наверняка что-то значат… Читай дальше – и всё поймёшь. Но сначала ты должен узнать, из чего сделан компьютер.

Кремниевый разум – *из чего сделаны компьютеры*

Вокруг Лудолфа, насколько он мог видеть, простирался фантастический мир. Робот тщательно документировал всё вокруг – от ослепительных камней до сверкающих песчинок. Он пытался понять, почему всё именно так, а не иначе. Почему песок такой сверкающий, почему озёра зелёные, а не голубые и почему камни фиолетовые, а не серые? На все эти вопросы нужно ответить. И у робота есть свой верный способ сделать это.

Большинство комет – это просто большие комья из снега и пыли. Но некоторые образовались, отколовшись от старых планет. Такие кометы в основном сложены горными породами, состоящими из химических элементов, например кремния и кислорода, атомы которых, в свою очередь, состоят из протонов, нейтронов и электронов. Живые организмы – деревья, черепахи и люди – имеют такие органы, как глаза, панцири, носы. Органы состоят из тканей, таких как кости или кожа, а ткани состоят из простых клеток. Клетки – строительные кирпичики жизни – состоят из ещё более простых веществ, это белки и сахара. А те состоят из атомов, как и всё остальное. Это законы физики, которые применимы и к Земле, и к комете, и везде во Вселенной.

Знание, из чего сделаны вещи и как они связаны друг с другом, – важный шаг к их пониманию. Однако целое всегда больше, чем сумма его частей. Дом – это не просто груда кирпичей, а место для жизни, а собака – не просто большой клок шерсти, а лучший друг.

Ты уже знаешь, что с роботами можно общаться, используя язык программирования. Язык программирования состоит из **команд**. Команды компи-

лируются в двоичный код (нули и единицы). После этого двоичный код преобразуется в физические сигналы (например, световые или электрические импульсы).

Даже для программиста, который не является глупым одноклеточным организмом, процесс преобразования сигналов в команды достаточно сложен, и при его выполнении легко ошибиться. Насколько умной должна быть машина, чтобы решить эту задачу без ошибок?

Роботы

Роботы не так сложны, как может показаться на первый взгляд. Их «органы» можно пересчитать по пальцам одной руки!

Робот – это компьютер, имеющий дополнительные инструменты для выполнения практических задач, таких как замешивание теста (лопасти миксера) или написание домашнего задания по математике (руки с пальцами, чтобы держать карандаш).

Ты без труда можешь представить себе лопасти миксера и руки, но что представляет собой *компьютер*?

Компьютеры

Первое, что приходит на ум, – это ноутбук, например такой, как у тебя дома. Ты можешь даже понимать, что твой смартфон – это тоже небольшой компьютер. Все эти компьютеры могут решать любые вычислительные задачи, а значит, они являются *универсальными* (также известными как компьютеры *общего назначения*).

Универсальные роботы – это *универсальные компьютеры*, а специализированные роботы (например, кухонный комбайн) – это *специализированные компьютеры*.



Для любознательных

Изобретателем универсального компьютера был английский математик Алан Тьюринг.

В интернете ты без труда найдёшь дополнительную информацию об этом герое и его печальной судьбе.

Таким образом, **компьютер** – это машина, способная решить любую *вычислимую* задачу (то есть задачу, имеющую решение). А поскольку всякую задачу

можно закодировать числами (нулями и единицами), то вместо «решить» можно сказать *«вычислить»*.

Итак, компьютер – это *машина, которая может вычислять*. Кто бы мог подумать?!

Поскольку роботы – это просто компьютеры, оборудованные дополнительными механическими инструментами, такими как руки и ноги, то всё, что до сих пор говорилось о роботах, применимо ко всем компьютерам. Все компьютеры, как и роботы, общаются друг с другом посредством сигналов и могут программироваться с помощью языков программирования.



Задача № 8

Как называется крупнейшая в мире сеть, соединяющая миллионы компьютеров, по которой электрические и световые сигналы бегут, словно по шоссе, из одного конца мира в другой?

Интегральные схемы

Современные компьютеры сконструированы из *электронных компонентов*, называемых **интегральными схемами**, или *чипами*. Интегральные схемы – это такие «клетки», из которых состоят «ткани» электронных машин.

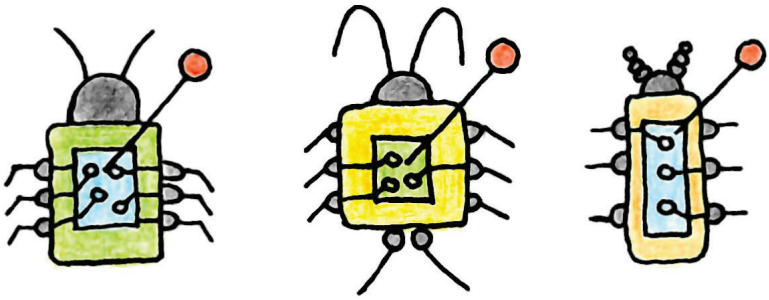
Без сомнения, изобретение интегральной схемы является величайшим открытием двадцатого века. Они присутствует практически во всём: в компьютерах, смартфонах, стиральных машинах, кухонных роботах, радиоприёмниках и автомобилях. Трудно представить себе мир без интегральных схем.



Задача № 9

Какое открытие, сравнимое по значимости с интегральной схемой, было сделано в восемнадцатом веке?

Заглянув внутрь компьютера, можно обнаружить интегральные схемы, напоминающие забавных зелёных жуков.



Интегральная схема соединяет в себе комбинацию простых электронных компонентов, которая выполняет основные операции, такие как сложение и вычитание.

Интегральные схемы *объединяются вместе* для создания ещё более интеллектуальных функций. Например, из компонентов, выполняющих сложение, можно собрать схему, выполняющую умножение. Однако все эти схемы взаимодействуют друг с другом с помощью электрических сигналов, представляющих собой старый добрый двоичный код.

Для любознательных

Одна из самых простых и полезных интегральных схем – это логическая функция **NAND** (*NOT-AND – НЕ-И*). Она состоит всего из пяти компонентов, и с её помощью можно собрать любую другую вычислительную схему.

Это означает, что сложение, вычитание, умножение, деление и все другие математические операции можно выполнить с помощью набора простых схем NAND!

Например, чтобы сложить два числа, достаточно собрать интегральную схему из девяти схем NAND.

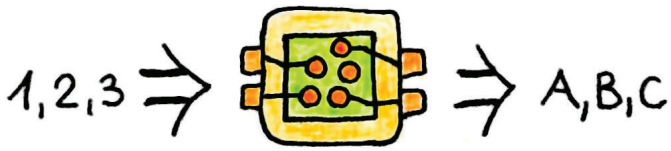
Ввод и вывод

Почти все компьютеры (включая роботов) имеют интегральную схему для **ввода и вывода**. Она *принимает* сигналы на входе и *отправляет* сигналы на выход.

Примерами устройств ввода могут служить антенна, клавиатура или сенсорный экран, а примерами устройств вывода – наушники или динамики.

Компьютер можно представить как чёрный ящик, получающий сигналы на входе, вычисляющий закодированные в них задачи и посылающий результаты на выход.

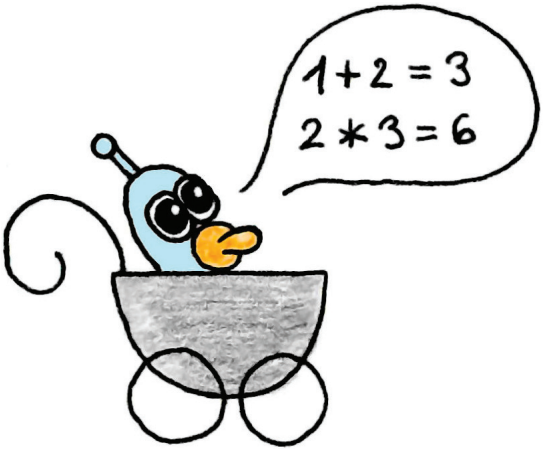
Без устройства ввода и вывода любой компьютер был бы бессмысленным, потому что он не смог бы принимать исходные данные для решения задач и выводить полученные результаты.



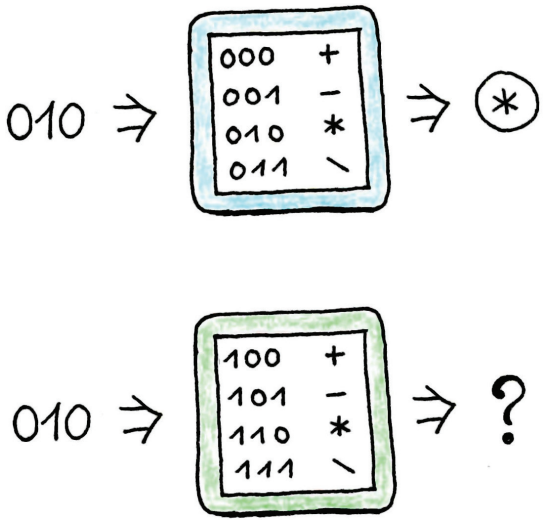
Процессор

Процессор, или **ЦП** (*центральное процессорное устройство – Central Processing Unit, CPU*), обычно является самой сложной интегральной схемой любого компьютера. Именно в нём происходят все вычисления.

Схема процессора выполняет основные математические и логические операции, такие как сложение и умножение. Вот почему компьютеры так хороши в математике: для них это так же естественно, как дышать для человека.



Помимо математических функций, процессор содержит *набор команд*, которые называются **инструкциями** и различаются в зависимости от вида процессора. Именно поэтому одна и та же программа не может работать и в роботе Лудолфе, и в стиральной машине. Эти две машины имеют разные процессоры с *разными наборами инструкций*. Программа для разных процессоров должна быть скомпилирована с помощью разных компиляторов, как описано в главе о программах.



Компиляторы – это, по сути, всего лишь очень сложные программы. Но как написать программу-компилятор без компилятора? Самый первый компилятор был программой, написанной на языке инструкций процессора непосредственно в виде двоичного кода. Создание такого компилятора потребовало немало усилий, хотя он был довольно прост и транслировал программы на очень простом языке программирования. Затем, используя этот компилятор, уже можно было написать другой для более сложного языка программирования, а тот – для ещё более сложного и интеллектуального языка программирования и т. д.

Нет ничего необычного в том, что компилятор для более поздней версии данного языка программирования написан на более ранней версии того же языка программирования.



Для любознательных

ЦП – универсальный процессор, способный выполнять любые вычислительные задачи. Но некоторые *специальные вычисления* слишком сложны для него, поэтому были разработаны *специализированные процессоры*, способные выполнять лишь несколько конкретных типов вычислений, зато очень хорошо и быстро.

К наиболее известным специализированным процессорам относятся **видеокарты**, известные как **графические процессоры** (**Graphics Processing Unit**, **GPU**). Видеокарты отлично справляются с вычислительными задачами, связанными с отображением графики (например, в компьютерных играх). Но они почти бесполезны для чего-то ещё, поэтому всегда присутствуют в компьютере только вместе с ЦП.

И наконец, процессоры могут *управлять* другими компонентами и устройствами, такими как устройства ввода и вывода, а также памятью компьютера.



Задача № 10

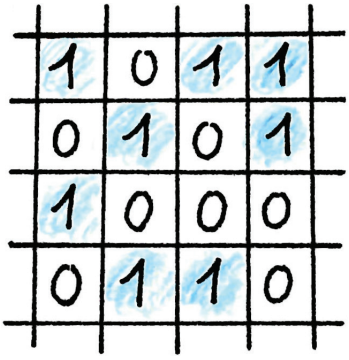
Подумай вот над чем: все мальчики и девочки умеют считать с первого класса. Означает ли это, что люди тоже являются компьютерами?

Память

Одного умения выполнять арифметические операции недостаточно. Представь, что тебе нужно вычесть одно число из другого. Но, прочитав второе число, ты забываешь первое. На самом деле такого не происходит, потому что человеческий мозг может запоминать – он имеет *память*.

Память – это способность сохранять *данные* для последующего использования. Память компьютера можно представить как ленту с нулями и единицами, которые можно легко стереть или перезаписать.

Вместо карандаша и ластика электронные вычислительные машины (такие как Лудолф или смартфон) используют *электрический заряд*. Ячейка памяти имеет заряд (один) или разряжена и не имеет заряда (ноль).



Процессор выполняет вычисления над данными, хранящимися в памяти, и сохраняет их результаты.



Задача № 11

Перерисуй и увеличь сетку памяти компьютера до размера 8×8 квадратов и сохрани в ней числа 2 и 3 из бинарного алфавита Лудолфа.

Что произойдёт с этой памятью после того, как процессор перемножит эти числа?



Для любознательных

Компьютеры, как и люди, имеют кратковременную и долговременную память. Кратковременная память называется **ОЗУ** (оперативное запоминающее устройство), и все данные, хранящиеся в ней, теряются при выключении компьютера. А информация в долговременной памяти сохраняется даже после перезагрузки компьютера.

Примерами долговременной памяти являются жёсткие диски, флеш-накопители и даже карты памяти фотоаппаратов (в конце концов, современные фотоаппараты – это тоже компьютеры).

Как программировать роботов

Странный мир – *введение в программирование*

Вселенная – поистине удивительное место. Нам даже не нужно покидать свою планету, чтобы обнаружить очень необычных существ. Бактерии процветают в кипящей воде близ вулканических кратеров на дне океана, одноклеточные водоросли развиваются в арктических льдах, а бархатные губки прекрасно себя чувствуют в радиоактивных резервуарах на атомных электростанциях. Но представь, насколько ещё более странные и удивительные организмы можно найти в далёких мирах.

Жизнь обладает удивительной способностью адаптироваться к окружающей среде и всегда находит способ выжить. Горячая вода из подводного вулкана токсична для людей, а обычный воздух токсичен для бактерий, которые обитают в воде близ вулкана. Чем более необычен мир, тем необычнее существа, живущие в нём.

Наконец-то ситуация начинает обретать смысл! Теперь ты знаешь, как работают компьютеры, и можешь начать программировать их, чтобы вернуть Лудолфа домой.



Задача № 12

Перечитай оглавление первой части книги и постарайся вспомнить наиболее важное из каждой главы.

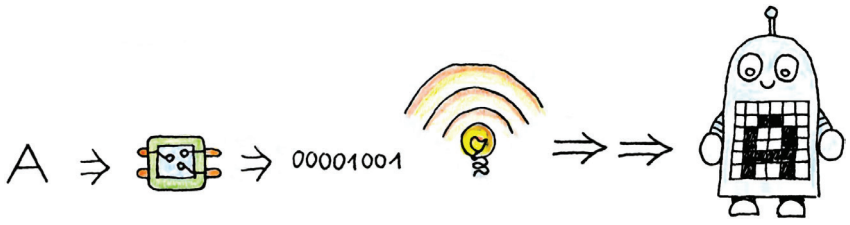
Лудолф – это робот. Роботы – это компьютеры. Компьютеры – это машины, использующие вычисления для решения задач.

