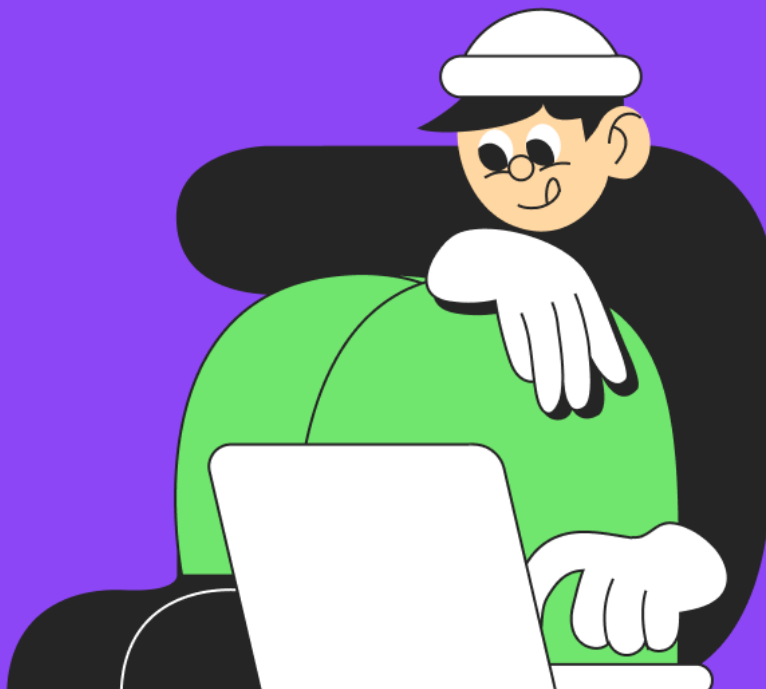
GeekBrains

Что такое программирование и языки программирования



Оглавление

Заключение

[00.01.35]
Приветствие

Всем привет! Это первый урок курса «Введение в программирование».

Меня зовут Ильнар Шафигуллин. Я методолог программы «Разработчик» в компании GeekBrains. Помимо этого, я преподаватель Казанского университета на механико-математическом факультете и закончил, соответственно, мехмат. Стаж преподавания — более 10 лет. Я кандидат физико-математических наук, то есть по образованию математик. Защитил диссертацию по вещественному, комплексному и функциональному анализу. То есть это математика без реального применения. Тем не менее с наукой я немного знаком.

Что касается программирования, я преподаю программирование в основном на языке Java. Это мой основной язык как преподавателя.

Помимо этого, я работал в области Computer Vision. Это Data Science. Мы делали проект по подсчёту количества пассажиров в автобусе, когда над входом в автобус стоит камера. То есть на основе видеопотока с видеокамер мы можем посчитать пассажиров, которые заходят и выходят из автобуса. Соответственно, там должен быть запрограммирован не Java, а Python. Но к этому мы ещё вернёмся. Таким образом, есть опыт не только в преподавании, но и в продакшене у меня большой.

[00.03.29]
На этом уроке

1. Поговорим о языках программирования. Узнаем, почему их так много и чем они друг от друга отличаются.
2. Обсудим алгоритмы и о том, как их надо записывать.
3. Посмотрим решение первых задач, как простых, так и более сложных.

Чтобы провести время достаточно продуктивно, приготовьте листочек и ручку (маркер, карандаш). Во время занятия будет несколько заданий, которые крайне желательно выполнять, для этого я буду специально давать некоторое время.

Помимо этого, вы, естественно, можете писать в комментариях или под уроком любые вопросы и эмоции, которые у вас возникают. Я также буду просить вас писать что-то в комментариях. Но, кроме этого,

подготовьте, пожалуйста, листочек и ручку. Выделим для этого буквально одну минуту, а за это время люди подтянутся к трансляции.

Далее мы напрямую перейдём к тому, что такое языки программирования, как их надо учить, и какие с этим возникают сложности.

Пока у нас есть немного времени, можете написать что-то в комментариях, если листочки и ручки уже приготовлены.

[00.05.03]

Языки программирования и подходы к обучению

Есть ли жизнь после Java и призмы языков программирования

Есть один небольшой риторический вопрос: «Есть ли жизнь после Java и призмы языков программирования?». Разберёмся, что здесь имеется в виду.

Я преподаю язык программирования Java. Это язык с достаточно сильно выраженными особенностями. То, как пишут на языке Java, редко пишут на других языках. В связи с этим есть проблемы.

Когда люди только начинают учить язык программирования, они считают, что программирование целиком и тот язык программирования, который учат сейчас — одно и то же. Это приводит к следующему: вы научились писать на одном языке и думаете, что всё программирование такое, а когда начинаете писать на другом языке, то повторяете принципы первого языка.

Для джавистов есть даже такое обидное выражение: «Java головного мозга». Потому что те, кто начал с языка программирования Java, продолжают писать так, как было в первом языке. Это свойственно не только для Java, но и практически для любого другого языка. Поэтому важно понимать, что язык программирования и программирование целиком — разные вещи.

У вас может возникнуть вопрос: «В чём проблема? Я сейчас учу один язык, неважно какой, а потом, если понадобится, выучу другой». Да, так действительно можно. Но переход с одного языка на другой доставляет много сложностей.

Например, вы сильно углубились в какой-то один язык и не прочувствовали момент, когда ушли в особенности этого языка. В этом случае вы не поняли программирование, как составление алгоритмов. Поэтому познакомимся с этим чуть ближе.

[00.05.32]

Пример с ребёнком и молотком

Возьмём пример из реальной жизни. Например, вы поняли, как делать какую-то одну вещь, и решили, что всё делается так.

Если дать маленькому ребёнку в руку молоточек, то всё вокруг него станет гвоздём. Он будет тюкать им по всему: по телевизору, кружке, столу и по гвоздю соответственно. Если мы научились чему-то, наше познание может сузиться до этого. Нам необходимо мыслить немного шире.

[00.07.44]

Пример с изучением иностранных языков

Возьмём пример, связанный с изучением иностранных языков. Когда вы только начинаете изучать языки, возникает большая проблема.

Вы поняли перевод слов, знаете, как, например, с русского языка перевести на английский, но когда строите предложения и делаете дословный перевод, грамматика не соблюдается, а люди вас плохо понимают на английском языке. Недостаточно заменить слова и оставить ту же конструкцию, надо понять, какая конструкция должна быть изначально.

На слайде есть несколько мемов на эту тему. Но основная идея заключается в том, что недостаточно перевести слова и оставить их в той же конструкции. Получится то же, что и с «Java головного мозга».

Вы взяли новый язык программирования, поменяли слова из языка Java на слова из языка, например, Python, а в результате почему-то ничего не работает или работает, но над вами все смеются, потому что так делать не надо. То же самое связано и с обычными языками, которые мы учим.

Вы можете попробовать заменить русские слова на английские, но если оставить их в пределах правил русского языка, то ничего хорошего не получится. Да, люди вас, наверное, поймут, додумают, но в программировании такое не работает. Компьютер вряд ли сможет додумать всё за вас.

Поэтому важно понимать, как и на каком языке надо писать, какие особенности имеет тот или иной язык программирования.

[00.09.20]

Примеры со словами «собака» и «привет»

Раз мы начали говорить об обычных языках, на которых общаемся, скажем, что сейчас существует более семи тысяч языков. Естественно, их было больше: какие-то исчезли, а какие-то остались. Но на

сегодняшний день по данным «Википедии» существует более семи тысяч языков, на которых люди могут общаться. На сорока наиболее популярных говорят две трети населения планеты.

