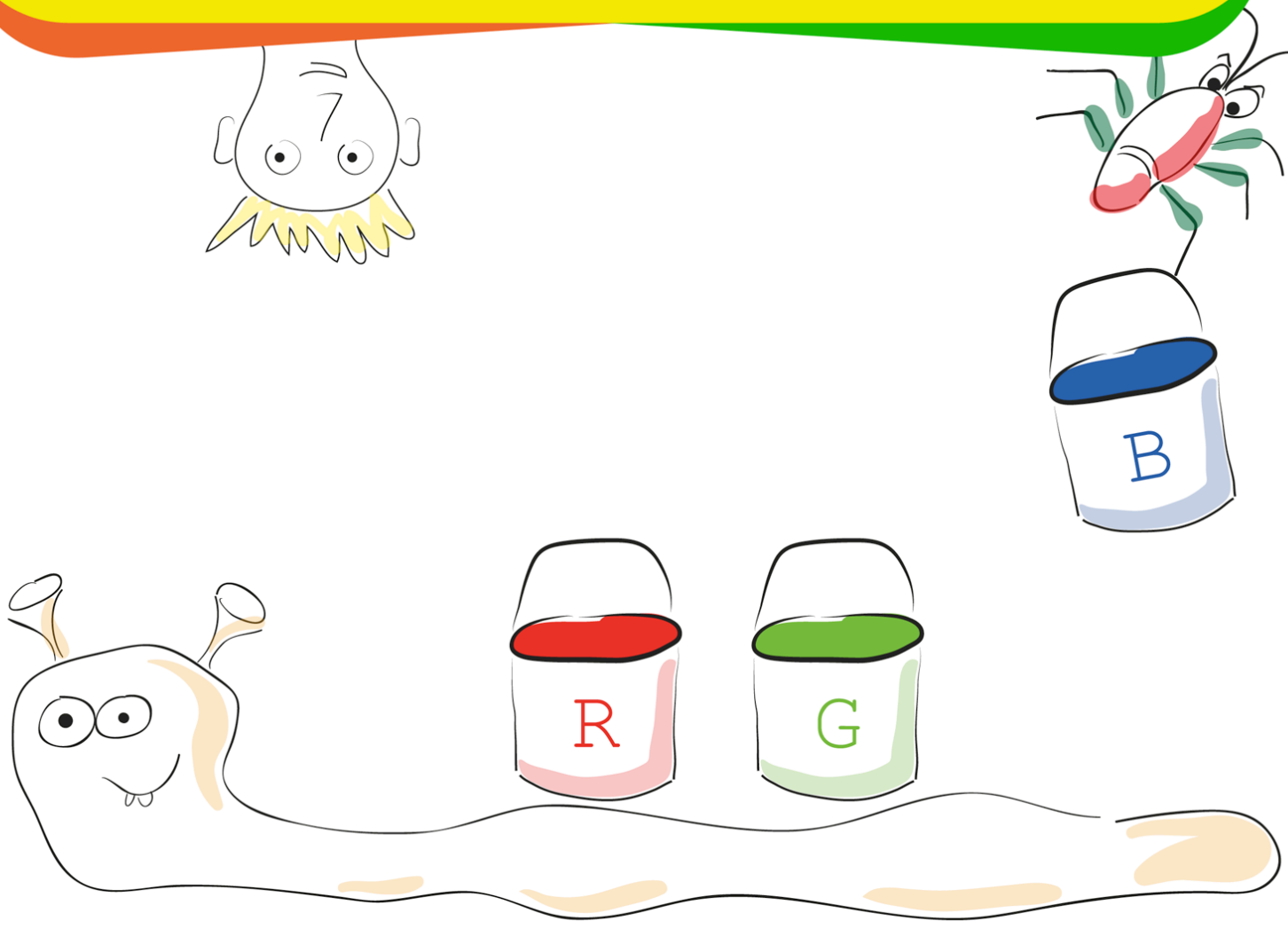
Игорь Грессус

Яша учится программировать



Детская JAVA

```
void setup() {
  size(1000, 700);
}
void draw(){
  fill(random(255),random(255),random(255));
  ellipse(random(width), random(height), 10, 10);
}
```

«Программисты завтрашнего дня – это волшебники будущего. По сравнению со всеми остальными людьми вы словно носители магической силы»

Основатель и президент Valve
Гейб Ньюэлл (Gabe Newell)

«Если вы можете запрограммировать компьютер, то вы сможете достигнуть своей мечты»

Дик Костоло (Dick Costolo)
Генеральный директор Twitter

«Изучение программирования станет огромной ракетой-носителем для вашего будущего, независимо от ваших профессиональных планов. Изучение программирования сделает вас очень крутым!»

Макс Левшин (Max Levchin)
Сооснователь Paypal

«Программисты меняют мир. Они создают новые, удивительные вещи быстрее, чем когда-либо прежде. Любой человек, обладающий воображением, сможет научиться писать код»

Джефф Уилки (Jeff Wilke)
Старший вице-президент Consumer Business, Amazon.com

«Изучение программирования расширяет кругозор, помогает улучшить мышление и сформировать образ мыслей о вещах, я считаю, оно полезно во всех областях»

Билл Гейтс (Bill Gates)
Сооснователь Microsoft

«Я думаю, что каждому человеку стоит научиться программировать, потому что это учит вас как думать»

Стив Джобс (Steve Jobs)
Сооснователь Apple

Источник: www.code.org

Рецензия на книгу Игоря Грессуса «Яша учится программированию»

Данная работа представляет собой научно-популярное пособие для младших и средних школьников. Автор книги задался целью популяризировать профессиональный язык программирования Процессинг, что основан на JAVA.