В мире существует множество машин, которые используют различные процессоры с разными наборами инструкций. Эти инструкции кодируются двоичным кодом (*нулями и единицами*).

Поскольку программирование в двоичном коде – слишком сложная задача для большинства людей, со временем было создано множество языков программирования, которые имеют некоторое сходство с человеческой речью. Текст, написанный на этих языках, транслируется в двоичные инструкции процессора с помощью специальных программ, называемых компиляторами. Программа – это набор инструкций, описывающих порядок решения определённой задачи.

Компьютеры используют (электрические) сигналы для общения с окружающей средой, включая людей и самих себя. Сигналы преобразуются в двоичный код и обратно в сигналы с помощью устройств ввода и вывода. На компьютере вся информация, данные и программы хранятся в двоичном коде. Компьютеры могут «думать» только в двоичном коде.

Память компьютера – это место, где хранятся данные и откуда они загружаются.



Языки программирования

В мире существует множество языков программирования. Некоторые из них настолько просты, что содержат лишь несколько команд. На этих странных языках можно написать любую программу, но для этого потребуется много сил и времени:

```
+++++[>+++++[>+++[>+>+>+<<<-]<-]>>->++<<<<-]
>>>-->>+.<.<.>>+++.---.<<++.
```

Программы, написанные на Python и JavaScript, читаются намного проще:

```
message = "Привет, Мир!"
print(message)
```

```
var message = "Привет, Мир!"
console.log(message)
```

Это примеры знаменитой программы «Привет, Мир!» на разных языках программирования. Программа просто выводит на экран строку `"Привет, Мир!"` и завершает работу. Помимо программы, которая вообще ничего не делает, это самая простая из возможных программ. Именно поэтому она приводится в самом начале большинства учебников по программированию.



Задача № 13

Внимательно рассмотри примеры программ выше и попробуй ещё больше упростить их.

LudolfC

Лудолф – это особый компьютер со своим собственным языком программирования, который называется **LudolfC**. В следующих главах ты будешь учиться программировать на языке LudolfC, чтобы в конечном итоге переписать программу управления робота и спасти его с далёкой кометы.

Посмотри, как выглядит программа «Привет, Мир!» на языке программирования LudolfC:

```
message := "Hello World!"
Display(message)
```

Панель управления доступна по следующему адресу в интернете:

ludolf.github.io.

В оставшейся части книги ты будешь писать код в этой панели. Если возникнет какая-то ошибка, то просто удали код, обнови веб-страницу и начни сначала. Не бойся экспериментировать, в этом нет ничего плохого!

Понемногу обо всём – *типы данных*

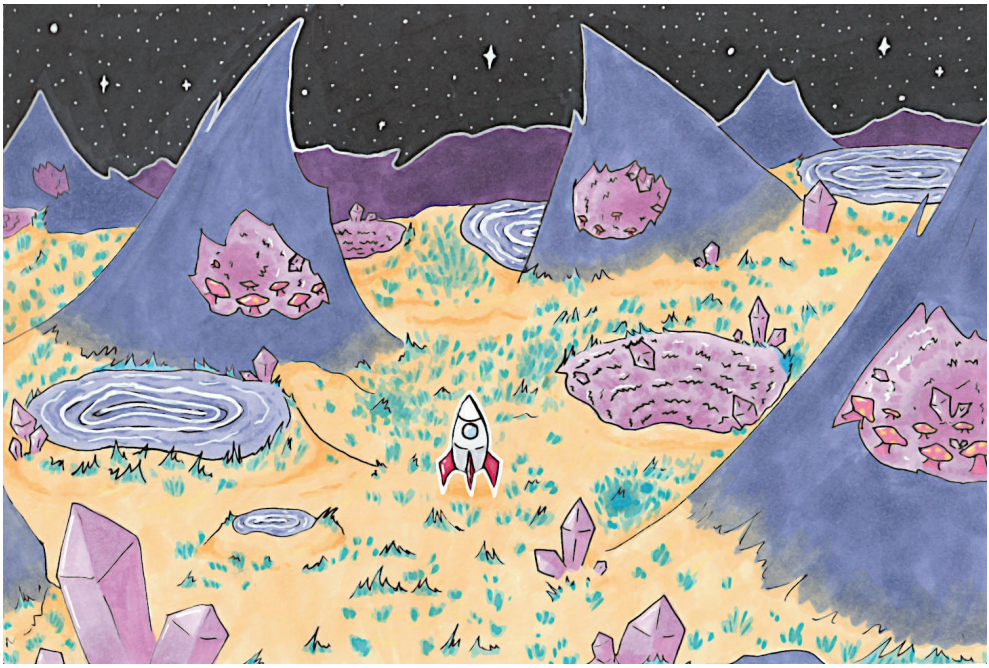
Столько дел – а времени мало! Лудолф продолжает работать без перерыва с момента посадки на комету. Каждый камень, каждая песчинка будут сфотографированы, измерены, взвешены и занесены в базу данных. Он также классифицирует все образцы по их характеристикам. От его внимания не ускользнут и крупные образования, такие как скалы, расщелины и озёра. Всё будет описано и классифицировано. Робот движется вперёд, не пропуская ни сантиметра местности. При этом он измеряет температуру, радиацию и состав атмосферы.

Но что это? Некоторые объекты явно собираются в группы и двигаются вместе. Робот только что обнаружил доказательство существования инопланетной жизни! И пусть это не маленькие зелёные человечки в летающих тарелках, а микроскопические организмы, они всё равно великолепны. Похоже, что они питаются пылью, которая время от времени падает сверху, словно майский дождь. При ближайшем рассмотрении становится ясно, что жизнь на комете выбрала совершенно иной путь развития, чем на Земле. Основу их молекул, как и молекул микросхем Лудолфа, составляет широко распространённый кремний, и поэтому они демонстрируют радикально иные свойства и поведение, чем живые организмы на Земле.

Перевернув страницу руководства, ты натыкаешься на следующую таблицу:

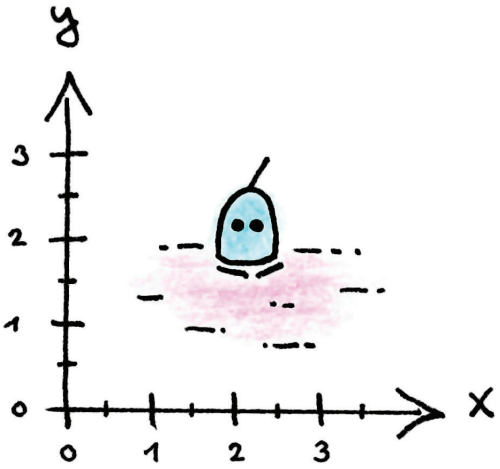
Число	Описание
0	песок
20	ракета
30	кратер
40	энергия

Так вот что это такое! На маленьком экране изображена карта местности, где сейчас находится Лудолф. Ты пытаешься представить, как эта местность может выглядеть в реальности.



Программы обычно – но не всегда – обрабатывают данные. Данные могут быть числами, которые можно превратить в пиксели и отобразить на экране. Данные могут быть замерами температуры, координатами текущего местоположения Лудолфа и массой образцов. Даже карта, отображаемая в данный момент на экране, – это данные.

Данные могут быть разных типов. Например, числа можно складывать, а текстовые сообщения – нет. Существуют простые данные, такие как числа, и составные данные, такие как координаты местоположения, состоящие из двух или более числовых элементов, *x* и *y*.



Тип данных сообщает компьютеру, с какими именно данными он имеет дело. Например, банан – это фрукт, а Лудолф – робот. Тип данных определяет их характеристики.

Каждый язык программирования имеет свой набор базовых типов данных, таких как числа и текст. Кроме того, языки программирования позволяют программистам создавать свои типы данных, составленные из базовых типов. Например, тип данных `Coordinate`, описывающий координаты точки на плоскости, будет состоять из двух элементов, `x` и `y`, каждый из которых относится к типу числовых данных *Number*.

Тип данных подсказывает, что можно и что нельзя делать с данными. Числа можно складывать, вычитать, умножать и делить, координаты можно перемещать в обоих направлениях, фрукты можно есть, а робота можно отправить в космическую экспедицию и т. д.



Для любознательных

Некоторые языки программирования требуют, чтобы программист указал тип для каждого фрагмента данных, используемого в программе. Такие языки называются *статически типизированными*.

Компиляторы некоторых языков программирования могут автоматически определять тип данных по операциям, которые к ним применяются. Такие языки, к которым относится и LudolfC, называют языками с *динамической типизацией*.

Числа

Числа не требуют особых пояснений: `0`, `1`, `2`, `3`, `16`, `256`, `-1` и `14,5` – всё это числа.

В таблице ниже показаны математические операции, которые можно выполнять с числами:

Код	Операция	Пример	Результат
`+`	сложение	`1 + 2`	`3`
`-`	вычитание	`2 - 1`	`1`
`*`	умножение	`2 * 3`	`6`
`/`	деление	`4 / 2`	`2`
`%`	остаток от деления	`5 % 3`	`2`
`=`	сравнение	`3 = 3`	`True`

Операция **вычисления остатка от деления** (или, как её ещё называют, **деление по модулю**) представляет особый интерес для программистов. Результат показывает, что осталось после деления нацело на число справа. Например, если у тебя имеется десять пряников и ты хочешь поделить их между тремя друзьями, то друзья получат по три пряника, и у тебя ещё останется один пряник:

```
10 % 3 = 1
```

Вычисление остатка от деления помогает гарантировать, что результат никогда не превысит делитель, потому что остаток не может быть больше количества частей, на которые делится начальное число (делимое). Если в результате деления у тебя останется больше пряников, чем друзей, то остаток можно снова разделить между ними и повторять раздел, пока число пряников не станет меньше, чем число друзей.

Если после раздела ничего не осталось, то результат равен нулю. Деление по модулю исправит завышенные числа, приведя их значения в соответствующие пределы:



```
0 % 5 = 0
1 % 5 = 1
2 % 5 = 2
3 % 5 = 3
4 % 5 = 4
5 % 5 = 0
6 % 5 = 1
```



Задача № 14

Какими будут результаты следующих вычислений?

```
5 % 4
55 % 5
222 % 10
```

Текст

Текст – это последовательность символов, заключённая в кавычки. Например, `"Привет, Мир!"`.

Тексты могут быть любой длины, и к ним могут применяться операции объединения и сравнения:

Код	Операция	Пример	Результат
`+`	объединение	`"A" + "B"`	`"AB"`
`=`	сравнение	`"A" = "A"`	`True`

Текстовые данные можно также объединять с нетекстовыми данными, например с числами:

```
text := "abc" + 123
```

В результате получится текст `"abc123"`.



Первый операнд в операции + определяет тип данных результата. В примере выше + выполняет операцию объединения двух текстовых значений, но если первым операндом будет число, то + попытается выполнить операцию сложения чисел. Если за знаком + окажется текст, то программа остановится и выведет сообщение об ошибке, потому что сложение числа с текстом не имеет смысла.



Задача № 15

Какими будут результаты следующих операций?

```
"a" + 0 + "b"
"a" + 1 + 2
"" + 1 + 2
1 + "a"
```

Истинность

Истинность – это, пожалуй, самый необычный тип данных в программировании. Но не бойся: он очень простой и очень полезный.

Истинность – это тип данных, получаемых при выполнении *логических* операций, которые могут иметь только два значения: *истинно* (true) или *ложно* (false). Это похоже на игру, в которой на вопросы можно отвечать только «да» или «нет». Такие вопросы типа «да/нет» называются *выражениями* и относятся к типу данных «истинность».

Код	Значение	Пример
`True`	истинно	`1 < 2`
`False`	ложно	`1 = 2`

Тип данных истинности является частью области математики, известной как *логика*. В логике имеются операции, которые часто используются в компьютерных программах:

Код	Значение	Пример
`&`	… и …	`True & False`
`|`	… или …	`True | False`
`!`	не …	`!True`
`=`	… равно …	`True = True`

Логическое умножение (&) полезно, когда нужно, чтобы *несколько* условий выполнялись одновременно. В обычной человеческой речи логическое умножение выражается союзом *«и»*. Например, *«если идёт дождь и светит солнце, появляется радуга»*. В этом случае *«идёт дождь»* – одно условие, а *«светит солнце»* – другое. Радуга появится, только если оба условия выполняются (истинны).

Это составное выражение можно записать с помощью логического умножения так:

если идёт дождь **&** светит солнце.

Чтобы составное выражение давало истинный результат, все содержащиеся в нём подвыражения (условия) должны выполняться (быть истинными).



Задача № 16

Заполни следующую таблицу логического умножения:

Выражение	**Результат**
`True & True`	`True`
`True & False`	`False`
`False & True`	`?`
`False & False`	`?`

Логическое сложение (|) удобно, когда нужно, чтобы выполнялось хотя бы одно из условий (было истинным). В повседневной речи логическое сложение выражается союзом *«или»*. Например, *«если на улице дождь или снег, то лучше остаться дома в уюте и тепле»*. Достаточно дождя или снега (или и того, и другого), чтобы захотелось остаться дома. Но если нет ни дождя, ни снега, то можно с радостью выйти погулять.

Это составное выражение можно записать с помощью логического сложения так:

если на улице дождь | снег.



Задача № 17

Заполни следующую таблицу логического сложения:

Выражение	**Результат**
`True | True`	`True`
`True | False`	`True`
`False | True`	`?`
`False | False`	`?`

Логическое отрицание (!) изменяет значение на противоположное. Оно превращает истину в ложь, а ложь – в истину. В повседневной речи логиче-

ское отрицание выражается частицей *«не»*. Например, отрицание условия «удобный» – это «не удобный». На языке программирования логическое отрицание записывается так:

! удобный.

Логические операции отрицания можно комбинировать как угодно и тем самым менять смысл условий на противоположный. Поскольку тип данных истинности имеет только два возможных значения, два отрицания, идущие подряд, взаимно уничтожаются. Аналогично двойное переворачивание монеты (с роботом на аверсе и кометой на реверсе) даёт тот же результат, что и не переворачивание.



Таким образом, двойное отрицание условия «удобный» снова превращает его в «удобный». Вот как это выглядит:

!! удобный
=
не ! удобный
=
не не удобный
=
удобный.



Задача № 18

Заполни следующую таблицу логического отрицания:

Выражение	**Результат**
`! True`	`False`
`! False`	`True`
`!! True`	`?`
`!! False`	`?`

Тип данных истинности встречается намного чаще, чем можно было бы ожидать. При сравнении данных любых типов результат всегда будет иметь значение типа истинности. Это означает, что результат может быть либо «истинно», либо «ложно», без промежуточных значений:

`1 = 1` истинно

`1 = 2` ложно

`1 < 2` истинно

`1 >= 2` ложно.

Таким образом, тип данных истинности является типом данных, выражающим все условия, которые являются базовыми строительными блоками компьютерных программ. Подробнее об условиях мы поговорим в следующих главах.