А теперь подумаем, о чём эти люди могут на этих языках говорить.

Возьмём, например, слово «собака». На русском языке — «собака», на английском — dog, на немецком — hund, и ещё более сорока языков, на которых это слово можно произнести. Но сама собака, как животное, живое существо, от этого никак не поменяется. Мы почему-то называем её по-разному, но сам объект остаётся таким же.

Или слово «привет». На русском языке оно произносится как «привет», на французском — bonjour**,** на английском — hello и т. д. Опять же, смысл не меняется, но средства его донесения у каждого языка могут быть разные. Где-то похожие, а где-то нет.

Таким образом, важно понимать, что возникает следующая конструкция: есть смысл, который мы хотим донести, и средства донесения этого смысла. В такой ситуации язык, на котором мы разговариваем, считается средством донесения этого смысла, но никак не смыслом. То есть собака не перестанет быть собакой, если мы называем её hund, а не «собака», как привыкли.

[00.11.01]

Пример со звучанием лягушки в разных странах

Есть ещё несколько примеров, которые мне очень нравятся.

Например, мы привыкли, что лягушки квакают. И в детских сказках, и в обиходе считается, что если лягушка издаёт какой-то звук, то это, скорее всего, «ква-ква». Но если вы спросите у какого-то иностранного гражданина, например, у англичанина или кого-то ещё, как, на его взгляд, квакают лягушки, то удивитесь. Вроде бы те же самые звуки, а произносят они их совершенно иначе.

Например, на слайде, где изображена лягушка иностранного гражданина, указано слово «риббит». Так издаёт звуки лягушка, по мнению англичан. То есть у них они не квакают, а говорят «риббит». Есть ещё «бреккеке» и «уп-уп» — предположительно, так говорят лягушки в Индонезии.

Вы можете выполнить одно лёгкое упражнение — набрать в Google «как квакают лягушки в разных странах» и посмотреть множество примеров, где люди пытаются подражать лягушкам.

Однако важно понимать, что лягушки на самом деле квакают одинаково, плюс-минус выходит один и тот же звук. Это люди интерпретируют его по-разному.

Возвращаемся к тому, что есть смысл и звуки, которые издают лягушки, петухи и другие животные, а есть интерпретация, средства донесения этого смысла, используемые людьми в разных странах. Это связано со спецификой наблюдения за животными в разных странах, с особенностями строения языков — то, как

люди разговаривают и произносят звуки. Так или иначе, мы получаем некоторое искажение, интерпретацию смысла.

Таким образом, есть смысл и средства донесения смысла.

[00.12.52]

Пример описания цветов радуги в разных языках

Другой пример связан с радугой. Есть радуга гражданина РФ и есть радуга иностранного гражданина. Выясним, в чём здесь смысл.

Мы с детства привыкли, что у нас есть семь цветов радуги. Все знают фразу «Каждый охотник желает знать, где сидит фазан». Но если посмотрим на английский язык, то в нём нет голубого цвета. Есть blue — «синий», light blue — «светло-синий», а голубого в нашем понимании нет. И сама радуга у них состоит не из семи цветов, а из шести.

Можно найти примеры языков и стран, где есть люди, которые считают, что радуга состоит из пяти цветов. Однако очевидно, что само атмосферное явление, радуга на небе, одна и та же. Мы видим её одинаково, но ввиду каких-то особенностей интерпретируем по-разному. Для нас очевидно, что цветов семь, но в других языковых группах их может оказаться иное количество: где-то больше, а где-то меньше. Просто люди так привыкли.

То же самое с языками программирования. Надо запомнить, что есть смысл и есть средства донесения (интерпретация) этого смысла.

[00.14.11]

Языки разных сфер

Однако не существует единственного хорошего языка. Нет какого-то языка, который идеально подходит для всего. У каждого языка есть какая-то сфера применения, в которой он проявляет себя лучше, чем остальные.

Исторически сложилось, что русский язык хорош для поэзии, прозы, написания романов. В качестве примера мы добавили на слайд дуб, так как все, скорее всего, помнят или, по крайней мере, слышали, что Лев Толстой в «Войне и мире» мог по 1–2 страницы описывать, как колышутся листья на ветру.

Но если мы говорим о немецком языке, то принято считать, что он хорош для юриспруденции. Этот язык более точный, он не предполагает нескольких смыслов. Да, там предложения размером с абзац, но в юриспруденции это допустимо, и двойных толкований нет.

Английский же язык подходит для математики и программирования.

Я математик. Мне, конечно, удобно пользоваться русским языком, когда пишу какие-то статьи или что-то ещё. Но когда пишешь математическую статью на русском языке, то часто возникает двойственность. Написанное на русском языке может интерпретироваться по-разному, а в математике это недопустимо. И у нас даже есть такое упражнение:

1. Пишем статью на русском языке.
2. Переводим её на английский язык.
3. Переводим статью обратно на русский язык.

В процессе у нас возникает понимание, что что-то не так. То, что мы написали на русском языке, кажется для нас очевидным. На самом деле, с точки зрения математики, всё далеко не очевидно и может быть интерпретировано по-разному. В этом смысле английский язык более строгий. И для математики, и для программирования он подходит гораздо лучше, чем русский язык.

Однако важно понимать, что основные публикации на научные, а именно: математические, темы проходят на английском языке. У этого есть несколько причин. Одна из них — английский язык более строгий.

[00.16.35]

Язык математики

Немного математики. На слайде указано определение предела последовательности. Оно может показаться страшным и непонятным. Не страшно, если вы ничего не понимаете, смысл не в этом.

Перед нами обычная математическая запись. Когда мы хотим дать какое-то определение или понятие, то пользуемся такими значками. Эти значки называются «кванторы» и несут в себе какой-то смысл. Если по-русски прочитать то, что здесь написано, получим следующее:

Для любого эпсилон больше нуля существует некоторый N, зависящий от этого эпсилон, такой, что для всех каких-то других номеров, больше, чем тот номер, который мы нашли, будет выполнять следующее неравенство. Если это выполняется, последовательность Xn сходится к числу a.

Для кого-то это может быть полнейшей абракадаброй. И это нормально. Но для математиков всё предельно ясно. Неважно, откуда этот математик. Он может быть из Китая, России, Европы, Америки или Танзании. Если человек знаком с математикой, для него это становится понятным.

Всё, что я произнёс до этого, можно написать на русском языке. Смысл при этом сохранится полностью. Можно сделать это аккуратно, чтобы не было двойных толкований, а смысл передался. Но если мне понадобится донести эту информацию до кого-то другого, например, до китайского математика, то весь

текст, который я написал, а занимать он может 3–4 предложения, потребуется перевести на другой язык. Это вызовет много сложностей.

Прямого перевода может не быть, и мне придётся адаптировать то, что хочу сказать, на другой язык — это вызовет много сложностей. Однако если заранее договориться, что означают те или иные значки, то можно очень сложные понятия и определения доносить до других математиков. И все всё поймут.

Если до этого мы говорили, что есть русский язык, немецкий и английский, то для других сфер можно придумать ещё какие-то языки. Например, математики придумали такой язык (как на слайде).

Представим себе нотную грамоту. Разлинованная тетрадь с точками и закорючками — это мелодия. Её тоже можно было бы записать как-то иначе, но договорились записывать в нотной грамоте. И все музыканты в целом понимают, о чём идёт речь.

Таким образом, для каждой сферы удобен тот или иной язык. И это не только привычные нам языки, на которых общаемся, но и экзотические языки, удобные для конкретной сферы применения.

[00.19.44]

Поговорим на одном языке с исполнителем

Снова рассмотрим некоторые мемы и разберёмся с тем, что здесь есть.