Для любознательных

В большинстве языков программирования тип данных истинности называется *булевым* в честь мистера Джорджа Буля, который в девятнадцатом веке открыл и разработал мощную вычислительную систему со значениями истинности и логическими операциями, получившую название *булева алгебра*.

Составные типы данных

Составные типы данных – это структуры, содержащие несколько элементов других типов данных. Составные типы данных объединяют связанные друг с другом данные, помогая организовать код. Они также часто добавляют к составным данным новые практические свойства.

Разные языки программирования предлагают разные способы представления составных типов данных. В LudolfC данные могут объединяться в *массивы* и *объекты*. Оба типа будут обсуждаться в последующих главах.

Порядок операций

Как ты наверняка помнишь, некоторые математические операции имеют *приоритет* над другими. Если в одном выражении имеются умножение и сложение, то умножение всегда выполняется первым независимо от того, на каком месте оно находится в выражении. Это свойство, также известное как *порядок* или *предшествование*, определяет порядок выполнения действий.

Когда компьютер вычисляет выражение, состоящее из нескольких операций, он сначала определяет их порядок, а затем просто выполняет вычисления, следуя этому порядку. Например, при вычислении выражения «1 + 2 * 3» сначала выполняется умножение «2 * 3», а затем полученный результат прибавляется к единице:

```
1 + 2 * 3
  1 + 6
    7
```

Если бы компьютер *не* учитывал порядок операций, он бы получил неверный результат:

```
1 + 2 * 3
  3 * 3
    9
```

Порядок операций можно изменять, заключая подвыражения в круглые скобки. Операции в круглых скобках всегда выполняются первыми:

```
(1 + 2) * 3
  (3) * 3
    9
```



Задача № 19

Вычисли результат следующего выражения:

```
1 + 2 * (2 + 1)
```

Логические операции также имеют определённый приоритет. Логическое умножение & должно выполняться раньше логического сложения |.



Задача № 20

Вычисли результат следующего выражения:

```
True | False & False
```

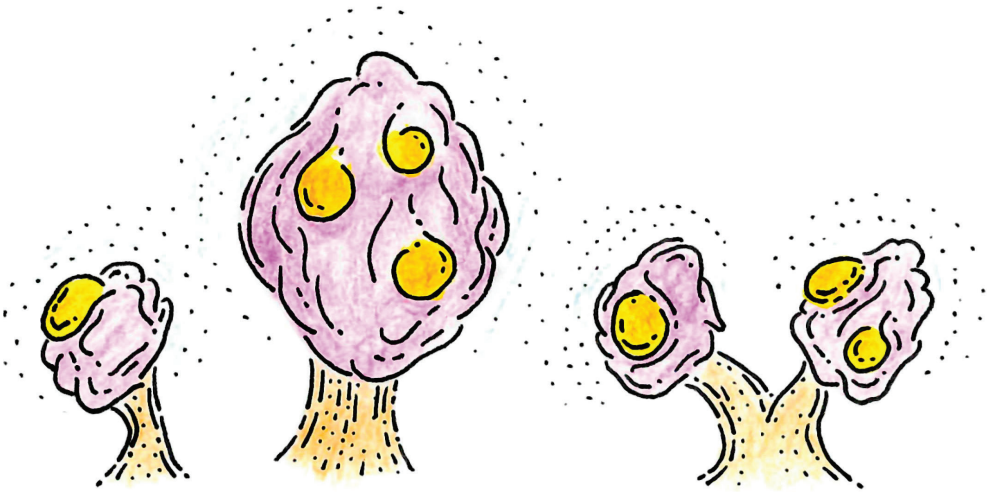
Упражнение

Для выполнения следующего упражнения открой веб-сайт **ludolf.github.io** на своём компьютере или смартфоне. Запусти программу, щёлкнув по кнопке с галочкой. После запуска программы в нижней области на экране отобразится результат.

1. Напиши программу, которая отображает число `1`.
2. Напиши программу, которая отображает число `123`.
3. Напиши программу, которая отображает число `1.5`.
4. Напиши программу, которая отображает результат вычисления суммы `1 + 2`.
5. Напиши программу, которая выводит текст `"abc"`.
6. Напиши программу, которая выводит результат сравнения `1 < 2` (один меньше двух).

И то, и это – *переменные*

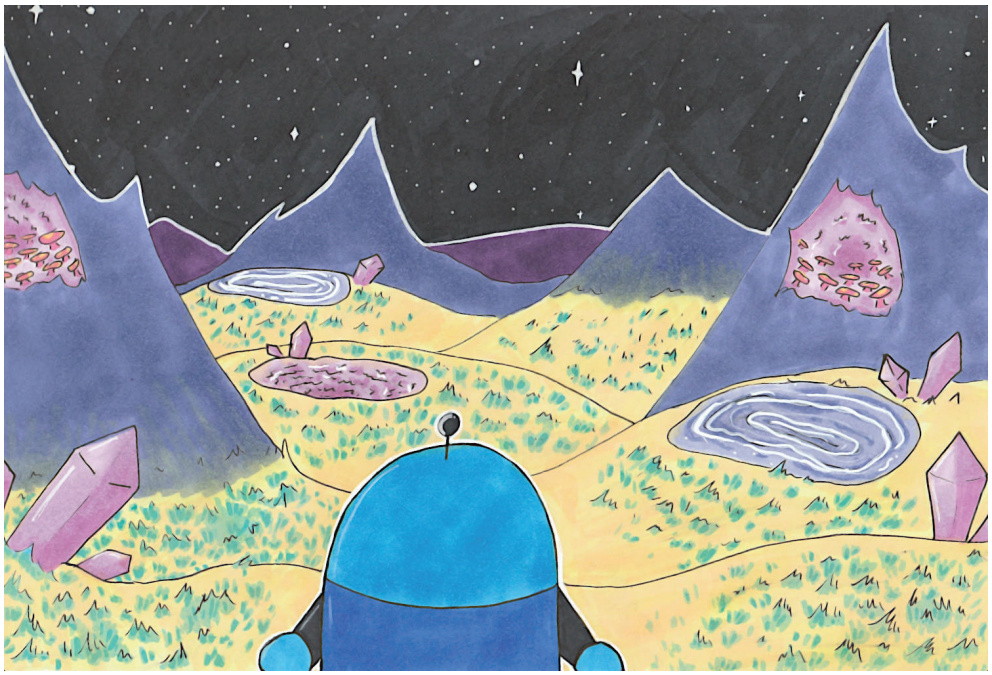
Чем дальше Лудолф удаляется от места высадки, тем более разнообразным и странным выглядит пейзаж. На первый взгляд всё кажется знакомым, но при этом каким-то другим и чуждым. При прикосновении к фиолетовым кристаллам они издают звуки, похожие на звучание струн арфы. Песок, подобно горным ручьям, плавно течёт длинными извилистыми зигзагами. В отдалённых оазисах синяя трава окружает метановые озёра, наполненные мельчайшими организмами, напоминающими земных простейших, и водорослями. Красивые пурпурные грибы освещают скальные пещеры, излучая ослепительное сияние, то затухая, то вновь разгораясь через регулярные промежутки времени, словно в такт дыханию. Проведя измерения, робот выяснил, что грибы излучают радиоактивное ультрафиолетовое излучение, улавливаемое кристаллами.



Описывая что-то в реальной жизни или в компьютерной программе, всегда следует использовать правильные названия.

Синие растения, например, можно назвать *Bluestellar Blades* (лучи синей звезды), а музыкальные камни – *Opera Ores* (поющие кристаллы).

Чтобы дать чему-то название, нужно это что-то осмыслить.

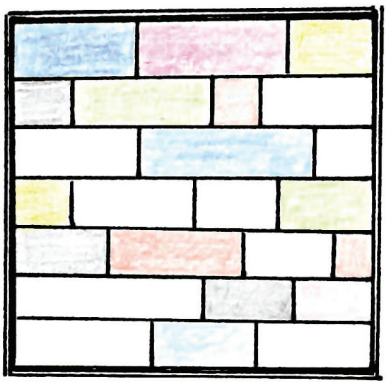


Что такое переменные?

Помнишь, что такое *компьютерная память*? Память – это место, где хранятся данные для процессора. Процессор, выполняя программу, читает данные из памяти и записывает их в память. Для доступа к памяти компьютерные программы используют переменные.

Переменная – это именованное местоположение в памяти компьютера. Переменные могут использоваться для загрузки или изменения данных в памяти. Создание новой переменной помещает новые данные в новую позицию в памяти.

Если представить память компьютера в виде шкафа, то переменные можно сравнить с подписанными ящиками, внутри которых хранятся какие-то предметы.



Как создаются переменные

Для создания переменной с заданным именем и значением в языке LudolfC используется команда **присваивания** (`:=`):

```
имя := значение
```

Следующий фрагмент кода создаёт в памяти две переменные с именами `x` и `y`. Переменная `x` имеет числовой тип данных, а переменная `y` имеет текстовый тип:

```
x := 123
y := "Привет, Мир!"
```



Не путай присваивание `:=` со сравнением `=`, которое не создаёт новую переменную, а просто сравнивает два значения: «1 = 2».



Для любознательных

В некоторых языках программирования операция присваивания значения переменной записывается как один символ «равно» (=), а операция сравнения – как двойной символ «равно» (==).



Задача № 21

Что получится в результате вычисления следующих выражений?

```
x := 1
x = 2
```

Имена переменных

Имя переменной может состоять из любых комбинаций букв, цифр и символа подчёркивания, но при этом не должно начинаться с цифры. Значение переменной – это значение любого доступного типа данных:

```
my_number_1 := 1
MyTEXT := "abc"
_true := True
```



Для любознательных

В разных языках программирования распространены разные стили написания имён переменных.

При использовании *верблюжьего регистра*, одного из наиболее распространённых стилей, имена переменных начинаются со строчной буквы, а каждое следующе слово в имени начинается с заглавной буквы. Вот пример использования верблюжьего регистра для записи имени переменной: `camelCase`. Заглавные буквы в имени напоминают горбы верблюда.

Другой популярный стиль – *змеиный регистр*. При использовании этого стиля имена переменных состоят только из строчных букв, а слова отделяются подчёркиванием, например `snake_case`. Такие имена напоминают змейки, составленные из сегментов.

Верблюжий регистр распространён в языках программирования JavaScript и Java, а змеиный регистр – в Python (как ты уже мог видеть выше).

Имя переменной, начинающееся с заглавной буквы, показывает, что переменная является какой-то особенной.

Большинство примеров в этой книге написаны с использованием змеиного регистра. Однако ты в своих программ можешь использовать тот стиль, какой тебе больше по душе!



Задача № 22

Определи, какие имена переменных можно использовать в языке LudolfC, а какие нет:

```
a
A
ab
a b
a_A
_ab
a__b_
1
a1
1a
a12b
a%
a#b
```

Значения переменных

Значение переменной можно изменить. При этом содержимое соответствующего местоположения в памяти будет перезаписано и исходное значение будет потеряно:

```
my_variable := 1
my_variable := 2
```



Задача № 23

Какое значение будет иметь переменная x в конце программы?

```
x := 1
x := 34
x := 256
```

Использование имени переменной в любом месте программы равнозначно использованию текущего значения переменной. Следующие две программы абсолютно одинаковы:

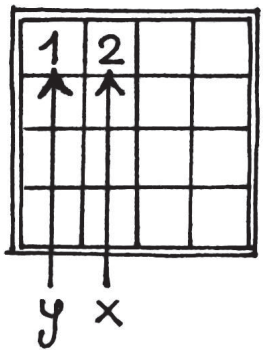
```
x := 1
y := x + 2
z := x + y + 4
```

```
x := 1
y := 1 + 2
z := 1 + 3 + 4
```

Когда создаётся новая переменная, программа всегда выделяет для неё новое место в памяти, куда можно поместить присваиваемое ей значение. В результате каждая переменная имеет своё отдельное пространство в памяти и не зависит от других переменных.

Несмотря на то что значение переменной `x` изменилось и стало равно `2`, значение переменной `y` в конце следующей программы осталось равным `1`:

```
x := 1
y := x
x := 2
```





Задача № 24

Какие значения будут иметь переменные `x` и `y` в конце следующей программы?

```
x := 1
y := x + 1
x := x + y
```

Использование переменных

Итак, зачем нужны переменные? Почему бы просто не записывать их значения в коде программы? Дело в том, что в больших программах легко забыть текущее значение! Кроме того, значение переменной может быть неизвестно программе, например, если оно поступает с устройства ввода:

```
outside_temperature := ?
what_time_it_is := ?
```

Иногда значения могут быть настолько длинными или сложными, что записывать их везде в таком виде будет просто неудобно. В следующем примере переменная `Pi` содержит значение числа пи, используемого Лудолфом для вычисления длины окружности и площади круга:

```
Pi := 3.14159265358979323846264433

r := 25
c := Pi * 2 * r
a := Pi * r * r
```

Переменные также могут участвовать в выражениях, результат которых присваивается этим же переменным. Это часто делается при увеличении или уменьшении значения переменной:

```
x := 1
x := x + 1
```



Для любознательных

Процесс создания переменной путём присваивания ей значения называется *инициализацией*.



Каждая переменная должна быть инициализирована перед использованием!

Перед первым использованием переменной ей обязательно нужно присвоить какое-то значение, иначе программа завершится с ошибкой:

~~`w := w + 1`~~ ⇒ ERROR!



Задача № 25

Какое значение будет иметь переменная `x` в конце программы?

```
x := 100
x := x + 1
x := x + x
x := x - 1
```



Для любознательных

Теперь ты знаешь, как создавать переменные, но как от них избавиться? Каждая переменная занимает определённый объём памяти, который не бесконечен. Было бы хорошо своевременно освобождать место, удаляя ненужные переменные.

В старых языках программирования, таких как C и C++, программист мог сразу же удалить ненужные переменные и освободить занимаемую ими память. Однако программисты – всего лишь люди, а люди несовершенны. Иногда они забывали удалить ненужные переменные, и тогда в процессе выполнения программы под них снова и снова выделялось место, пока свободное место в памяти не заканчивалось и программа не завершалась с ошибкой.

В современных языках программирования, таких как JavaScript, Python и LudolfC, в среду выполнения программы добавлены специальные сборщики мусора. Они регулярно проверяют память и, обнаружив неиспользуемое пространство, быстро освобождают его. *Сборщик мусора* – это просто вспомогательная программа, которая работает параллельно с основной программой.

Сборщики мусора избавляют программиста от необходимости беспокоиться об освобождении памяти, поскольку оно выполняется автоматически и без его участия.

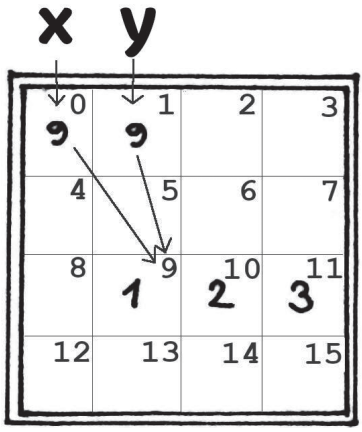
Указатели

Составные данные вносят некоторые сложности, которые отсутствуют при работе с базовыми типами данных. Составные типы данных – это наборы из нескольких значений, располагающихся вместе в памяти компьютера, например в первых трёх ящиках в первом ряду сверху.

Когда переменной присваивается значение составного типа данных, например массива или объекта, то ей просто присваивается *адрес* этого расположения. Этот адрес называется **указателем**, потому что он указывает на расположение в памяти со значениями.

При присваивании такой переменной другой переменной присваивается только адрес указателя на новую переменную, а не значение, на которое она указывает:

```
x := [1, 2, 3]
y := x
```



На рисунке показаны две переменные, `x` и `y`, которые хранят адрес одного и того же массива (подробнее о массивах я расскажу в следующей главе). Если изменить значение элемента массива, используя переменную `x`, то это изменение можно будет заметить при использовании переменной `y`, потому что они обе содержат один и тот же адрес и, соответственно, указывают на одно и то же место в памяти.

Это всё равно что положить мелки во второй ящик в первом ряду сверху. Любой, кто заглянет в ящик, следуя описанию его местоположения, найдёт там мелки:

```
x := ["toys", "socks", "books"]
y := x

y[1] := "crayons"

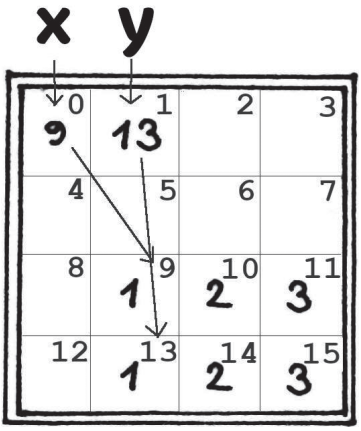
x = ["toys", "crayons", "books"]
y = ["toys", "crayons", "books"]
```

При записи полного значения для него выделяется место в памяти, а адрес этого места присваивается новой переменной. Как и отдельные дома на улицах, эти переменные не зависят друг от друга:

```
x := ["toys", "socks", "books"]
y := ["toys", "socks", "books"]

y[1] := "crayons"

x = ["toys", "socks", "books"]
y = ["toys", "crayons", "books"]
```



В следующей главе я расскажу о массивах. И, поняв, что такое массивы и как с ними работать, ты можешь вернуться сюда, чтобы ещё раз прочитать раздел об указателях и лучше понять их.

В программах можно не только создавать свои переменные, но и пользоваться так называемыми *встроенными переменными*. В LudolfC к таким переменным относятся, например, `Ludolf`, `Map` и `Display`. Подробнее об этих специальных переменных я расскажу в следующих главах.

Упражнение

Для выполнения следующего упражнения открой веб-сайт **ludolf.github.io** на своём компьютере или смартфоне. Запусти программу, щёлкнув по кнопке с галочкой. После запуска программы в нижней области на экране отобразится результат.

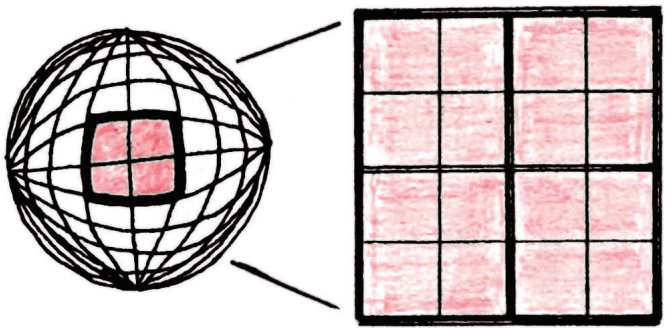
1. Создай новую переменную `x` со значением `1` и выведи её на экран.
2. Создай новую переменную `y` со значением, равным значению переменной `x`, и выведи её на экран.
3. Создай новую переменную `z` со значением, равным сумме значений `x` и `y`, и выведи её на экран.
4. Увеличь на два значение переменной `z` и выведи её на экран.
5. Увеличь в два раза значение переменной `z` (умножение обозначается символом `*`) и выведи её на экран.
6. Прибавь к значению переменной `z` число два, раздели сумму на два и выведи результат на экран (деление записывается символом `/`).
7. Создай новую текстовую переменную `str` с произвольным значением и выведи на экран это значение.

Шаг за шагом – *массивы*

Комета очень большая, и на ней легко заблудиться. Многие проходы между кристаллическими горами заканчиваются тупиками, а другие возвращаются к началу. Каждый свой шаг Лудолф документирует в своей базе данных и тем самым создаёт карту местности, которая помогает ему двигаться вперёд.

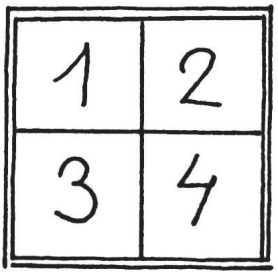
Карты могут существовать, только если в пространстве есть точка начала отсчёта. На Земле для этого используется система географических координат. Она позволяет точно определять местоположение любого объекта на планете, и её можно использовать даже на других планетах. Модель планеты – её глобус – можно представить в виде листа бумаги с нарисованной сеткой линий, наклеенного на мяч. Координаты подскажут, в какой клетке на этом листе находится нужное озеро, скала или пещера.

На карте Лудолфа изображена только малая часть кометы, где робот ждёт твоей помощи. Поскольку комета большая, а всё на ней такое маленькое, то робот может игнорировать круглую форму кометы.

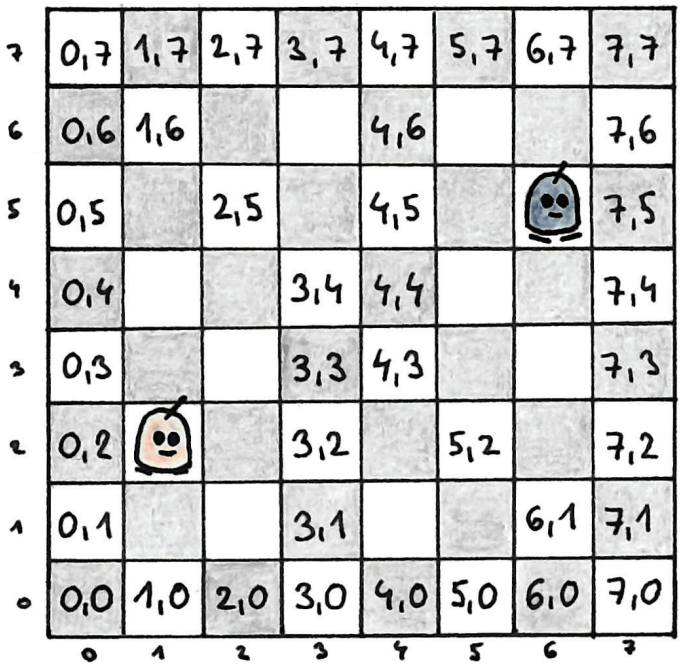


Что такое массивы

Массивы относятся к составному типу данных, который организует значения в упорядоченные элементы. Массив может содержать практически бесконечное количество значений в одной переменной.



Для чего нужны массивы? Они хорошо подходят для работы с большим количеством элементов, например с пикселями на экране или клетками на шахматной доске или карте. На следующей шахматной доске в некоторые клетки вписаны их координаты в виде чисел, разделённых запятой:



Работа с шахматной доской без массивов потребовала бы создания отдельной переменной для каждой клетки со значением `0`, если клетка не занята,

со значением 1, если клетка занята белой фигурой, и со значением 2, если клетка занята чёрной фигурой.

```
chess00 := 0
chess01 := 0
chess02 := 0
...
chess12 := 1
...
chess65 := 2
...
chess77 := 0
```

Это было бы очень неудобно, но, к счастью, есть массивы.

Как записываются массивы

В языке LudolfC массив записывается как список значений, перечисленных через запятую, заключённый в квадратные скобки [и].

Пустой массив:

```
array1 := []
```

Массив с тремя значениями одного и того же типа данных:

```
array2 := [1, 2, 3]
```

Массив с тремя значениями разных типов данных:

```
array3 := [1+2, "Привет", x]
```

Последний массив в примере содержит результат сложения двух чисел в первом элементе, текст – во втором и значение переменной x – в третьем.



Задача № 26

Какие значения будет содержать массив в переменной a в конце следующей программы?

```
x := 1
a := [x, x + 1, x + x]
x := x + 1
```

Многомерные массивы

Массив всегда одномерный, но есть возможность вкладывать массивы друг в друга и таким способом создавать многомерные массивы, например представляющие двумерную шахматную доску или трёхмерный кубик Рубика.

Одномерный массив:

```
a1 := [1,2,3]
```

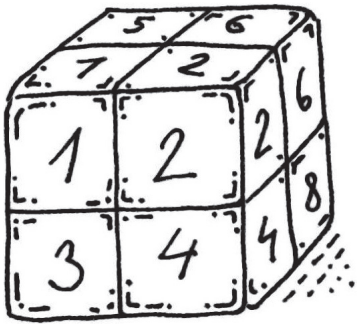
Двумерный массив:

```
a2 := [[1,2,3],
       [4,5,6],
       [7,8,9]]
```



Задача № 27

Создай трёхмерный массив, представляющий кубик с размерами 2×2×2.



Как работать с массивами

Указывая координаты элемента в квадратных скобках, можно получить его значение из уже созданного массива. Эти координаты называются **индексом массива**.

```
array := [1,2,3]
index := 2
value := array[index]
```



Нумерация индексов начинается с нуля:

```
a := [11, 256, 99899]

value_0 := a[0]        // 11
value_1 := a[1]        // 256
value_2 := a[2]        // 99899
```

При работе с многомерными массивами индексы записываются отдельно для каждого вложенного массива и разделяются запятой:

```
a2 := [ [1,2], [3,4] ]

value_00 := a2[0,0]     // 1
value_01 := a2[0,1]     // 2
value_10 := a2[1,0]     // 3
value_11 := a2[1,1]     // 4
```



Задача № 28

Какое значение имеет элемент куба из предыдущей задачи с координатами `1,0,1`?

Массивы, как и текст, можно объединять и сравнивать:

Код	Пример	Результат
`+`	`[1,2] + [3]`	`[1,2,3]`
`=`	`[1,2] = [1,2]`	`True`



Задача № 29

Какие значения будет содержать массив `array` в конце следующей программы?

```
array := [1,2]
array := array + [3] + array
```



Индекс массива никогда не должен превышать или равняться размеру массива, иначе в программе возникнет ошибка:

```
array := [1,2]

okay := array[0]
okay := array[1]
error := array[2]     // ERROR!
```

Почему в последнем выражении возникла ошибка? Массив в примере содержит всего два элемента, но так как отсчёт индексов в массивах начинается с нуля, действительны только два индекса: `0` и `1`.



Для любознательных

В общем случае индексы любого массива находятся в диапазоне от нуля до размера массива минус один: от `0` до размера –1. Эта информация пригодится при автоматическом обходе массива.

Размер массива

Каждый массив имеет специальное свойство `size`, которое содержит число, определяющее его текущий размер, то есть количество элементов, которые содержатся в массиве в данный момент.

Подробнее о *свойствах* я расскажу в главе, посвящённой объектам; пока просто запомни, что доступ к свойствам осуществляется с помощью простого символа *точки*:

```
size := array.size
```

Размер всегда определяется только для одного измерения. При работе с многомерными массивами размер каждого вложенного массива необходимо определять отдельно:

```
array := [[1,2,3],
          [3,4,5]]

size := array.size
size0 := array[0].size
size1 := array[1].size
```



Задача № 30

Какие значения будут иметь переменные `size`, `size0` и `size1` в конце предыдущей программы?

Массивы в памяти компьютера

Из главы о переменных ты уже знаешь, что переменные составных типов данных содержат только указатели на свои значения. Массив можно представить как этикетку на коробке, полной разных вещей. Присваивание переменной-массива другой переменной можно сравнить с добавлением новой этикетки на ту же коробку. Неважно, какая этикетка используется для поиска коробки. При изменении содержимого коробки эти изменения будут видны под любыми этикетками на коробке.

Поскольку в примере ниже `array2` – это просто новая этикетка на коробке с этикеткой `array1`, в конце следующей программы обе переменные будут ссылаться на массив, содержащий элементы `5` и `2`:

```
array1 := [1,2]
array2 := array1

array2[0] := 5

array1 = [1,5]
array2 = [1,5]
```



Задача № 31

Какое значение получит переменная `x` в конце следующей программы?

```
arr1 := [1,2]
arr2 := arr1
arr3 := arr2

arr1[0] := 4
arr3[0] := 5
arr2[0] := arr1[0] + 1

x := arr1[0] + arr3[0] + arr1[1]
```

Карты местности, отправленные с далёкой планеты Лудолфом, на самом деле представляют собой игровое поле в стиле шахматной доски. На нём нет ни пешек, ни коней, ни других шахматных фигур, но есть Лудолф, его ракета и другие предметы. Эта карта доступна в переменной `Map` в виде двумерного массива.



Задача № 32

Используй панель управления для исследования переменной `Map`.

Упражнение

Для выполнения следующего упражнения открой веб-сайт **ludolf.github.io** на своём компьютере или смартфоне. Запусти программу, щёлкнув по кнопке с галочкой. После запуска программы в нижней области на экране отобразится результат.

1. Создай и отобрази пустой массив.
2. Создай и отобрази массив с одним числом, одной текстовой строкой и одним значением истинности.
3. Создай массив с тремя разными числовыми значениями, сохрани этот массив в переменной и отобрази значение элемента с индексом `0`.
4. Создай двумерный массив с двумя вложенными массивами по одному элементу в каждом, сохрани этот массив в переменной и выведи на экран значение элемента второго массива.
5. Создай массив с несколькими разными значениями, сохрани этот массив в переменной и отобрази его размер с помощью специального свойства `size`.

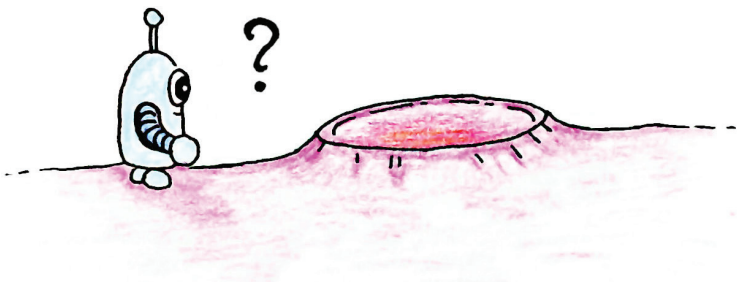
Так или иначе – *условные инструкции*

Комета Когоутек II, как и все тела в космосе, часто подвергается бомбардировке метеороидами, ледяным мусором и частицами звёздной пыли. Землю от разрушительных ударов защищает плотный слой атмосферы, при преодолении которой большинство падающих объектов испаряются или долетают до поверхности лишь в виде небольших метеоритов. Комета Когоутек II не имеет такой защиты. Её поверхность усеяна кратерами и воронками, очень опасными, потому что со временем покрываются зыбучими песками.