Мы поняли, что существует множество различных языков. Остаётся понять, на каком же языке нам надо говорить в тот или иной момент.

Представьте, что у вас есть знакомый шеф-повар, которому вы решили помочь. И он просит вас снять кожицу с помидора. Если повар будет пользоваться профессиональным языком, на котором он общается на кухне в ресторане, то скажет: «Бланшируй томаты». Для людей, хорошо знакомых с процессом приготовления пищи, кто уже работал на кухне, будет всё понятно. Но если вы увидели слово «бланшируй» в первый раз, то можете не понять, что же надо сделать. Исходя из того, кто выступает слушателем информации, которую хотите донести, надо выбирать правильный язык.

Если мы хотим, чтобы с помидоров сняли кожицу, потребуется сказать несколько по-другому: «Сделай надрезы на помидорах, опусти их на несколько минут в кипящую воду, потом достань и положи в холодную воду. После этого через несколько минут достань помидоры из воды и сними с них кожицу». Таким образом, поймёт обыватель. А если мы говорим о шеф-поваре на кухне, который общается с опытными сотрудниками, он скажет: «Бланшируй томаты».

Это относится ко всем сферам. Если мы говорим о математике и при этом пытаемся объяснить то, что делаем, не математику, писать квантово смысла нет. А если говорим о нотной грамоте и хотим ей пользоваться, надо понимать, что человек, которому мы это объясняем, знает её.

[00.21.45]

Практика

Задача 1

Мы попробуем решить одну маленькую простую задачу, но решать будем её в некоторых ограничениях. Человек, который будет выполнять наше решение, может сделать только несколько определённых действий. Посмотрим, что это за задача.

Представим, что у нас есть пять гирь разного веса. Наша задача — найти вес самой тяжёлой гири. Человек, который будет выполнять наши поручения, может выполнять только несколько действий:

1. Взять одну или несколько гирь.
2. Сравнить гири по весу.
3. Поменять все гири местами.
4. Запомнить вес каждой гири.

Я добавлю немного реквизита в нашу студию. Вы пока можете на листочках готовить решение. А чтобы вам было понятнее, возьмём наши импровизированные гирьки и поставим на стол.

Предположим, что у нас есть пять таких гирь — слева направо. Тяжёлые гири мы решили не брать. Вам необходимо написать какое-то решение, какой-то алгоритм, по которому ваш друг, выполняя указанные действия, сможет найти самую тяжёлую из них.

Засечём пять минут, чтобы вы подумали над решением задачи подумать. Я же повторю её суть.

Мы имеем пять гирь. Можно сказать: «Возьми первую и четвёртую. Сравни по весу. Найди самую тяжёлую».

Хорошо. Я нашёл, что гиря с массой 6 тяжелее гири с массой 1.

Потом можно сказать: «Возьми третью и пятую. Сравни их».

Ок. Я знаю, что третья тяжелее.

Затем можно сказать: «Возьми вторую и четвёртую».

То есть надо записать какой-то алгоритм. Алгоритм — это последовательность действий, указаний, которые надо совершить, чтобы по результатам сравнить, какая из гирь самая тяжёлая.

Важно! *Когда вы смотрите на экран и видите номера всех гирь (их массу), то можете сказать, что, очевидно, вторая самая тяжёлая. И алгоритм, который будет работать так: «Возьми вторую, она самая тяжёлая», конечно, сработает правильно.*

Однако, если потом попытаемся это решение применить к другому случаю — возьмём вторую гирю и скажем, что она самая тяжёлая, решение будет неправильным. То, что вы заранее видите размер каждой гири, конечно, хорошо. Но в общем случае, когда мы захотим решить задачу с разными гирями, алгоритм не сработает.

Поэтому попробуйте придумать решение сейчас. Напишите на листочке или в комментариях, каким образом, пользуясь любыми из указанных на слайде действиями, можно найти самую тяжёлую. Отведём на это пять минут.

Итак, мы снова в эфире. Я надеюсь, что все смогли придумать хоть какое-то решение.

Сделаем маленькое лирическое отступление. Когда вы чему-то учитесь, у вас есть два основных варианта:

1. Формат потребления контента.
2. Формат участия в процессе.

Когда вы смотрите видео на YouTube какого-то популярного блогера или что-то подобное, в большинстве случаев это считается потреблением контента. В таком случае, переключаясь на другое видео, информация предыдущего забывается.

То же самое может быть и здесь. Нашу лекцию можно смотреть, не участвуя в процессе, не делая каких-то заметок и заданий, которые мы сейчас будем разбирать. Я покажу, как они решаются. Но если вы заранее задумались об этой задаче, попробовали её решить, вам станет гораздо интереснее и понятнее решение, которое будет дано. Потому что если я просто покажу решение, то многие подумают: «Ну да, всё очевидно и просто», и в голове это не отложится.

Ранее сказанное относится не только к этой задаче. Дальше пойдут другие задачи, более сложные. Поэтому я прошу отнестись к этому более ответственно — заниматься не просто потреблением контента, а участвовать в процессе обучения. Особенно это касается людей, которые будут это смотреть в записи, а не на прямой трансляции.

Чаще всего эту задачу предлагается решить следующим образом: чтобы исключить возможность видеть цифры на, я отверну импровизированные гири, и видеть цифры мы будем только тогда, когда будем на них смотреть.

Чаще всего предлагается примерно такое решение:

1. Берём первые две гирьки: 1 и 8. Для вас — слева направо. Сравниваем их между собой. Ту, что тяжелее, перетаскиваем дальше. 8 тяжелее, поэтому будет стоять чуть правее. Оставляем их.

2. Берём следующие две гирьки, то есть вторую и третью. Сравниваем их. 8 тяжелее 3. Значит, 8 и 3 меняем местами, чтобы 8 была ближе к правому краю. Прячем их номера обратно и не запоминаем. Всё, что мы делаем — просто переставляем, ничего не запоминая.

3. Берём следующие две гирьки. Это 8 и 6. 8 тяжелее 6, поэтому 8 опять перейдёт в правую сторону. А так как мы их не запоминали, то снова прячем. Забываем про них.

4. Берём последние две гирьки. Это 8 и 2. 8 опять побеждает. Переставляем их и прячем.

5. Гирек не осталось. Сравнивать больше нечего.

Учитывая, как мы записали это решение, самая тяжёлая гирька оказалась крайней справа. Это гиря с весом 8.

Всё хорошо — алгоритм работает. Но у него есть некоторые излишества, дополнительные вещи, которые мы делали, чтобы найти максимум. То есть нашей задачей было найти вес самой тяжёлой гири, а мы ещё её перетаскивали.

Представьте, что это не стаканчик с нарисованной восьмёркой, а восьмикилограммовая гиря, или ещё хуже — что-то более тяжёлое, например, большие пудовые гири или штанги, которые пришлось бы переставлять с места на место. Это достаточно тяжело и избыточно. Нас не просили этого делать — ставить на край самую тяжёлую гирю. Задачей было найти вес самой большой гири. И здесь решение, которое, я уверен, многие нашли, можно немного видоизменить.

Для чистоты эксперимента я верну всё как было. Обратите внимание, что мы не запоминали значение, то есть не использовали нашу память, а только перетаскивали объекты.

Теперь, чтобы продемонстрировать второе решение, мне нужен помощник — морковка. Те, кто смотрел занятие у Игоря по умению учиться, скорее всего, знают технику Pomodoro. Там у вас были будильники в виде помидорки. У меня тоже был будильник в виде морковки, но, к сожалению, от него осталась только верхняя часть, и сейчас она нам поможет.