Лудолф должен действовать осторожно и тщательно документировать все кратеры на карте. Это непростая задача, потому что ежедневно появляются новые кратеры от падения метеороидов. Падающие камни представляют серьёзную угрозу для Лудолфа и его ракеты. Однако жажда знаний перевешивает риск.

Интеллект – это способность адаптироваться к изменениям. Поэтому он важен в мире, где не только погода, но и всё остальное быстро меняется. Новая программа Лудолфа должна быть достаточно интеллектуальной, чтобы робот мог правильно реагировать на изменения окружающей среды.

Чтобы Лудольф знал, куда идти, нужно ввести в программу так называемые условные команды. Условные команды помогают определить, как действовать в конкретной ситуации.



Что такое условные инструкции?

Условная команда выполняется, только если соблюдается определённое *условие*. *Тело* условной команды – это набор команд. Например, если Лудолф натыкается на камень, он должен обойти его.

Если условие истинно, то его тело (набор команд) выполняется. Если условие ложно, то выполнение программы продолжается с первой инструкцией, следующей за телом. Например, если на пути нет препятствия, то роботу не нужно его объезжать, и он может продолжить движение прямо.

Как записываются условные инструкции

В языке LudolfC условные инструкции записываются как ключевое слово `if`, за которым следует условное выражение, возвращающее значение логического типа, и набор операторов, заключённый в фигурные скобки { и }, представляющий тело условной инструкции. Тело выполняется только в том случае, если условие, то есть его значение, равно `True`.

```
if есть кратер {
  обойти его
  вернуться на дорогу
}
продолжить движение прямо
```

Есть кратер или нет, можно определить по отправленной карте, которая доступна в переменной `Map`. `Map` – это массив, и ты уже знаешь, как работать с массивами:

```
position := Map[x,y]

if position = Rock {
  обойти его
  вернуться на дорогу
}
продолжить движение прямо
```

Тело условной инструкции представляет собой отдельную программу. В него можно добавить любую команду, включая другие условные инструкции. Благодаря условным инструкциям программа станет более интеллектуальной.



Задача № 33

Напиши программу с одним условием, которое содержит другое условие.

А если условие не соблюдается?

Иногда полезно произвести какое-то особое действие, когда условие не соблюдается. Например, если нет кратера, то Лудолф должен собрать образцы почвы, прежде чем продолжить путь.

В языке LudolfC есть возможность использовать расширение условной инструкции – команду **else**. Оно содержит тело, заключённое в фигурные скобки { и }. Тело выполняется только в том случае, если условное выражение вернёт `False`.

```
if есть кратер {
  обойти его
  вернуться на дорогу
}
else {
  собрать образцы почвы
}
продолжить движение прямо
```



Поскольку инструкция `else` является всего лишь расширением условной инструкции, её нельзя использовать без части `if`.



Задача № 34

Какое значение получит переменная `x` в конце следующей программы?

```
x := 1
if x > 1 {
  x := 2
}
else {
  x := 3
}
```

Проверка второго условия в инструкции else

Условные инструкции и инструкции расширения можно комбинировать и создавать составные условные команды, охватывающие различные ситуации. Это полезно, когда в программе необходимо предусмотреть несколько вариантов действий и выбрать только один.

```
if есть кратер {
  обойти его
  вернуться на дорогу
}
else if есть гриб {
  собрать грибы
  вернуться на дорогу
}
else {
  собрать образцы почвы
}
продолжить движение прямо
```



Задача № 35

Какое значение получит переменная x в конце следующей программы?

```
x := 1

if x > 1 {
  x := 2
}
else if x <= 1 {
  x := 3
}
else {
  x := 4
}
```

Выполнение такой составной инструкции прекращается на первом условии, которое будет соблюдено, и не продолжается, даже если далее последуют ещё сто условий, которые тоже соблюдаются:

```
if False {
  не выполнится
}
```

```
else if True {
  выполнится!
}
else if True {
  не выполнится
}
else if True {
  не выполнится
}
else {
  не выполнится
}
```



Задача № 36

Какое значение получит переменная x в конце следующей программы?

```
x := 0
if x > 0 {
  x := 1
}
else if x >= 0 {
  x := 2
}
else if x >= 0 {
  x := 3
}
```

Упражнение

Для выполнения следующего упражнения открой веб-сайт **ludolf.github.io** на своём компьютере или смартфоне. Запусти программу, щёлкнув по кнопке с галочкой. После запуска программы в нижней области на экране отобразится результат.

1. Напиши условную инструкцию, тело которой никогда не будет выполнено.
2. Напиши условную инструкцию, тело которой всегда будет выполняться.
3. Присвой любое число переменной x. Напиши программу, которая присваивает значение 1 переменной x, если предыдущее её значение меньше единицы, или 2, если предыдущее значение больше или равно единице.
4. Присвой переменным x и y произвольные числа. Напиши программу, которая увеличивает значения обеих переменных на единицу, если x меньше, чем y, и на два в ином случае.

5. Присвой переменным x и y произвольные числа. Напиши программу, которая увеличивает значения обеих переменных на единицу, если x меньше y и y больше единицы, или на два в ином случае.

Условные инструкции добавляют в программы способность адаптироваться к ситуации. Благодаря этой способности Лудолф сделал большой шаг к возвращению домой.

Снова и снова – *циклы*

Очередной космический шторм обрушился на комету. Облако звёздной пыли, полное песчинок с острыми гранями, попалось на пути кометы. Скалы вокруг ракеты Лудолфа дрожали и оглушительно гудели. Молнии змеились по чёрному небу, и вся комета, кажется, разлеталась на тысячи осколков.

Когда робот видит надвигающуюся опасность, он бросает всё и спешит к своему космическому кораблю, пока двигатель ещё работает. Снаружи раздаётся грохот и треск. Ракета содрогается от мощного удара. Всё погружается во тьму, загорается аварийное освещение. Включается сирена. Исследовательский комплекс остался без питания. Ситуация критическая. Что-то явно не так.

Людям скучно выполнять рутинные задачи, например аккуратно складывать одежду, а роботы прекрасно справляются с ними. Это объясняется тем, что повторения одних и тех же действий для роботов на самом деле не являются повторениями. Циклы имеются во всех языках программирования. Одна команда, например, может задать миллион повторений некоторых действий без необходимости записывать их миллион раз.

Что такое цикл?

Инструкция **цикла** выполняет заданные команды только при соблюдении *условия*. Тело цикла – это набор команд. Если заданное условие не соблюдено, то программа просто перепрыгивает через тело цикла, не выполняя его.

В этом она напоминает условную инструкцию. Разница лишь в том, что оператор цикла не прекращает выполнение своего тела после первого прохождения, а проверяет условие ещё раз и выполняет тело снова, если условие соблюдается, и так может продолжаться вечно при определённых условиях.

Как записываются циклы

В языке LudolfC команда цикла записывается как **`while`**. Инструкция цикла содержит условное выражение и тело с набором команд, заключённым в фигурные скобки { и }. Эти команды будут выполняться снова и снова, пока условие соблюдается, то есть его возвращает значение `True`.

```
while не хватает энергии {
  искать ресурсы
}
запустить ракету
```

Чтобы ракета могла взлететь, уровень энергии должен быть не менее 100 пунктов. Если Лудолф будет набирать десять пунктов энергии на каждом шаге, то его программа будет выглядеть так:

```
energy := 0

while energy < 100 {
  energy := energy + 10
}
```



Задача № 37

Какое значение получит переменная `x` в конце следующей программы?

```
x := 3

while x > 1 {
  x := x - 1
}
```



Задача № 38

Какое значение получит переменная `y` в конце следующей программы?

```
x := 0
y := 0

while x < 10 {
  x := x + 2
  y := y + 1
}
```

Вложенные циклы

Циклы, как и условные инструкции, можно вкладывать друг в друга. Это может пригодиться при работе с многомерными массивами, такими как карта Лудолфа. Для обхода двумерного массива требуется два цикла, трёхмерного – три и т. д.

Каждый цикл выполняет обход вдоль одного измерения массива. Координаты `x` и `y` задают индексы текущего столбца и строки. Напомню, что нумерация индексов массивов всегда начинается с нуля:

```
size := Map.size

x := 0
y := 0

while x < size {
  while y < size {
    position := Map[x,y]
    y := y + 1
  }
  x := x + 1
}
```



Задача № 39

Напиши программу, которая обходит массив не по строкам, а по столбцам.



Задача № 40

Какое значение получит переменная `result` в конце следующей программы?

```
arr := [[0,1,2], [3,4,5], [6,7,8]]
size := arr.size
result := 0
i := 0

while i < size {
  result := result + arr[i,i]
  i := i + 1
}
```

Циклы и условные инструкции являются самыми полезными командами в компьютерных программах. Без циклов машины были бы не такими умными, какими они являются сегодня.



Не забывай изменять условие в теле цикла. В противном случае цикл будет выполняться вечно, и программа никогда не завершится:

```
i := 0
while i < 1 {
  // будет повторяться вечно
}
```

Упражнение

Для выполнения следующего упражнения открой веб-сайт **ludolf.github.io** на своём компьютере или смартфоне. Запусти программу, щёлкнув по кнопке с галочкой. После запуска программы в нижней области на экране отобразится результат.

1. Присвой любое число переменной `x`. Напиши программу, которая увеличивает значение этой переменной двести раз, используя инструкцию `while`.
2. Создай массив любого размера, содержащий только числовые значения. Напиши программу, которая перебирает все элементы массива и сохраняет их сумму в переменной `x`.
3. Создай двумерный массив любого размера, содержащий только числовые значения. Напиши программу, которая перебирает все элементы массива и сохраняет их сумму в переменной `x`.
4. Присвой переменной `x` любое число. Присвой переменной `y` противоположное значение (`-x`). Напиши программу, которая многократно уменьшает значение `x` на единицу и увеличивает значение `y` на единицу, пока `x` не станет равно `y`. Какие значения будут иметь эти переменные в конце программы?

Выполнение по требованию – *функции*

Когда шторм стих, робот приступил к определению последствий. Как оказалось, один из метеоритов, словно пуля, пробил щит ракеты, повредил память компьютера и проделал дырку в топливном баке, что привело к утечке топлива.

Робот перезагрузил систему и продолжил оценку ущерба. Результаты были весьма неутешительны: ракета потеряла значительную часть энергии, а навигационные системы вообще были уничтожены полностью.

К счастью, сам робот не был повреждён. Лудолф сохранил работоспособность, потому что у роботов есть собственный источник питания. Но он не может вернуться домой без энергии для ракеты, а получить её без управляющей программы он не в состоянии. И даже в этой непростой ситуации робот сохраняет спокойствие. Ремонт сломанного щита и заделывание отверстий – наименьшая из его забот. Главная проблема – как обеспечить себя достаточным количеством энергии для возвращения домой?

Однако всякая проблема имеет решение. В конце концов, на комете полно радиоактивных грибов, пропитанных энергией. Если бы Лудолф мог начать сбор немедленно, то, возможно, он смог бы стартовать и отправиться в путь домой. Но как это сделать? Робот сбит с толку и беспомощен. Программные функции управления движением были повреждены во время шторма.

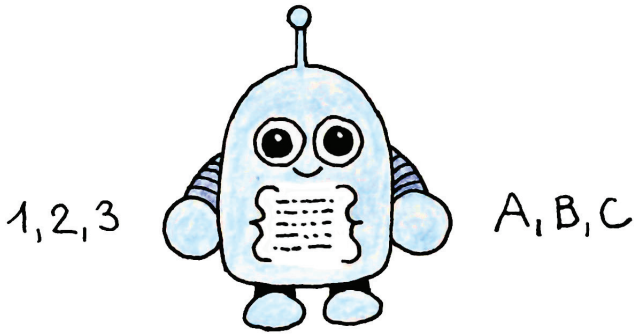
Всё во Вселенной существует не просто так. Всякая вещь служит полезной цели и улучшает жизнь её владельца. Что хорошего в руке, если её нельзя поднять или почесать за ухом?

Что такое функции

Функции – это процедуры, которые можно использовать, чтобы разбить большую программу на более мелкие части.

Функцию можно вызывать с различными входными значениями, которые называют *параметрами*. Например, сложение – это базовая математическая функция, которая складывает два числа. Эти два числа являются параметрами функции.

Помнишь главу о вводе и выводе? Функции похожи на миниатюрные компьютеры: они тоже принимают входные данные и выводят результаты своих вычислений.



Ещё один пример – функция, отображающая значение на экране. Эта функция принимает значение в параметре и отображает его.

Роботы имеют много полезных функций, таких как движение вправо, картографирование местности или написание домашнего задания по математике.

Без функций программы быстро стали бы нечитаемыми. Программы можно разбить на множество небольших функций, взаимодействующих друг с другом. Компьютер может не заметить большой разницы между длинной программой и несколькими короткими функциями, но программист-человек, безусловно, замечает.



Для любознательных

Разложение сложной проблемы на более мелкие и простые задачи называется *декомпозицией* и является одним из важнейших приёмов, используемых при создании компьютерных программ.

Как записываются функции

Функция на языке LudolfC определяется *списком параметров* в круглых скобках (и), за которым следует набор инструкций в фигурных скобках { и }, составляющих тело функции. Результат функции определяется последним выражением в её теле:

```
add := (x, y) {
  x + y
}
```

Тело функции, в принципе, может быть пустым, но такая функция не несёт никакой пользы, поскольку она ничего не делает и только занимает память:

```
f := (){}
```

Обычно функции содержат несколько инструкций. Фактически их тело выглядит как небольшая программа и может включать в себя другие вложенные функции:

```
f := () {
  command 1
  command 2
  command 3
  g := (){ ... }
  ...
  result
}
```

Как вызывать функции

После создания функцию можно вызвать в любой момент, использовав *её имя и список значений*, соответствующих её входным параметрам, в круглых скобках:

```
result := add(2, 3)
```



Задача № 41

Напиши и вызови функцию для сложения трёх чисел.



Задача № 42

Выясни, как работает предыдущая функция сложения трёх чисел и как с её помощью сложить два числа.

Функции, которые ничего не возвращают

Некоторые функции просто выполняют требуемое действие (например, выводят заданное значение на экране) и не возвращают никаких результатов. Такие функции называются *процедурами* и могут вызываться без присваивания результата переменной:

```
Display("Привет, Мир!")
```



Для любознательных

Функции всегда возвращают какой-то результат (например, сумму двух чисел) или выполняют какое-то действие (например, изменяют пиксели на экране). В противном случае они были бы бесполезны, как и пустая функция выше.

Функции без параметров

Функции не обязательно должны иметь параметры. Функции без параметров всегда выполняют одно и то же действие (например, поднимают руку робота) или всегда возвращают один и тот же результат (например, текущие дату и время):

```
raise_hand()

today := Date()
```

Ещё одним примером функций без параметров являются функции, которые всегда возвращают одно и то же значение (константу):

```
two := (){ 2 }
```

Использование функций

Функции упрощают создание программ. Вместо перечисления всех команд в виде длинной последовательности можно разделить программу на несколько частей-функций и использовать эти функции. Такие программы проще читать и легче изучать, поскольку функциям обычно принято давать описательные имена:

```
energy := 0

while is_low_energy() {
  collect_energy()
}
```



Задача № 43

Напиши функции `is_low_energy` и `collect_energy`, используемые в примере выше.

Область видимости функции

Функции часто имеют параметры, имена которых совпадают с именами некоторых переменных. Параметры функций – это особые переменные, которые существуют *только внутри* своей функции. Это делает функции независимыми от остальной части программы и превращает их в по-настоящему независимые подпрограммы. **Область видимости** параметра ограничена его функцией:

```
func := (x) {
  x := 2
}

x := 1
func(x)
y := x
```

Какое значение получит переменная `y` в конце этой программы? Поскольку параметр функции – это особая переменная, область видимости которой ограничена только телом функции, исходная переменная `x` остаётся нетронутой. Поэтому значение переменной `y` в конце программы по-прежнему будет равно `1`.



Задача № 44

Какое значение получит переменная z в конце следующей программы?

```
func := (x) {
  x := x + 3
  x + 1
}

x := 1
y := func(x)
z := x + y
```

Функции, используемые для сбора энергии, которые было предложено написать в задаче № 43, можно определить, как показано ниже, используя особенности области видимости переменных:

```
is_low_energy := (energy) {
  energy< 10
}

collect_energy := (energy) {
    energy + 1
}

energie := 0

while is_low_energy(energy) {
  energy := collect_energy(energy)
}
```

Не только параметры, но и все переменные, созданные внутри функции, ограничены областью видимости функции. Всё, что происходит в теле функции, в нём и остаётся:

```
f := () {
  x := 1
}
f()
y := x + 2 // ERROR!
           // здесь переменная x не существует
```

Ограничения области видимости функций распространяются на все команды, тела которых заключаются в фигурные скобки { и }.



Для любознательных

В большинстве языков программирования область видимости переменных определяется с помощью стека. *Стек* – это составная структура данных, которую можно представить как стопку листов бумаг на рабочем столе. Каждый лист содержит краткое описание всех переменных программы и их текущие значения.

При вызове функции на вершину стопки добавляется новый чистый лист, и на нём записываются все переменные, объявленные в теле функции, и все её параметры. Программа всегда сначала ищет нужную переменную на верхнем листе, затем на листе под ним и т. д.

Когда функция завершается, соответствующий лист бумаги удаляется из стопки, и работа продолжается с использованием предыдущего листа, оказавшегося на вершине стопки.

Один лист бумаги представляет область видимости переменных, написанных на нём.

Замыкания

Функции не обязательно должны быть полностью изолированы от остального кода. Они могут также использовать переменные из ближайшего окружения. Такие функции называют **замыканиями**, потому что они *замыкают* ранее свободные переменные в функцию.

```
x := 0
increase := () {
  y := x + 1
}
```

Переменная, замкнутая внутри функции, будет иметь значение, которое в ней было в момент вызова функции:

```
x := 0
increase := () {
  x + 1
}

increase() = 1

x := 5
increase() = 6
```



Используй замыкания с большой осторожностью, потому что легко забыть, какие переменные использует функция и что она с ними делает. Программы с замыканиями быстро становятся запутанными и полны неожиданных ошибок.



Задача № 45

Попробуй избавиться от замыкания в функции `increase`, добавив параметр.

Функция отображения любого значения на экране панели управления доступна через переменную `Display`. Эта функция принимает один или несколько параметров, которые требуется отобразить:

```
Display(123)
Display("Привет, Мир!")
Display(x + y)
Display(Map)
Display("Начало: ", Map[0,0])
```



Задача № 46

Поэкспериментируй с функцией `Display` в панели управления.



Для любознательных

Функции – уникальная особенность языков программирования. Фактически с их помощью можно написать любую компьютерную программу.

Функции являются ещё одним фундаментальным элементом компьютерного программирования. Чтобы написать любую программу, программисту достаточно переменных с циклами или функций. Стиль программирования, использующий только функции, называется *функциональным программированием*.

Современные языки программирования, такие как LudolfC, обычно дают возможность использовать и функции, и переменные с циклами, позволяя программисту выбирать предпочтительный стиль или выгодно комбинировать оба стиля.

Упражнение

Для выполнения следующего упражнения открой веб-сайт **ludolf.github.io** на своём компьютере или смартфоне. Запусти программу, щёлкнув по кнопке с галочкой. После запуска программы в нижней области на экране отобразится результат.

1. Напиши функции, выполняющие основные математические операции, такие как сложение, вычитание, умножение и деление.
2. Напиши функцию, которая умножает свой единственный параметр на самого себя. Какой результат вернёт эта функция для параметра `5`? Как называется эта функция в математике?
3. Напиши функцию с тремя параметрами, которая возвращает второй параметр, если первый имеет значение `True`, и третий параметр в противном случае.
4. Напиши функцию с одним числовым параметром, которая отображает его значение с помощью функции `Display`, а затем вызывает себя с параметром, уменьшенным на единицу, если его значение больше нуля.

Я робот – *объекты*

SOS! Соединение прервано. Невозможно установить связь с Землёй. Соединение прервано. Попробуйте ещё раз. SOS! Соединение прервано.

Без программы управления невозможно установить связь с Землёй.

Даже в этой ситуации Лудолф не сдаётся. Не только потому, что ему в своё время не объяснили, что в трудную минуту можно опустить руки и впасть в уныние, но и потому, что он прекрасно понимает, что любые неудачи – временные. Да, основной канал связи больше не работает, но можно организовать новый.

Маленькая жестяная коробочка послужит спасательным модулем.

Сними дополнительную панель управления. Панель управления – это простой компьютер с экраном, клавиатурой и сигнальной лампочкой. Руководство по управлению прикреплено к коробочке. Он начинает писать буквы от руки на листе бумаги.

Запусти спасательный модуль. Хлопок – и важный груз направляется на Землю. Надеемся, он попадёт в добрые и умелые руки!

Компьютерная программа никогда не будет пытаться смоделировать весь мир с точки зрения деталей и многообразия. Напротив, в своих программах программист сосредоточивается только на самом главном, опустив всё остальное. Например, программе навигации по суровой комете достаточно знать текущие координаты робота. Такие детали, как количество транзисторов в процессоре, разрешение камеры или материал корпуса, можно полностью опустить:

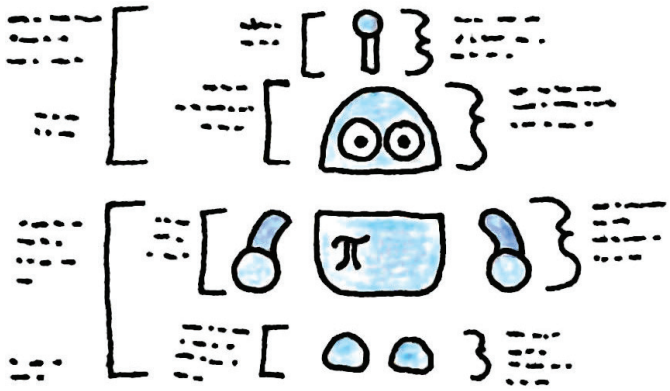
```
Ludolf := [x, y]
```

Для моделирования мира в языках программирования используются объекты.

Что такое объекты

Объекты – это составные типы данных, которые объединяют данные и поведение в одной переменной.

Объекты – это структуры, похожие на массивы, но обладающие более широкими возможностями. В отличие от элементов массива, к которым можно получить доступ только с помощью числовых индексов, свойства объектов имеют собственные имена. Робот тоже является объектом, поскольку состоит из различных компонентов, таких как антенна, компьютерная голова и механические руки:





Для любознательных

Объекты были добавлены в современные языки программирования, чтобы упростить моделирование реального мира, но они имеются не во всех языках.

Объектно ориентированное программирование (ООП) – это, пожалуй, самый популярный стиль программирования в последние десятилетия, использующий объекты.

Объектно ориентированные языки часто поддерживают идею о том, что *всё является объектом*, даже числа и текстовые строки. Эту идею мы обсудим далее в этой главе.

Как записываются объекты

В языке LudolfC объект создаётся заключением списка свойств в фигурные скобки { и }. После *имени* объекта следует двоеточие, а свойства отделяются друг от друга запятой:

```
object := {
  name1: value1,
  name2: value2,
  name3: value3,
  ...
  nameX: valueX
}
```

Значение свойства может быть любого типа, в том числе другими объектами и функциями:

```
mixed_object := {
  a: 1,
  b: "B",
  c: {},
  d: (){}
}
```

Также можно создать объект, не имеющий никаких свойств. Но такой пустой объект практически бесполезен:

```
empty_object := {}
```

Свойства объекта используются для моделирования *атрибутов*, таких как названия и серийные номера, а также *статуса* объекта, например местоположения на карте. Статус объекта может меняться, например по мере перемещения робота по карте. Поведение объектов – действий, которые объект способен выполнять, – моделируется с помощью функций.

```
robot := {
  nickname: "Лудолф",
  sernum: 256,
  position: { x: 0, y: 0 },

  add: (a, b) {
    a + b
  },

  sub: (a, b) {
    a - b
  }
}
```

Как работать с объектами

Получить значение требуемого свойства объекта можно, указав имя переменной, в которой хранится объект, и имя свойств *через точку*:

```
serial := robot.sernum
five := robot.add(2, 3)
```

Доступ к свойствам вложенных объектов может осуществляться по цепочке. Например, вот как можно найти координату x вложенного объекта:

```
coordinate_x := robot.position.x
coordinate_y := robot.position.y
```



Поскольку объекты – это составные типы данных, их присваивание переменным приводит только к копированию указателя на объект, то есть его адреса в памяти.



Задача № 47

Какое значение получит переменная x в конце следующей программы?

```
o := { a:1 }
q := o

qa := 2
x := oa + qa
```

Использование объектов

Объекты можно использовать для создания хорошо организованных и простых для понимания программ. Представьте, что объекты – это небольшие программы, содержащие всё необходимое для решения узкого круга задач. Все важные свойства объединены в объекты. Задачи, решаемые различными объектами, чётко отделены друг от друга.

Объекты не должны быть слишком маленькими или слишком большими. Каждый объект должен решать чёткую задачу, достаточно простую для понимания. Когда задача становится слишком длинной или сложной, то с помощью объектов её следует разделить на более мелкие задачи. Из этих более мелких объектов можно сформировать свойства более крупных объектов:

```
coordinates := {
  x: 0,
  y: 0
}

energy := {
  capacity: 100,
  current: 23
}

robot := {
  location: coordinates,
  engine: energy
}
```

В предыдущем примере программы объект `robot` состоит из объектов `coordinates` и `energy`.

Внутри объекта

После выполнения действия некоторые свойства, определяющие состояние объекта, могут изменяться. Например, после выполнения действия перемещения изменяются координаты. Поскольку действие перемещения и координаты являются частями объекта, важно иметь возможность доступа к ним *изнутри самого объекта*.

В LudolfC свойства объекта тоже являются переменными. Однако эти переменные существуют только внутри объекта:

```
Ludolf := {
  position: { x: 0, y: 0 },
  move: (x, y) {
    position.x := x
    position.y := y
  }
}
```



Задача № 48

Какие координаты будут иметь роботы Лудолф и Эйлер в конце следующей программы?

```
ludolf := {
  position: { x: 0, y: 0 },
  move: (x, y) {
    position.x := x
    position.y := y
  }
}

euler := {
  position: { x: 0, y: 0 },
  move: (x, y) {
    position.x := x
    position.y := y
  }
}

ludolf.move(1,2)
euler.move(5,3)
```

Подобно функциям, переменные со свойствами объекта имеют область видимости внутри объекта и поэтому делают недоступными одноимённые переменные вовне:

```
position := { x: 0, y: 0 }
position.x := 2
x := 1
y := 3
position.y := 4
```



Задача № 49

Какие значения будут иметь координаты `x` и `y` объекта `position` в конце предыдущей программы?



Для любознательных

В некоторых языках программирования имеется специальное имя, используемое для доступа к свойствам объекта. Это имя представляет текущий объект внутри функций, определяющих поведение объекта, и определено только внутри этих функций. С помощью этого специального имени можно работать с переменными вне объекта, имеющими те же имена, что и некоторые свойства объекта, которые в противном случае были бы недоступны внутри объекта.

В JavaScript это имя `this`, в Python – `self`, а в LudolfC – знак доллара `$`.

Всё является объектом

Свойства объектов актуальны не только для роботов. Каждый элемент во Вселенной обладает определёнными свойствами и поведением (например, атомным числом и температурой кипения). То же относится и к элементам программы, таким как числа. Каждое число имеет своё значение и набор действий, которые можно с ним выполнить. Например, к каждому числу можно прибавить ещё одно число. Это означает, что числа, как и все другие элементы программирования, тоже являются объектами:

```
four := 1.plus(3)
five := four.plus(1)
```

Предыдущая и следующая программы действуют одинаково:

```
four := 1 + 3
five := four + 1
```



Задача № 50

Перепиши следующую программу, используя объекты:

```
x := 1 + 2 + 3
```



Для любознательных

Большинство объектно ориентированных языков программирования выражают все элементы как объекты, но при этом используют типы данных, не являющиеся объектами, для повышения производительности. Объекты имеют довольно сложную структуру, и компьютерам сложнее работать с ними, чем, скажем, с обычными числами. Такие типы данных, не являющиеся объектами, называют *простыми* или *элементарными типами данных*.

Объект, моделирующий робота, и его характеристики доступны через переменную `Ludolf`. Этот объект имеет следующие свойства:

Свойство	Тип данных	Пример использования
`position`	объект `{x, y}`	`Ludolf.position.x`
`up`	функция	`Ludolf.up()`
`down`	функция	`Ludolf.down()`
`right`	функция	`Ludolf.right()`
`left`	функция	`Ludolf.left()`
`avoid`	функция	`Ludolf.avoid()`
`charge`	функция	`Ludolf.charge()`



Задача № 51

Попробуй поработать с переменной `Ludolf` в панели управления.

Фабрики объектов

Часто бывает необходимо создать несколько объектов с одинаковой структурой. Запись полного определения объекта снова и снова не самая приятная задача, особенно если объекты очень большие. Используя фабрики объектов, можно значительно упростить создание таких объектов.

Фабрики объектов – это обычные функции, которые возвращают объект, сконструированный на основе параметров. Чтобы отличать их от обычных функций, имена фабрик обычно начинаются с заглавной буквы:

```
Robot := (n, s, x, y) {{
  name: n,
  sernum: s,
  position: { x: x, y: y },

  move: (x, y) {
    position.x := x
    position.y := y
  }
}}
```

Объекты можно производить точно так же, как на настоящей фабрике производятся роботы на конвейере:

```
ludolf := Robot('Лудольф', 'A001', 0, 0)
euler := Robot('Эйлер', 'A002', 10, 20)
```



Для любознательных

В языках программирования фабрики имеют различные имена и формы. Одни языки предлагают классы с конструкторами, а другие – специальные функции внутри объекта.