Решение:

1. Берём один стаканчик. Будем считать, что этот первый стаканчик самый тяжёлый из всех просмотренных. Действительно, я посмотрел только один стаканчик, то есть взял одну гирьку. И из всех, что я видел, это самая тяжёлая. Соответственно, на самую тяжёлую я буду ставить нашу морковку.

2. Теперь берём вторую гирьку и сравниваем её с той, которая сейчас считается самой тяжёлой. 8 тяжелее 1.

3. Перекладываем знак самой тяжёлой гирьки на 8. Про 1 я забыл и помню только то, что пока самая тяжёлая находится здесь — она идёт второй.

4. Берём следующую гирьку и сравниваем её с той, что сейчас считается самой тяжёлой — 8. 8 победила, поэтому знак самой тяжёлой гири остаётся на ней.

5. Берём следующую гирю, сравниваем — опять 8 победила, самой тяжёлой остаётся она.

6. Сравниваем самую тяжёлую гирю — 8 — с последней. 8 снова победила. То есть самой тяжёлой остаётся гиря с весом 8.

Когда мы нашли самую тяжёлую гирю, нам не пришлось их (гири) перетаскивать.

На самом деле в программировании можно решать разными способами. Первое решение, которое я дал, требовало каких-то действий, дополнительных манипуляций. А для второго решения использовалась дополнительная память, нам надо было запоминать, где находится тот или иной объект — хотя бы один. То есть в первом случае мы больше работали, так сказать, руками, а во втором — головой.

Далеко не всегда один способ применим для всех ситуаций. В одном случае лучше переставлять, а в другом — запоминать. Всё зависит от конкретной ситуации. Мы об этом ещё поговорим.

Обратите внимание, что решений может быть несколько. Я уверен, есть и другие решения. Например, когда вы перебираете гири не подряд, а устраиваете соревнования, где первые две гири участвуют в первом туре, вторые — во втором, а третья будет бороться с победителями. То есть сначала мы сравнили первые две гири и нашли самую тяжёлую, далее сравнили вторую пару, а затем проделали то же самое с другими комбинациями.

Таким образом, решений может быть масса. Просто какие-то решения более рациональные по количеству действий, которые надо совершить, а также по объёму используемой памяти. Например, сейчас мы использовали только одну дополнительную ячейку, чтобы запомнить, какая гирька была самой тяжёлой. Другие какие-то решения могут быть не всегда рациональными.

Так как компьютеры сейчас работают очень быстро, вы далеко не всегда будете замечать, что ваше решение не оптимально — придётся выполнять дополнительные действия. Но если мы говорим про работу с большими системами, то там это очень важно. Иначе ваш условный сайт начнёт тормозить, а это уже никому не понравится.

Посмотрим, как решение, которое мы воспроизвели, можно записать.

[00.36.50]

Текстовое написание решения задачи 1

Один из способов записи — посредством русского языка.

Все совершённые действия мы записали на русском языке:

1. Берём первую гирю и считаем, что сейчас она самая тяжёлая.
2. Берём вторую гирю и сравниваем её с первой.
3. И так далее.

Всё, что мы сделали, можно записать на русском языке. И люди, которые русский язык понимают, в целом это решение смогут воспроизвести. То есть ничего дополнительного для этих людей не требуется.

Но если мы хотим, чтобы это решил человек, который русский язык не знает, а знает, например, английский, китайский или немецкий, возникнет гора проблем. Нам потребуется перевести всё на другой язык, записать это с учётом его особенностей, и только тогда человек сможет воспроизвести решение.

Если мы говорим про машину, то ситуация ухудшается. Додумать она не сможет, а решение потребуется произвести точное.

Поэтому для записи алгоритмов, а то, что мы рассматривали, и есть алгоритм — последовательность действий, которые необходимо выполнить — используется такая форма записи.

Чтобы записи этих алгоритмов могли воспроизводить разные люди, используются блок-схемы.

[00.39.38]

Блок-схема 1

Посмотрим на нашу блок-схему и разберёмся в том, что здесь происходит. А происходит то же, что мы делали руками. Однако представлено это в виде схемы.

Если на неё посмотреть внимательнее, она становится интуитивно понятной: есть набор стрелочек, какие-то вопросы и действия, которые совершаются. Рассмотрим всё детальнее.

С краю вынесен отдельный блок, и сказано: «Даны 5 чисел — 1, 8, 3, 2, 6». Так стояли гири, когда мы решали свою задачу.

Что мы делаем:

1. Берём самую первую гирю — с весом 1 — и говорим, что она максимальная.

То, что я надевал морковку на стаканчик, считается присваиванием какой-то переменной. Переменная — это некий лейбл, метка, которая может быть добавлена на значение. И когда я обращусь к максимуму, придётся найти то, с чем она связана.

Вспомним нашу морковку. Она находилась на гирьке 8. Когда надо было обратиться к максимальной переменной, к гирьке, не смотря на номер, я понимал, если морковка стоит на какой-то из них, значит, максимальный вес здесь.

На блок-схеме мы делаем то же самое, но вместо морковки — более программистский подход. Берём переменную (лейбл) и присваиваем ей то значение, которое хотим. В нашем случае мы взяли первую гирьку и сказали, что она будет максимальной.

2. Сравниваем следующую гирьку — 8 — с максимальной.

   Если вдруг 8 оказывается больше, идём по стрелочке Yes («Да») и говорим, что теперь максимальной будет 8. А если 8 окажется меньше 1, тогда пойдём по стрелочке No («Нет») и будем сравнивать объекты дальше.

Если внимательно посмотрите на слайд, то увидите решение, которые мы воспроизвели.

1. Взяли первую гирьку и сказали, что она максимальная.
2. Сравнили со второй гирькой. Оказалось, что гирька с весом 8 тяжелее. Теперь максимальной стала 8.
3. Взяли следующую гирьку, её масса 3, сравнили с максимальной — 8.
4. Снова пошли по следующей ветке. Если 3 меньше 8 — идём вправо, по ветке No.
5. Сравнили следующие гирьки — 2 и 8. 8 опять оказалась тяжелее, поэтому идём по стрелочке No.
6. Сравниваем 6 и 8. 8 оказывается максимальной.

То, что мы написали в пяти предложениях на предыдущем слайде, можно записать в виде такой блок-схемы. Да, чтобы её понять, требуется некоторое понимание блок-схем, немного с ними поработать, но это не очень сложно. Если вы понимаете, что мы идём от красного блока Start по стрелкам, то, как в детском лабиринте, переходим с одного объекта на другой, идём с одной стрелки на другую и так доходим до конца. В конце — решение.

Поэтому, если вы записали блок-схему в таком виде, её сможет понять человек, даже не говорящий на русском языке.

Но у этой блок-схемы есть большая проблема. У нас жёстко зафиксированы числа, с которыми мы работаем. Если мы возьмём гири других весов или поменяем их местами, наша блок-схема сломается.

Например, у меня первое число будет не 1, а 10. Тогда мне придётся заменить блок **8 > 1?, 3 > 8?** и т. д. В этом случае и ответ, и все действия, которые мне потребуется совершить, тоже станут другими. То есть схема, которую мы сейчас составили, решает только одну конкретную задачу. Это очень неудобно, так как для новой задачи придётся придумывать новую схему.

В программировании это решается следующим образом.

[00.44.10]

Блок-схема 2

Как в старом КВН-номере: «Что же случилось? Были цифры, стали буквы».

1. Мы переименовали все гирьки и сказали, что первая гирька будет называться **a**. Так же, как мы брали морковку и говорили, что она будет отвечать за максимальный вес.
2. Теперь говорим, что первая гирька будет с именем **a**. И каждый раз, когда потребуется найти первую гирьку, просто скажем, что эта гирька с именем **a**.
3. Далее возьмём вторую гирьку и назову её **b**.
4. И так далее.