Классы – это причудливые фабрики с дополнительными возможностями, такими как наследование свойств дочерним классом от родительского.

Упражнение

Для выполнения следующего упражнения открой веб-сайт **ludolf.github.io** на своём компьютере или смартфоне. Запусти программу, щёлкнув по кнопке с галочкой. После запуска программы в нижней области на экране отобразится результат.

1. Напиши объект, содержащий все нужные тебе свойства, и сохрани его в переменной со своим именем.
2. Напиши объект, содержащий основные арифметические операции и свойство со значением лудольфова числа π (найди его значение в интернете).
3. Напиши объект, содержащий некоторые известные свойства кометы Когоутек II.

Назад домой

Возвращение – *решение*

Компьютеры лишены человеческих эмоций. Они не умеют смеяться или плакать. Однако, когда на панели управления ракеты вспыхивает индикатор, подтверждающий, что она получила ваш сигнал, буря сигналов захлёстывает разум Лудолфа, устанавливая все цепи в гармоническую связь. Робот счастлив. Его призыв о помощи донёсся до Земли. Экспедиция спасена!

Лампочки замигали, а панель управления загудела, выполняя интенсивные вычисления. Сигналы преобразуются в биты, биты – в инструкции, а инструкции используются для загрузки новой программы управления в память. Щелчок. На мгновение всё затихло. Вспышка. Замигал зелёный свет. Работает! Когда открылся люк ракеты, поднявшаяся кристаллическая пыль на какое-то время затмила обзор. Долгое ожидание наконец закончилось. Лудолф выходит на фиолетовую поверхность, вооружённый новой программой. Однако многое изменилось. Шторм преобразил ландшафт, словно он был сделан из кубиков лего. Старая карта Лудолфа стала бесполезной в этих условиях. Новая программа должна быть достаточно интеллектуальной, чтобы справиться с этими изменениями.

Сможешь ли ты создать программу, достаточно интеллектуальную, чтобы преодолеть все препятствия и накопить достаточно энергии для взлёта?

Твоя задача – написать программу, которая управляет перемещением робота на местности, помогая ему обходить препятствия и собирать энергию. Руководство по управлению, которое было в коробке, прилетевшей из космоса, расскажет тебе несколько интересных фактов:

Что	Количество энергии
Один шаг (перемещение в соседний квадрат)	1 (расходование)
Обход кратера	2 (расходование)
Энергетический гриб	10 (накопление)
Начальное состояние	30 (константа)
Для запуска ракеты	100 (не меньше)

Поскольку каждое действие приводит к накоплению или расходованию энергии, программа должна будет направлять робота так, чтобы накопить хотя бы 100 единиц энергии (для чего нужно собрать не менее десяти грибов) за как можно меньшее количество шагов. В противном случае робот может израсходовать всю собранную энергию прежде, чем вернётся в ракету.



Для любознательных

Количество шагов, которые должна выполнить программа для выполнения задачи, является важнейшим свойством любой программы, известным как *эффективность*.

Чем выше эффективность, тем быстрее работает программа и тем меньше ресурсов, таких как энергия, она использует. Программист должен стараться писать максимально эффективные программы, чтобы экономить энергию, запасы которой на Земле ограничены и которую нельзя тратить впустую.

Когда дело доходит до написания сложных программ, лучшее решение редко бывает очевидным сразу. Поэтому программисты начинают с создания несовершенного, но работающего решения, которое затем постепенно улучшают. Часто в процессе его создания само собой всплывает идеальное решение.

Первая попытка

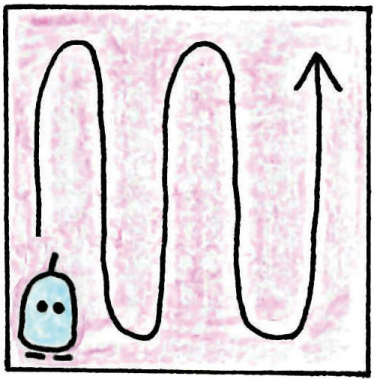
Итак, какое решение является несовершенным, но работающим? Любое, какое придёт в голову первым! Используя две вложенные инструкции цикла, можно передвигаться по всей карте, перемещая робота в нужном направлении:

```
x := 0
y := 0
while x < Map.size {
  while y < Map.size {
```

```
    // переместить робота в правильном направлении

    y := y + 1
  }
  y := 0
  x := x + 1
}
```

А какое направление правильное? Квадрат с координатами (0,0) находится в нижнем левом углу. При увеличении значения горизонтальной координаты (переменной x) перемещение происходит на один столбец вправо, а при увеличении значения вертикальной координаты (переменной y) перемещение происходит на одну клетку вверх в текущем столбце. Когда робот достигает конца цикла, он перемещается вправо и меняет направление на «сверху вниз» или «снизу вверх».



Поскольку направление имеет только два значения (вверх и вниз), для его обозначения можно использовать тип данных истинности, который тоже имеет только два значения (`True` и `False`):

```
Up := True
Down := False

direction := Up
```

Значение переменной `direction` определяет направление движения робота с помощью условной команды:

```
if direction = Up {
  Ludolf.up()
}
else {
  Ludolf.down()
}
```

Поскольку направление хранится в программе как значение истинности, для его изменения можно использовать операцию отрицания:

```
direction := !direction
```

Вся программа будет выглядеть так:

```
Up := True
Down := False

direction := Up
x := 0
y := 0

while x < Map.size {
  while y < Map.size {

    if direction = Up {
      Ludolf.up()
    }
    else {
      Ludolf.down()
    }

    y := y + 1
  }

  y := 0
  x := x + 1

  Ludolf.right()
  direction := !direction
}
```



Задача № 52

Запусти программу с помощью панели управления.

Однако эта программа работает некорректно. Одна из причин – кратеры на пути, в которых робот может застрять навсегда. Вторая причина – ошибочность предположения, что робот начинает свой путь в левом нижнем углу карты, в точке с координатами (0,0). Взглянув на присланную роботом карту, можно заметить, что это не так. На самом деле Лудолф стоит у своей ракеты, и его сначала нужно доставить в точку с координатами (0,0).

К решению обеих проблем нужно подходить со всей тщательностью.

Обход кратеров

Лудолф обнаружил несколько кратеров на комете, которые следует обходить стороной. Благодаря руководству по управлению, присланному роботом, тебе известно, что кратеры обозначаются на карте числом `30`. Чтобы программа была понятнее, это число можно сохранить в переменной:

```
Crater := 30
```

Узнать значение поля, на котором в данный момент находится робот, можно так:

```
x := Ludolf.position.x
y := Ludolf.position.y
field := Map[x, y]
```

К счастью, у робота есть специальная функция, помогающая выбираться из кратеров. Она определена как свойство с именем `avoid`. Робот должен использовать эту функцию, если достигнет поля с кратером, иначе он там застрянет:

```
if field = Crater {
  Ludolf.avoid()
}
```



Роботу требуется довольно много энергии для выполнения функции `avoid`, поэтому вызывать её желательно, только когда робот действительно оказался на поле с кратером.

Код, помогающий выбираться из кратера, можно сохранить в новой функции:

```
avoid_crater := () {
  x := Ludolf.position.x
  y := Ludolf.position.y

  field := Map[x, y]

  if field = Crater {
    Ludolf.avoid()
  }
}
```

Программа всегда вызывает эту функцию перед переходом на новое поле:

```
while x < Map.size {
  while y < Map.size {

    avoid_crater()
```

```
    if direction = Up {
      Ludolf.up()
    }
    else {
      Ludolf.down()
    }
    ...
```

Переход в начало

Как переместить робота в нижний левый угол? Самый быстрый способ – перемещать его вниз и влево, пока он не достигнет поля (0,0). Другими словами, пока координата `x` или `y` не станет отличной от нуля. Чтобы объединить эти два условия, можно использовать операцию логического сложения `|`, которая означает «ИЛИ»:

```
while Ludolf.position.x != 0 |
      Ludolf.position.y != 0 {

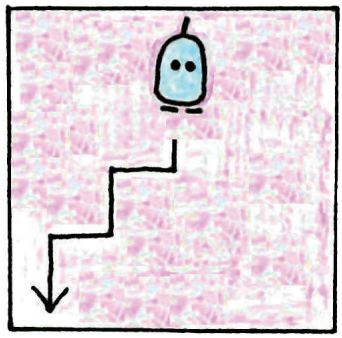
  Ludolf.down()
  Ludolf.left()
}
```

Конечно, нельзя забывать о кратерах:

```
while Ludolf.position.x != 0 |
      Ludolf.position.y != 0 {

  avoid_crater()

  Ludolf.down()
  Ludolf.left()
}
```





Задача № 53

Попробуй запустить эту программу с помощью панели управления.

Поскольку функция перемещения робота в определённое место наверняка пригодится в дальнейшем, было бы неплохо её усовершенствовать, чтобы она позволяла перемещаться в любую точку на карте. Новая функция не будет иметь больших отличий, а просто станет немного умнее: вместо перемещения в точку с координатами (0,0), где направление было очевидно, новая функция должна будет определить направление на основе текущего положения робота относительно целевых координат:

```
move_to := (x, y) {
  while Ludolf.position.x != x |
        Ludolf.position.y != y {

    avoid_crater()

    // переместить робота в правильном направлении
  }
}
```

Итак, какое направление правильное? Робот должен двигаться влево, если стоит справа от цели, и наоборот. Робот должен двигаться вниз, если стоит выше цели, и наоборот.

```
if Ludolf.position.x > x {
  Ludolf.left()
}
else if Ludolf.position.x < x {
  Ludolf.right()
}

if Ludolf.position.y > y {
  Ludolf.down()
}
else if Ludolf.position.y < y {
  Ludolf.up()
}
```



Задача № 54

Проверь работу этой функции, используя панель управления.

Энергия

Главная задача Лудолфа – поиск и сбор энергии из радиоактивных грибов. Благодаря руководству по управлению, присланному роботом, тебе известно, что грибы обозначаются на карте числом `40`. Чтобы программа была понятнее, это число можно сохранить в переменной:

```
Energy := 40
```

Когда робот встречает гриб, он поднимает его и использует функцию зарядки, чтобы получить энергию.

```
x := Ludolf.position.x
y := Ludolf.position.y

field := Map[x, y]

if field = Energy {
  Ludolf.charge()
}
```

Для большей ясности этот код можно сохранить в новой функции и вызывать её после функции, помогающей выбираться из кратера:

```
while x < Map.size {
  while y < Map.size {

    avoid_crater()
    charge_energy()

    if direction = Up {
      Ludolf.up()
    }
    else {
      Ludolf.down()
    }
    ...
```

Полный код можно найти по адресу **ludolf.github.io/en/1.txt**.



Задача № 55

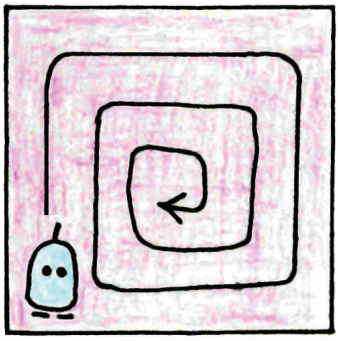
Запусти программу с помощью панели управления.

Лучшее решение

Несмотря на то что, следуя этой программе, робот найдёт все грибы в близлежащих окрестностях, запустив программу, ты обнаружишь, что ему не удалось собрать достаточно энергии (100 очков). Программа неэффективна и тратит много энергии на блуждание по всем полям, где нет грибов.

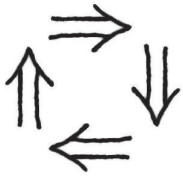
Если внимательно посмотреть на карту, то легко заметить, что грибы растут только на двух *самых внешних* уровнях (в первой и второй вертикали и горизонтали). Эти знания могут оказаться полезными. Если ценные грибы имеются только на краях карты, то зачем тратить время на осмотр всей области?

Вместо того чтобы выполнять обход карты поле за полем, робот должен пойти по спирали, напоминающей ракушку улитки: начать с нижнего левого угла, переместить вдоль первого столбца вверх, затем вправо, вниз и влево. Затем повторить всё то же самое, сместившись на один уровень ближе к центру. После второго повторения можно остановиться без всяких сожалений и вернуться в ракету, потому что грибы растут только на первых двух уровнях:



После достижения конца первого столбца по пути вверх необходимо повернуть вправо и двигаться до конца верхней строки, затем вниз по последнему столбцу и по достижении самой нижней строки повернуть влево и дойти до позиции, находящейся правее начальной позиции, чтобы не тратить энергию понапрасну для возврата на шаг назад перед обходом второго уровня.

Аналогично происходит обход второго уровня, только на этот раз движение вдоль столбца или строки выполняется не до последней, а до предпоследней позиции:



Программа будет выглядеть примерно так:

```
while уровень < 2 {

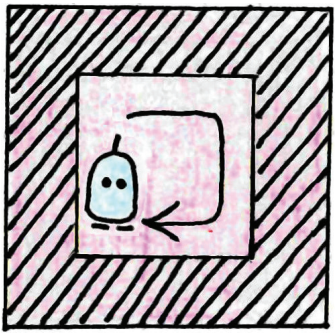
  // двигаться вверх
  // до верхнего левого угла

  // двигаться вправо
  // до верхнего правого угла

  // двигаться вниз
  // до нижнего правого угла

  // двигаться влево
  // до поля перед нижним левым углом
}
```

После первого витка спирали робот переходит на один уровень ближе к центру, и вся программа повторяется на новом уровне погружения.



Для обозначения каждой стороны понадобится одна переменная, хранящая текущее состояние ребра. Движение из нижнего левого угла начинается с начала карты, с нуля, а в верхнем правом углу – с конца карты:

```
bottom := 0
left   := 0
top    := Map.size - 1
right  := Map.size - 1
```

Движение вверх начинается в левом нижнем углу. Робот всегда перемещается на одно поле вверх вдоль оси координат `y`:

```
y := bottom
while y <= top {
  // шагнуть вверх
  y := y + 1
}
```

По достижении верхнего края нужно указать, что самый левый столбец пройден, и в следующем витке спирали робот должен будет двигаться по второму слева столбцу:

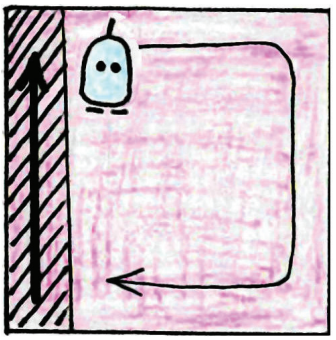
```
left := left + 1
```

Ты уже знаешь, как управлять роботом. Не забывай на каждом шаге пытаться зарядить аккумуляторы энергией:

```
y := bottom
while y <= top {

  Ludolf.up()
  charge_energy()

  y := y + 1
}
left := left + 1
```



Достигнув конца первого столбца, робот должен повернуть и начать движение вправо вдоль оси `x`. Робот начинает движение в левом верхнем углу и на каждом шаге перемещается на одно поле вправо:

```
x := left
while x <= right {

  // шагнуть вправо
  x := x + 1
}
```

По достижении правого края нужно указать, что самая верхняя строка пройдена, и в следующем витке спирали робот должен будет двигаться по второй сверху строке:

```
top := top - 1
```

Оставшаяся часть довольно очевидна:

```
x := left
while x <= right {

  Ludolf.right()
  charge_energy()

  x := x + 1
}
top := top - 1
```



Задача № 56

Напиши остальную часть программы, следуя которой робот пройдёт вниз и затем влево, и проверь её в панели управления.

Полный код можно найти по адресу **ludolf.github.io/en/2.txt**.

К ракете

После получения необходимой энергии не остаётся причин для задержки. Необходимо как можно скорее улететь. Эта часть программы будет похожа на движение к начальной точке, только робот должен добраться до поля с ракетой, а не до начала координат (0,0). К слову сказать, у нас уже есть функция для этого:

```
x := Rocket.position.x
y := Rocket.position.y

move_to(x, y)
```

Ура! Лудолф забирается в ракету и готов к отправке:

```
Rocket.start()
```

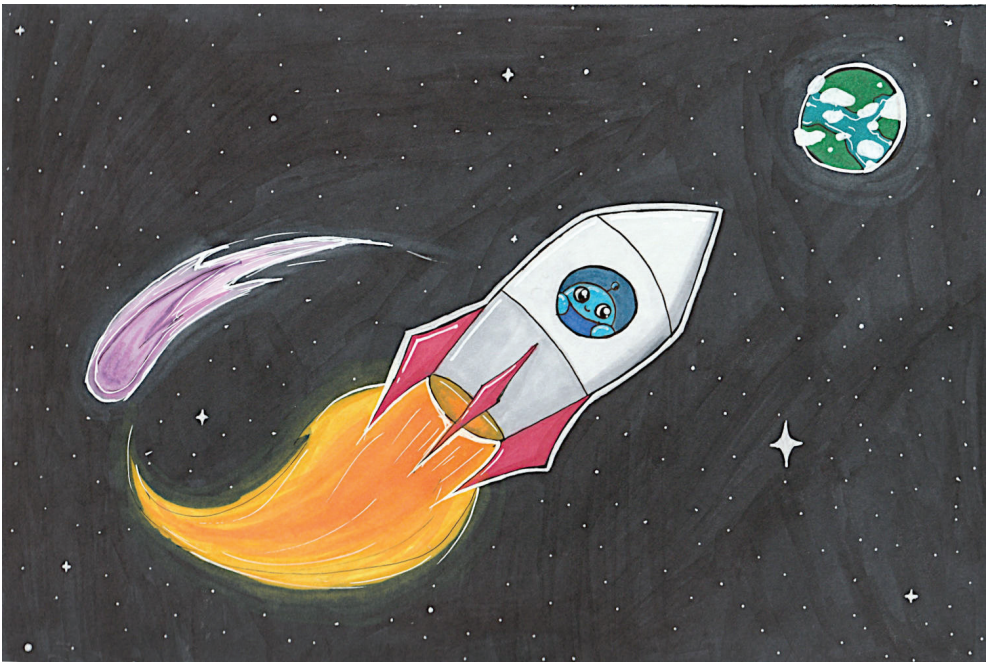
Полный код можно найти по адресу **ludolf.github.io/en/3.txt**.



Задача № 57

Запусти программу с помощью панели управления.

Итак, исследователь космоса благополучно вернулся домой!



Упражнение

Для выполнения следующего упражнения скопируй полный код в панель управления **ludolf.github.io** на своём компьютере или смартфоне.

1. Перепиши программу так, чтобы робот двигался против часовой стрелки, а не по часовой.
2. Перепиши программу так, чтобы робот начинал собирать энергию с правого верхнего угла, а не с левого нижнего.
3. Напиши программу, которая пытается обойти кратер сразу после того, как его увидит, а не после того, как в него войдёт.

Прощай, комета! – *эпилог*

Все готово к старту. Лудолф бросает последний взгляд на приборы системы управления и видит успокаивающий зелёный свет. Программа работает исправно. Обратный отсчёт начинается, до старта остаются две минуты. Слышен шум ракетных двигателей. Всё дрожит. Системы работают на максимальной мощности. Космическую экспедицию ждёт ещё одна задача. Одна минута до начала. Тепло двигателя плавит кристаллический песок и поднимает искрящуюся пыль. Полёт начинается. *Три*. Пристегни ремни безопасности. *Два*. Сохраняй устойчивое положение. *Один*. Ракета взмывает в чёрное небо и направляется к синему дому вдали на горизонте.

Робот в последний раз смотрит в окно и прощается с фиолетовым миром своим ослепительным хвостом. Прощай, комета! А может, и до свидания. Может быть, мы снова увидимся через сто лет!

Любую компьютерную программу всегда можно написать несколькими способами. В одних случаях программа может получаться короткой, в других – длинной, в одних – быстрой, в других – медленной, в одних – простой и понятной, в других — сложной и запутанной. В предыдущей главе я показал только одно из возможных решений. Подумай о решённой нами задаче ещё раз и попробуй придумать другую программу, которая точно так же вернёт Лудолфа домой.



Заключительная задача

Удали ранее написанный код и начни с нуля.

И не мошенничай!

Сравни новую программу со старой, когда она будет готова. Они, безусловно, будут иметь отличия, но при этом выполнять одну и ту же работу. У каждого, как известно, свой вкус.

Приложение

Решения

Задача № 1

Человек! Робот спроектирован так, чтобы быть максимально похожим на человека, потому что на планете нет существа умнее.

Задача № 3

```
00010110
00011111
00001110
00011001
00010110
00100000
```

Задача № 4

```
00010110 Л
00011111 У
00001110 Д
00011001 О
00010110 Л
00100000 Ф
```

Задача № 5

JavaScript, Python, Java, C и C++ – одни из самых популярных языков программирования. Все они имеют разную историю и применяются для разных целей. Например, JavaScript используется для создания веб-сайтов, Python – для автоматизации, а C и C++ – для программирования устройств, таких как стиральные машины.

Задача № 6

Завязать шнурки можно несколькими способами. Твоя программа может быть совершенно другой, и это нормально.

1. Удерживая концы шнурка, перекрести их.
2. Заверни один конец вниз и затяни.
3. Создай две петли (как ушки зайчика).
4. Перекрести образовавшиеся петли.
5. Заверни петлю под низ и затяни.
6. Готово!

Задача № 7

```
00000000
01111100
01100110
01100110
01111100
01100110
01100110
01111100
```

Задача № 8

Интернет!

Задача № 9

В XVIII–XIX веках изобретение паровой машины открыло новый способ производства энергии, что внесло огромный вклад в промышленную революцию и ускорило экономическое и социальное развитие во всём мире.

Задача № 10

Люди – это нечто большее, чем компьютеры и роботы. Ни один компьютер не может смеяться или любить кого-то.

Задача № 11

```
00000010 2
00000011 3
00000000
00000000
00000000
00000000
00000000
00000000
```

Результат умножения перезаписывает первый байт, поскольку после вычисления результата входные числа больше не нужны:

```
00000110 6
00000011 3
00000000
00000000
00000000
00000000
00000000
00000000
```

Задача № 13

Предложение `"Привет, Мир!"` сначала сохраняется в памяти под именем `message`. Но, поскольку оно используется только один раз, нет никакой необходимости давать ему имя.

Это означает, что программу можно упростить так:

```
print("Привет, Мир!")
```

и

```
console.log("Привет, Мир!")
```

Задача № 14

```
5 % 4    = 1
55 % 5   = 0
222 % 10 = 2
```

Задача № 15

```
"a" + 0 + "b" = "a0b"
"a" + 1 + 2   = "a12"
"" + 1 + 2    = "12"
1 + "a"       // ERROR! Операция прибавления текста
              // к числу не имеет смысла.
```

Задача № 16

Выражение	**Результат**
`True & True`	`True`
`True & False`	`False`
`False & True`	`False`
`False & False`	`False`

Задача № 17

Выражение	Результат
`True | True`	`True`
`True | False`	`True`
`False | True`	`True`
`False | False`	`False`

Задача № 18

Выражение	Результат
`! True`	`False`
`! False`	`True`
`!! True`	`True`
`!! False`	`False`

Задача № 19

```
1 + 2 * (2 + 1) =
1 + 2 * 3       =
1 + 6           =
```

7

Задача № 20

```
True | False & False =
True | False         =
```

True

Задача № 21

```
x := 1
x = 2 =
1 = 2 =
```

`False`

Задача № 22

Имя	Допустимое	Причина
`a`	да	
`A`	да	
`ab`	да	
`a b`	нет	пробелы в именах не допускаются
`a_A`	да	
`_ab`	да	
`a__b_`	да	
`1`	нет	имя не может начинаться с цифры
`a1`	да	
`1a`	нет	имя не может начинаться с цифры
`a12b`	да	
`a%`	нет	символ `%` не может присутствовать в именах
`a#b`	нет	символ `#` не может присутствовать в именах

Задача № 23

256

Задача № 24

```
x := 1
y := 1 + 1
x := 1 + (1 + 1)
```

y = 2
x = 3

Задача № 25

```
x := 100        // 100
x := 100 + 1    //101
x := 101 + 101  // 202
x := 202 - 1    //201
```

Задача № 26

```
x := 1
a := [1, 1 + 1, 1 + 1]
x := 1 + 1
```

```
a = [1, 2, 2]
```

Присваивание нового значения переменной x не влияет на значения в массиве, поскольку они никак не связаны.

Задача № 27

```
cube := [
  [
    [1, 2],
    [3, 4]
  ],
  [
    [5, 6],
    [7, 8]
  ]
]
```

Задача № 28

```
cube[1,0,1] = 6
```

Задача № 29

```
array := [1,2]
array := array + [3] + array =
         [1,2] + [3] + [1,2] =
         [1,2,3,1,2]
```

Задача № 30

2, 3, 3

Задача № 31

```
a1 := [1,2]
a2 := a1
a3 := a2

a1[0] := 4
a3[0] := 5
a2[0] := 4 + 1

x := 4 + 5 + 4 + 1
   = 14
```

Задача № 33

Ты можешь создать тысячи таких программ. Вот одна из них:

```
if x < 1 {
  x := x + 1
  if x < 1 {
    x := 1
  }
}
```

Задача № 34

```
x := 1

if 1 > 1 { // False
  x := 2   // не будет выполнена
}
else {
  x := 3   // будет выполнена
}

x = 3
```

Задача № 35

```
x := 1

if 1 > 1 {       // False
  x := 2         // не будет выполнена
}
else if 1 <= 1 { // True
  x := 3         // будет выполнена
}
else {           // не будет выполнена
  x := 4
}

x = 3
```

Задача № 36

```
x := 0

if 0 > 0 {       // False
  x := 1         // не будет выполнена
}
else if 1 >= 0 { // True
  x := 2         // будет выполнена
}
else if 1 >= 0 { не будет выполнена
  x := 3
}

x = 2
```

Задача № 37

Выполним команду шаг за шагом и изменим значение переменной x:

```
x := 3
```

1.

```
while 3 > 1 { // True
  x := 3 - 1  // будет выполнена
}
```

2.

```
while 2 > 1 { // True
x := 2 - 1    // будет выполнена
}
```

3.

```
while 1 > 1 { // False
  x := 1 - 1  // не будет выполнена
}
```

```
x = 1
```

Задача № 38

Поскольку переменная x увеличивается в каждой итерации, то цикл выполнит пять итераций, после чего переменная x получит значение 10:

Итерация	**x**	**y**	**x < 10**
0	0	0	True
1	2	1	True
2	4	2	True
3	6	3	True
4	8	4	True
5	10	5	False

Задача № 39

```
while y < size {
  while x < size {
    position := Map[x,y]
    x := x + 1
  }
  y := y + 1
}
```

Задача № 40

```
0 + 4 + 8 = 12
```

Задача № 41

```
add := (x, y, z) {
  x + y + z
}
nine := add(2, 3, 4)
```

Задача № 42

Если в третьем параметре передать нуль, то результат сложения двух других чисел не изменится. Всё, что прибавляется к нулю, остаётся прежним:

```
five := add(2, 3, 0)

2 + 3 + 0 = 5
```

Задача № 43

```
energy := 0

is_low_energy := () {
  energy < 10
}

collect_energy := () {
  energy := energy + 1
}
```

Задача № 44

```
1 + (1 + 3 + 1) = 6
```

Задача № 45

```
increase := (x) {
  x + 1
}

increase(0) = 1
increase(5) = 6
```

Задача № 47

```
o := { a:1 }
q := o

q.a := 2

x := 2 + 2
```

x = 4

Задача № 48

`{x: 1, y: 2}` и `{x: 5, y: 3}`

Задача № 49

`{x: 2, y: 4}`

Задача № 50

```
x := 1.plus(2).plus(3)
```

Задача № 56

```
y := bottom
while y <= top {
  Ludolf.up()
  charge_energy()
  y := y + 1
}
left := left + 1

x := left
while x <= right {
  Ludolf.right()
  charge_energy()
  x := x + 1
}
top := top - 1

// движение вниз
y := top
while y >= bottom {
  Ludolf.down()
  charge_energy()
  y := y - 1
}
right := right - 1

// движение влево
x := right
while x >= left {
  Ludolf.left()
  charge_energy()
  x := x - 1
}
bottom := bottom + 1
```

Благодарности

Иллюстрации: Томас Тулка, Катерина Флекова
Обложка: Катерина Флекова

Книги издательства «ДМК Пресс»
можно купить оптом и в розницу на складе издательства по адресу:
Москва, ул. Электродная, д. 2, стр. 12, офис 7, тел. **+7 (499) 322-19-38**,
а также заказать на сайте **www.dmkpress.com**
с доставкой в любой регион РФ

Томас Тулка

**Цифровые приключения:
как программируют компьютеры**

Главный редактор *Мовчан Д. А.*
Зам. главного редактора *Яценков В. С.*
editor@dmkpress.com
Перевод *Киселев А. Н.*
Корректор *Абросимова Л. А.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 11.54. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**

Присоединяйтесь к маленькому роботу,
отправленному в фантастическую экспедицию
в этой интерактивной детской книге!
Начните исследовать мир компьютеров,
изучите их язык и узнайте,
как они способствуют проникновению
цифровых технологий в нашу жизнь!

Благодаря забавным иллюстрациям и увлекательным заданиям эта книга сделает сложные идеи простыми для понимания. По мере развития сюжета вы откроете для себя основы программирования и приобретете навыки решения задач с помощью кода. Это удивительное приключение понравится любознательным детям, которые любят сложные задачи. Оно пробудит в них интерес к информатике и языкам программирования и покажет, насколько безграничны возможности современных технологий.

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
www.дмк.рф

ISBN 978-5-93700-420-8
9 785937 004208 >