У нас есть пять букв, и этого достаточно, чтобы переименовать пять гирь.

Далее работаем так же.

1. Говорим, что максимальный — первый элемент, то есть элемент **a**.
2. Далее идём по этой схеме (на слайде).

Маленькое упражнение

1. Поменяйте цифры в первом блоке на другие.
2. Замените их на разные числа.
3. Попробуйте найти в этой схеме максимальное значение.

Я думаю, что трёх минут для этого будет достаточно. Просто убедитесь, что схема, в которой мы заменили конкретные числа на буквы, позволяет решать разные задачи. Всё, что понадобится сделать для адаптации указанной схемы к решению другой задачи — в первом блоке заменить значения. Например, у вас будет:

- **a** равно не 1, а, например, 10;

- **b** равно не 8, а 18.

Придумайте другие значения, подставьте их и убедитесь, что если вы аккуратно, по заданной блок-схеме проходите, в конце всё равно окажется максимальное значение.

Даю вам на это три минуты. Сделайте это упражнение.

Итак, надеюсь, все убеделись, что эта блок-схема работает. Иногда я ошибаюсь в блок-схемах, меняя местами Yes и No, из-за чего схема бывает нерабочей. Поэтому проверяйте за мной. А если вдруг такое находите, смело пишите в комментарии, мы это исправим.

[00.49.30]

Слова и блок-схемы непонятны компьютеру

Мы до сих пор составляли какие-то алгоритмы. Разбирались с тем, как они работают, даже как-то их записывали. Но сейчас все эти алгоритмы до сих пор предназначены только для людей. Мы записывали их на русском языке и оформляли в виде блок-схем, но компьютер не понимает ни слова, ни картинки, ни блок-схемы. Поэтому нам надо каким-то образом донести до него информацию.

И хотя решение уже найдено, и мы записали его в нескольких видах, только здесь возникает программирование. Только здесь возникает язык программирования, на котором надо записать решение.

Если хотите, чтобы компьютер смог выполнить составленный вами алгоритм или блок-схему, надо написать это на понятном для него языке. То же самое с шеф-поваром, который говорит «Бланшируй» своему помощнику. Надо убедиться, что ваш помощник (компьютер) понимает поставленную перед ним задачу.

На слайде есть картинка «Моя твоя не понимает» и выделенное жирным слово «синтаксис». Слово «синтаксис» здесь очень важное. Синтаксисом называется набор правил, которые надо знать и соблюдать, чтобы компьютер понимал, что от него хотят.

[00.51.05]

Примеры решения на языках программирования

В качестве примера далее идёт реальный код, который может работать. Этот код написан на двух языках программирования. С одной стороны — язык Java, а с другой — Python.

Обратите внимание на код. Если на него посмотреть и попытаться вспомнить алгоритм, который мы расписывали, станет понятно, что здесь происходит.

Первые пять строк на Java и первые пять строк на Python отвечают за первый блок блок-схемы 2. Ранее мы определяли, как будут называться те или иные переменные и какие значения они станут принимать. Здесь будем делать то же самое.

В Java вводим переменную с именем **a** и присваиваем ей значение 1. А в Python **a** просто равно 1.

Языки отличаются не только по синтаксису, но и по внутреннему устройству. Если посмотрим на внешние отличия, то увидим, что в Java почему-то требуется указывать, какое значение будет храниться внутри.

Немного зная английский язык, мы поймём, что int — integer — это целое число. Поэтому в Java нам надо подсказать компьютеру, что в переменной **a** будет находиться число, равное 1.

Если мы говорим про Python, то там синтаксис немного другой. В Python можно не указывать тип значения, который будет находиться в переменной. Компьютер сам с этим разберётся.

Однако у вас не должно появиться ощущение, что Python лучше, чем Java. У разных подходов есть свои особенности. И то, что мы не указываем в Python тип, не лень программистов. Просто есть ряд особенностей, которые позволяют использовать то, что мы не указываем. И наоборот, у Java есть ряд преимуществ, которые достигаются именно тем, что этот тип надо указывать. Главное, чтобы вы поняли, что наличие типа в Java и его отсутствие в Python — просто некоторые особенности языков.

Ещё мы видим, что в Java надо ставить «**;**» после каждого выполняемого действия. В Python это не требуется.

Теперь если мы посмотрим на шестые строчки, то увидим, что наша морковка ставится на самую первую переменную (гирьку). В Java есть **int max = a;** — мы говорим, что первая гирька теперь максимальная. То же происходит на языке программирования Python: мы говорим, что **max** — первая гирька **a**.

Далее идут строки 7, 8, 9, 10 — это условия — вопросы: «Что нужно делать?», «Какие гири надо сравнивать?».

Например, **if** переводится как «если». То есть если **b > max** выполняется (условие в ромбике), делаем следующее: **max = b;**. А если это условие не выполняется, идём по ветке No, не будем делать **max = b;**, а перейдём на следующую строчку.

Посмотрите на этот код, сравните его с блок-схемой, рассмотренной выше, и поймёте, что на самом деле всё понятно. Для кого-то это станет первым опытом чтения и понимания программы. Вроде бы написаны какие-то странные вещи, но если знать алгоритм, который там написан, и не знать синтаксиса того или иного языка, вы всё равно догадаетесь, что же там такое.

[00.55.05]

Блок-схема 2

Вернёмся на блок-схему 2 и вспомним, что на ней происходит.

Мы задали значение, определили, что максимальное — первое. Далее последовательно сравниваем одну за другой (гири) с этим максимальным.

Если значение, которое мы сравниваем, оказалось больше максимального, меняем значение, хранящееся в переменной **max**. Говорим, что максимальное теперь это значение. А если нет, то просто переходим к следующей строке. Именно это записано у нас в коде на языке Java и на языке Python.

[00.55.40]

Примеры решения на языках программирования

Снова посмотрите на этот код. Поймите, что на самом деле нет ничего страшного. Да, есть синтаксические отличия. Например, в языке Java условие записывается в скобках, а в Python их можно не указывать, но придётся ставить «**:**» в конце этого условия.

Есть также одиннадцатые строки. Знание английского языка позволит понять, что **print** — это «печать». В нашем случае — печать на экране. В конце этой программы на экране появится максимальное значение.

Но! *Если набрать код на языке Python, он заработает. Однако на языке Java код в таком виде работать не станет. Просто Java немного более сложный язык, он требует дополнительных записей, которые мы сейчас убрали с экрана. На языке Python — полностью рабочая программа. Но смысловая часть языка Java тоже передана полностью.*

[00.56.43]

Задача 2

Лабиринт

Как видите, от гирек мы ушли. Теперь будем искать выход из лабиринта.

Если у кого-то есть робот-пылесос, который ездит по дому или квартире, вы примерно представляете, что сейчас потребуется сделать.

У нас есть вход в лабиринт, откуда мы стартуем. Нам надо найти выход из этого лабиринта и при этом записать некий алгоритм. Этот алгоритм позволит найти выход любому человеку, который умеет просто выполнять несколько действий, совершать какие-то проверки.

Посмотрите на этот лабиринт — не будем делать паузу, а разберёмся с этим в ходе выполнения задачи. Постарайтесь найти выход. Лабиринт не самый сложный, решение можно попытаться найти.

Вы заходите с нижнего правого края, где находится вход. Можно водить пальцем по экрану. Ищите выход — он находится вверху слева. Когда вы видите лабиринт целиком, это сделать проще.

Для простоты начнём с того, что вы зашли в лабиринт и видите всю картинку. Можете пальцем поводить по этому лабиринту, чтобы найти из него выход.

Я думаю, что уже есть ребята, которые нашли выход.

Знаете ли вы какой-нибудь алгоритм или правило, которое позволяет выбраться из любого лабиринта? Если знаете, пишите в комментариях. Мы их обязательно посмотрим.

Есть разные способы, как выбираться из лабиринтов, но самый простой наверняка известен нескольким людям на этой трансляции.

Посмотрим, какое решение может быть.

Есть правило левой руки — когда вы подходите к входу в лабиринт, касаетесь левой рукой стенки и идёте вдоль стенки. Если есть какие-то повороты, то идёте по ним, также касаясь левой рукой стенки.

Наши действия:

1. Зашли в лабиринт. Левая рука касается стенки. Идём дальше.
2. Упёрлись в стенку. Надо повернуть, чтобы левая рука не отрывалась от стены.
3. Идём в правую сторону. Стена пропала, поэтому мы заворачиваем туда, куда идёт наша стена.
4. Проходим вдоль стены.
5. Поворачиваем. Опять идём вдоль стены. Главное, чтобы левая рука не отрывалась от стены нашего лабиринта.
6. Уходим. Оказываемся в тупике. Ничего страшного, алгоритм продолжается.
7. Проходим дальше.
8. Поворачиваем и идём по другой стенке. Левая рука также не отрывается от стенки.
9. Проходим дальше.
10. Снова забредаем куда-то внутрь. Ничего страшного, есть правило и мы его соблюдаем.
11. Идём дальше и выходим и нашего лабиринта.

Решение всегда есть, если лабиринт построен правильно: есть выход и выход, лабиринт связан стенками, нет пустых пространств и нет разрывов. В этих условиях вы всегда найдёте выход из лабиринта. Лабиринт не всегда будет коротким, вернее, он всегда будет не самым коротким, но из него получится выйти.

Теперь для вас будет некоторое задание.

[01.00.54]

Задача 3

Запишите алгоритм для выхода из этого лабиринта.

Человек стартует внизу и делает шаг вперёд. Его действия:

- двигаться вперёд;
- поворачиваться;
- выполнять какие-то условия, например, проверять:
    - есть ли слева от него стенка или нет;
    - есть ли справа от него стенка или нет.

Но ваша задача — записать алгоритм, посредством которого человек сможет выбраться из этого лабиринта. Даю вам на это пять минут, а потом мы посмотрим один из возможных алгоритмов, блок-схем.

Итак, я надеюсь, что все смогли поиграться с этой задачей. Но обратите внимание, что это далеко не простая задача. Если у вас пока не получилось складного алгоритма или блок-схемы, ничего страшного, далеко не у всех это выходит сразу.

[01.07.06]

Блок-схема лабиринта

Посмотрим один из вариантов, который так часто приводится.

1. У нас есть некоторый старт. Нам надо ответить на вопрос: «Вы вышли из лабиринта?». Считаем, что мы только зашли, поэтому пойдём по ветке No.
2. Как только мы выйдем из лабиринта, пойдём по ветке Yes, и наш алгоритм закончится. Можно крикнуть «Ура-ура!».
3. Пока мы находимся внутри лабиринта, будем отвечать на какие-то вопросы. Например: «Левая рука касается стены?».
4. Если не касается, то конкретно эта блок-схема предлагает: «Повернитесь налево».
5. После этого алгоритм ещё раз попросит нас ответить на вопрос: «Левая рука касается стены?».

Когда такие стрелочки возвращаются, зацикливаются, то в программировании это называется циклом.

Вы можете выполнять какое-то количество действий. Вам не надо расставлять блоки так, как мы делали это раньше. Вспомните, у нас было пять сравнений, и мы делали их одно под другим. Теперь представьте, что у нас не пять гирь, а сто. В этом случае наша блок-схема не влезла бы в экран. Но в программировании это решается просто — у нас есть возможность зациклить наш алгоритм. То есть до выполнения условий мы будем это делать. Как только условие выполнится, вылетим по другой ветке — Yes.

6. Далее нас спрашивают: «Можно шагнуть вперёд?».
7. Если впереди стенка, то предлагается повернуть направо.
8. Мы повернули направо.
9. Нас опять спрашивают: «Можно шагнуть вперёд?».
10. Если до сих пор нельзя — опять поворачиваем направо, где нам задаётся предыдущий вопрос.
11. Если на вопрос «Можно шагнуть вперёд?», мы ответили Yes, то делаем шаг вперёд.
12. Снова зацикливаем наш алгоритм и проверяем, вышли мы из лабиринта или нет.
13. Если мы до сих пор в лабиринте, но сделали один дополнительный шаг, то начинаем всё заново.
14. Уточняем, касается ли наша левая рука стенки.
15. Если левая рука не касается стенки, то поворачиваем налево.
16. Если левая рука касается стенки, то уточняем, можно ли шагнуть вперёд.
17. Если шагнуть нельзя, поворачиваем направо.
18. И так далее.

Предполагается, что, пользуясь этим алгоритмом, из указанного лабиринта получится выйти.

[01.12.44]

Задача 4

Это более сложная задача. Далеко не все программисты решают её сразу. Естественно, опытные разработчики проблемы здесь не видят. Однако начинающего программиста, который только выходит на работу, эта задача может привести в ступор.

Предположим, что у нас есть два друга. Друг 1 и друг 2. Они движутся навстречу друг к другу с какой-то скоростью. У друга 1 скорость — 1 м/с, он идёт не спеша. Друг 2 торопится, его скорость — 2 м/с. У них

есть собака, она хорошо знает обоих друзей и периодически бегает от одного к другому. Когда друзья стартовали, собака от друга 1 бежит к другу 2, добегает до него и сразу возвращается к другу 1. У собаки тоже есть какая-то скорость. То есть пока собака будет бежать к другу 2, друг 2 тоже пойдёт ей навстречу. Таким образом, они пересекутся в какой-то точке. Потом собака развернётся и сразу же побежит обратно.

Вопрос. *Перед тем как друзья встретятся, сколько раз собака перебежит от друга 1 к другу 2?*

Здесь специально не указаны никакие значения. Для удобства, например, мы можем взять 1 м/с, 2 м/с и 5 м/с (скорость собаки). Примерное расстояние между друзьями — 10 км (10 тыс. метров).

Но наш алгоритм (решение) должен работать для любых цифр, какие бы мы не взяли. Для удобства можете взять какие-то конкретные цифры, но в целом алгоритм должен работать всегда.

Чтобы немного упростить эту задачу, вспомним, что мы проходили в школе.

[01.14.40]

Формулы из средней школы

Эта задача на скорость, время и расстояние.

Если:

S — расстояние

V — скорость

T — время

То у них есть такое соотношение:

V = S / T

S = V * T

T = S / V

Я надеюсь, что вам не очень больно и страшно от формул, которые были у нас в 6–7 классах. Они позволят проще подойти к решению этой задачи.

Представленная задача, как правило, в школе не решается. Она немного сложная. В школе бы решалась другая задача. Например: «Какое расстояние пробежит собака, бегая между этими ребятами?». И уже эта задача решается проще, о ней мы можем порассуждать отдельно. Но сейчас задача другая.

Друг 1 идёт в правую сторону, а друг 2 спешит к нему навстречу. Собака бегает сильно быстрее этих ребят. Она стартует с другом 1 и начинает бежать навстречу другу 2. Как только собака добегает до друга 2, она

разворачивается. То есть мы считаем, что один раз от друга 1 до друга 2 собака пробежала. Далее она начинает бежать к другу 1, а он продолжает к ней идти. Собака добегает до него и разворачивается. Оба друга подходят всё ближе и ближе, а собака постоянно носится между ними. Надо посчитать, сколько раз этот пёс перебежит от друга 1 к другу 2.

Это последнее задание на сегодня, с которым нам надо разобраться. Сделаем паузу на пять минут. Подумайте над решением этой задачи. Успокою сразу — эта сложная задача, особенно для тех, кто программированием, алгоритмами и решением задач занимается впервые, а школьную программу уже забыл. Но её решение мы разберём.

Чтобы решение задачи показалось более интересным, над ним надо немного подумать. Вернёмся к обсуждению через пять минут и посмотрим, как решить эту задачу.

Ну что же, немного пострадали от задачи, это нормально. Но сначала я бы хотел обратиться к ребятам, которые смотрят трансляцию в записи.

Если вы перемотали видеозапись и не думали над решением задачи, то советую не делать так. Лучше попытаться её решить, даже если она показалась сложной. Так, по крайней мере, станет интереснее думать над решением этой задачи. Если предварительно посмотреть решение и не попытаться составить его самому, оно покажется очень простым. А если сначала немного пострадать и подумать над её решением, станет интереснее, а главное, полезнее.

В этой задаче многое ещё не определено. Гораздо проще выполнять задачи, если есть конкретные числа. Можно сначала что-то решить с конкретными числами, а потом работать с алгоритмом.

У нас не определено расстояние в тот момент, когда ребята встретятся. Это может быть 1 метр, 10 метров и т. д. Нам важно понять, когда наш алгоритм должен остановиться. Однако между людьми не может быть 0 метров. Нам надо выбрать какое-то условие или меру, чтобы считать, что они достаточно близко подошли друг к другу. Это может быть 5 метров или 10 — какое-то конкретное значение. Нам надо понимать некоторое количество значений, заданных изначально. Например, скорость друга 1, друга 2 и собаки, а также изначальное расстояние, с которого они начали идти друг к другу навстречу.

Посмотрим на алгоритм, который позволит решить эту задачу.

[01.24.12]

Алгоритм решения задачи

Появилось много новых дополнительных переменных. Если до сих пор наши переменные назывались однобуквенно, и только у одной переменной было имя **max**, то здесь почему-то сразу стало много букв. Постараемся разобраться в том, что здесь произошло.

Принято, что все языки программирования ориентированы на английский язык. И в этой блок-схеме я тоже пользуюсь английским. У нас есть отдельная статья о том, почему английский важен и как его учить. В ней говорится, что, зная английский, вы сильно упростите себе жизнь, как программиста. Поэтому почитайте её, если ещё не читали.

Посмотрим, что же написано на нашей блок-схеме.

У нас есть первая переменная, которая называется count. Count переводится с английского как «количество». То есть в этой переменной мы будем считать, сколько раз собака перебежала от одного хозяина ко второму и обратно.

Есть переменная distance, что с английского переводится как «расстояние». Мы задаём, с каким же изначально расстоянием между ребятами мы начинаем решать свою задачу.

Есть переменная, в которой сразу написано много слов — **firstFriendSpeed.** Если бы мы поставили пробелы между словами, было бы непонятно, что такое **first, friend** и **speed** — первый, друг и скорость, или скорость первого друга.

Чтобы не добавлять пробелы, во многих языках программирования слова склеиваются, они пишутся без пробела, слитно. Но чтобы понимать, где начинается новое слово и заканчивается предыдущее, каждое новое слово пишется с большой буквы. Такой стиль именования переменных называется **CamelCase**, где Camel — «верблюд». То есть это верблюжий стиль. Почему-то программисты решили, что, когда некоторые буквы в середине названия заглавные, это похоже на горбики верблюда.

Этот стиль в программировании не единственный, но достаточно популярный. Например, в Java все переменные именуются в **CamelCase**.

Так мы зафиксировали значение первого друга.

1. Похожая на предыдущую переменная — переменная **secondFriendSpeed**. Это скорость второго друга, равная 2.
2. Есть ещё переменная **dogSpeed** — скорость нашей собаки.
3. Ещё есть один флаг, о котором мы поговорим отдельно. Это переменная **friend**, которая равна 2.

Мы инициализировали, то есть задали все значения, которые есть. У нас появился набор переменных, с которыми мы можем работать. Если наша задача изменится, то есть изменятся условия, в этом блоке достаточно будет поменять значения.

Например, если ребята окажутся ближе друг к другу, будет 5 км., то решаем новую задачу с новыми вводными. А сейчас будем решать задачу с имеющимися вводными.

1. Сначала проверим, чему равно **distance**. Перед нами стоит вопрос **distance > 10?** Это способ остановить алгоритм, с которым мы будем работать. Как только **distance** будет меньше 10, последует отрицательный ответ — **No**.
2. Скажем, что алгоритм завершился, и собака пробежала **count** раз. Вместо **count** подставим то значение, которое получится в ходе нашего алгоритма. Если вы считаете, что 10 метров — слишком далеко, чтобы встретиться, можете подставить 5 метров или 1 метр.

   Таким образом, есть какое-то условие, в результате которого мы поймём, завершился ли алгоритм.

3. Далее мы должны понять, куда бежит собака: от друга 1 к другу 2 или от друга 2 к другу 1. То есть у задачи есть два состояния: собака бежит слева направо и справа налево. В зависимости от этого задача решается по-разному, вычисление будет немного отличаться.
4. Посмотрим, что произойдёт, если переменная **friend** будет равна 2.

   Изначально мы сказали, что собака побежит к другу 2. Вспоминаем, что если переменная **friend** равна 2, значит, собака побежит от друга 1 к другу 2. А если **friend** будет равно 1, собака побежит от друга 2 к другу 1. Так мы укажем, к какому другу она побежит.

   Допустим **friend** равен 2, значит, на вопрос **friend = 1?** мы отвечаем **No**, так как **friend** равен 2.

5. Далее появляется новая переменная — time. Чтобы рассчитать **time**, расстояние между друзьями делим на сумму скорости друга 2 (secondFriendSpeed) и собаки (dogSpeed). Так как они бегут друг к другу навстречу, скорость их сближения — это сумма их скоростей. То есть расстояние между ними, которое изначально было 10 км, они пробегут, как скорость друга 2 и скорость собаки — **secondFriendSpeed = 2** и **dogSpeed = 5**. Таким образом, двигаться навстречу друг к другу они будут со скоростью 7 м/с.

   Так мы сможем рассчитать, через какое время собака добежит до друга 2.

6. Посчитали и запомнили.
7. Сказали, что дальше собака побежит к другу 1.
8. В это время друг 1 не стоял на месте, а двигался. Нам важно понять, когда собака добежала до друга 2, какое расстояние стало между другом 1 и другом 2.

   Это считается в ячейке, где текущее значение расстояния (distance) будет равно разности между расстоянием (10 км) и суммой скоростей друзей (firstFriendSpeed + secondFriendSpeed), умноженной на время (time), за которое собака пробежала от одного друга ко второму.

   То есть собака добежала до друга 2, друг 1 при этом шёл навстречу. Расстояние между ними стало меньше. Далее мы вычисляем, какое расстояние осталось между ними, когда собака добежала до друга 2.

Здесь уже используется программистский приём. Если я, как математик, буду смотреть на эту строчку — **distance = distance - (firstFriendSpeed + secondFriendSpeed) * time** — то скажу, что здесь что-то не так. **distance** стоит до и после знака «=». Это как в уравнении: **x** находится с обеих сторон от знака «=». Если перенести все **x** в одну сторону, они исчезнут, и получится нуль, равный чему-то.

В программировании знак равенства работает иначе. Опять же, от языка к языку синтаксис может меняться: где-то это один знак «=», где-то он с двоеточием, как в языке Pascal. Но здесь равенство не уравнение, а присваивание. То есть сначала мы вычислим то, что написано справа от знака «=», а когда получим конкретное значение, напишем его слева от знака «=». Посмотрим, как это будет работать.

Изначально **distance = 10 000 м**. Нам надо узнать, чему она будет равна потом. Для этого подставим имеющиеся значения в формулу **distance = distance - (firstFriendSpeed + secondFriendSpeed) * time** и посчитаем, что получилось. Справа от знака «=» у меня будет конкретное число. Получившееся значение запишем в переменную **distance**. Поменяется то, что находится внутри **distance**. То есть вместо 10 000 будет меньшее число, потому что из него мы что-то вычитали, ведь друзья стали ближе друг другу.

9. Далее используем тот же приём, чтобы этот счётчик перекинуть. Для этого берём формулу **count = count + 1**.

   Помним, что count = 0. Сначала вычислим то, что находится справа от знака «=».

   a. Вместо count подставим 0:

      0 + 1 = 1

   b. Полученное значение, то есть единицу, запишем в count. В переменной count станет храниться не 0, а 1. То есть после этой ячейки будет единица. А когда мы зайдём сюда потом, то вместо 1 будет 2 — после каждого посещения аккаунта число станет увеличиваться на единицу.

10. То же самое здесь — **distance = distance - (firstFriendSpeed + secondFriendSpeed) * time**. Каждый раз, когда мы станем сюда заходить, **distance** будет уменьшаться, так как ребята движутся друг к другу, и расстояние между ними сокращается.

    Таким образом, мы пошли по ветке **time = distance/(secondFriendSpeed + dogSpeed)**. Собака добежала до друга 2 и побежит дальше к первому.

    Мы выяснили, какое расстояние осталось между другом 1 и другом 2, и увеличили наш счётчик на единицу.

11. Отправляемся обратно и отвечаем на вопрос **distance > 10?**

    Если расстояние между друзьями больше 10, то есть они ещё не встретились по нашему условию, то спускаемся. Значение **friend** мы поменяли на 1, теперь там хранится единица.

12. Далее переходим вправо, где представлена формула **time = distance/(firstFriendSpeed + dogSpeed)**. Здесь мы складываем не скорость друга 2 и собаки, а скорость друга 1 и собаки, потому что собака теперь бежит навстречу другу 1.
13. Считаем время, когда собака добежит до друга 1, и меняем состояние алгоритма, так как пёс далее побежит к другу 2.
14. Далее идёт то же самое — пока собака бежала к другу 1, расстояние между друзьями сократилось снова. Поменяли значение счётчика.
15. Снова выясняем, насколько близко друзья находятся друг от друга. И когда они достаточно близко подойдут друг к другу, мы узнаем, сколько раз собака перебежала от одного к другому.

Алгоритм может быть немного сложным, особенно для первой лекции. Но не стоит пугаться. Посмотрите на него ещё раз. Здесь есть ряд вещей, возможно, новых для вас, но на них надо обратить внимание.

Например, мы перестали использовать буквы, чтобы именовать переменные, а начали применять конкретные слова. Если знать значение этих слов, то даже человек, который не посвящён в наш алгоритм, сможет понять, что **distance** — это расстояние, **firstFriendSpeed** — скорее всего, скорость первого друга, **secondFriendSpeed** — скорость второго друга, а **time** — время.

Будет гораздо лучше, если вы в своих алгоритмах будете использовать осмысленные конкретные слова, а не просто какие-то отдельные буквы. — **x, y, a, b** — которые понятны только вам и только сейчас. Если назовёте свои переменные **a, b, c**, то через какое-то время, например, через неделю, скорее всего, не сможете вспомнить, что там было. А если значения переменных будут осмысленными, то возвращаться к своим задачам станет проще.

Ещё мы снова обратились к циклам и можем повторять какие-то действия из раза в раз. Если собака бегает от первого друга ко второму и от второго друга к первому, и так несколько раз, то проще этот процесс зациклить. Программа сама поймёт, когда надо остановиться.

У нас есть точка выхода из программы. Например, когда нам надо добавить какое-то условие. Если оно выполняется, выходим из алгоритма. В нашем случае это расстояние между ребятами.

Мы также разобрали очень важный приём, когда знак «=» в программировании не уравнение, а присваивание значения. Сначала вычисляется всё, что находится справа, а потом записывается в переменную слева, даже если справа и слева находится одна и та же переменная.

[01.09.32]

Практическое задание

Можно ли посредством указанного алгоритма, отвечая на вопросы и выполняя действия, которые находятся в зелёных ячейках, выйти из представленного лабиринта?

Предполагается, что мы стартуем в нижней ячейке.

1. Первый вопрос: «Мы вышли из лабиринта?». Так как мы только в него вошли, то, очевидно, что ещё не вышли.
2. Второй вопрос: «Левая рука касается стены?». Так как слева стена есть, значит, наша левая рука её касается.
3. Следующий вопрос: «Можно шагнуть вперёд?». Шагнуть нельзя.
4. Ответ на это: «Повернитесь направо».
5. Поворачиваем направо.
6. Снова отвечаем на вопрос: «Мы вышли из лабиринта?».
7. Если можем шагнуть — шагаем.
8. И так далее.

По алгоритму, который здесь указан, надо ответить на вопрос: «Вы можете выйти из лабиринта?». Сейчас на этом останавливаться не будем. Поиграйте с этой блок-схемой. Попробуйте походить по этому лабиринту, используя указанный алгоритм. Он может быть неправильным, и, возможно, там есть проблемы, которые вам придётся найти и исправить. Но попробуйте с этим поиграть.

Уметь проверять алгоритмы очень важно. То есть написать алгоритм недостаточно, важно его проверить. А проверить его надо на практике: вы берёте алгоритм и начинаете его выполнять. Иногда это называется заменой компьютера.

Например, вы написали для компьютера какую-то программу, записали решение, алгоритм, и теперь хотите понять, что компьютер будет делать. Можете поставить себя на его место и попробовать выполнить все действия самостоятельно. То есть возьмите ручку или карандаш, готовый или собственный алгоритм и попытаетесь по этому алгоритму пройти. Если возникнут какие-то сложности, значит, в алгоритм надо внести изменения.

На семинаре по этой лекции мы к этому ещё вернёмся:

- обсудим, как можно записать решение так, чтобы оно работало;
- выясним, какие сложности есть у представленного решения (на слайде) — вы это узнаете сами.

Ваша задача — разобраться, как выбираться из лабиринтов. Презентация будет приложена к уроку.

[01.12.20]

Блок-схема лабиринта

Решение, которое вы найдёте, должно работать не только для первого лабиринта, рассмотренного нами ранее, но и для других лабиринтов, построенных по той же схеме.

В качестве примера приведу два лабиринта, представленных на слайде. Ваш алгоритм должен работать для каждого из них.

[01.37.17]

Заключение

Посмотрите этот алгоритм уже на свежую голову. Я думаю, что многие утомились, так как лекция получилась длинной и информативной.

Похожие задачи мы будем разбирать на семинарах, поэтому не пропускайте семинары. Лучше смотреть их не в записи, а посещать очно. Так вы сможете задавать свои вопросы преподавателю и получать на них ответы.

На сегодня всё. Спасибо. Пишите о своих эмоциях, впечатлениях и пожеланиях в комментариях. До встречи на следующей лекции.