Использование различных жанров и стилей речи, например, сказки, занимательных историй, погружающих в мир «монстриков» и возвращающих в обычную действительность, позволяет юным читателям с интересом следить не только за сюжетом книги, но и познавать сложнейшие величины, понятия, наконец, исходные положения программирования.

Незатейливая история обычного школьника Яши обрастает чудесами и невероятными былями-небылицами, которые и прокладывают путь к познанию сложного языка программирования.

Автору удалось передать серьезное, сложное содержание научных понятий благодаря различным способам, формирующим аналитическое, абстрактное мышление школьников. Обилие примеров, иллюстраций, включение системы ассоциаций, сравнений, синонимических форм из других языков, фразеологизмов – все это позволяет юным учащимся легче воспринимать и осваивать как системно-структурную организацию языка программирования, так и овладевать стратегией его использования.

Книга Игоря Грессуса представляет собою учебно-методическое пособие для младших школьников, стоящих на пороге изучения основ программирования .

Л. А. Фейман,
кандидат педагогических наук, доцент.

Обращение к начинающему программисту

Купив эту книгу по программированию на языке Процессинг, ты стал обладателем намного большего, чем просто уроков для начинающих. Ведь ты получил возможность делать открытия с помощью этой книги. Существует огромное множество различных миров, которые ты сможешь сам создать и исследовать! Начиная от стратегических игр, придуманных тобою самим, заканчивая эпическими сражениями в межгалактическом пространстве! Теперь в твоем распоряжении огромное пространство для воображения!

Эта серия книг растет вместе с тобой. От книги для самых начинающих, до книги с профессиональным подходом к программированию. От рисования кругов до работы с трехмерными объектами. От создания переменных к воплощению почти настоящего физического мира в твоей программе.

Используй эту серию, чтобы узнать больше о замечательных возможностях, которые язык Процессинг, основанный на профессиональном языке Java, может тебе предоставить. Читай и путешествуй вместе с ее героями, которые помогут тебе черпать вдохновение для твоего собственного путешествия в увлекательном мире программирования!

И, конечно же, заходи к нам в гости: www.programmingforkids.ru

Отзывы об этой книге

«I enjoy the narrative approach and the enthusiastic drawings»

«В этой книге мне очень понравился подход повествования и воодушевляющие рисунки».

Создатель языка Процессинг

Кэйси РИЗ (Casey REAS)

Обращение к родителям

Дорогие мамы и папы!

Ребенок хочет программировать? Что же делать, как научить?

Всего-то нужно заменить изучение скучных алгоритмов на создание собственных игр! Добавить красок и движения! Нарисовать кучу смешных и глупых картинок! Развернуть действие в сказочной стране! Перевести все незнакомые слова с английского на русский, помочь с их произношением. Убрать занудные домашние задания, и в то же время показать создание программ так, как это будет у самого ребенка на практике, со всеми возникающими на пути ошибками, опечатками и поиском решений! Экспериментировать, баловаться и дурачиться!

Какой бы язык для этого подошел?

- Простой в изучении, чтобы начать можно было с самого элементарного.
- Гибкий, что означало бы возможность реализовать свои идеи разными способами.
- С простейшим интерфейсом, в котором невозможно запутаться.
- Мультиплатформенным, чтобы на любом компьютере его можно было запустить.
- Бесплатным, ведь семейный бюджет не резиновый.
- Но в то же время, он должен являться разновидностью профессионального языка программирования, так, чтобы в будущем ребенку было максимально легко освоится в профессиональной среде.

И именно по такому великолепному языку, который называется **Processing**, полностью совместимому с профессиональным **Java**, мы написали свою книжку.

Эта книга — первый шаг в нашем большом проекте по обучению школьников полноценному профессиональному программированию в веселой и увлекательной форме.

Как вы можете помочь проекту

Рассказать друзьям

Расскажите об этом проекте друзьям, знакомым, тем, кому по вашему мнению он может оказаться интересным!

Мы будем рады любой помощи по популяризации этого проекта — пост в соцсетях, жж и др.

Ваши достижения важны для нас

Расскажите нам о ваших успехах, первых шагах. Отправьте нам ваш отзыв о книге, и мы сможем разместить его в следующих изданиях, на сайте, бумажной версии книги.

Финансовая помощь

Написание книг и поддержка сайта требует много ресурсов. Ваша финансовая помощь, которую вы оказываете, покупая эту книгу, помогает этому проекту развиваться дальше!

Напомним, что реквизиты можно узнать на сайте
www.programmingforkids.ru

Идеи и другая помощь

Если у вас есть идеи по развитию проекта, наполнению и продвижению сайта, или какие-нибудь другие мысли, пишите: programmingforkids@yandex.ru

И заходите на сайт: www.programmingforkids.ru

Краткая история создания языка Процессинг

Язык Процессинг синтаксически базируется на языке **Java**.

Он был создан в 2001 году Кэйси Ризом и Бэн Фраем в Массачусетском Технологическом Институте (MIT) и с тех пор активно развивается некомерческой инициативной группой (Processing Foundation, processing.org).

С самого начала язык Процессинг разрабатывался в целях обучения так, чтобы он мог быть первым языком программирования у начинающих программистов. Его создатели черпали вдохновение в младших языках, подобных БЕЙСИКУ и ЛОГО, но постарались расширить именно визуальные возможности языка.

Простой синтаксис и богатые возможности по созданию насыщенных графических и интерактивных программ обеспечили Процессингу огромную популярность в школах, колледжах и университетах. Более того Процессинг стали использовать архитекторы, дизайнеры, артисты для создания своих работ. Компании, такие как New York Times, General Electric, Nokia, Yahoo! стали использовать Процессинг для визуализации своих внутренних данных.

На основании языка Процессинг был создан язык Wiring для широко известной платформы Arduino, получившей огромную популярность в любительской робототехнике.

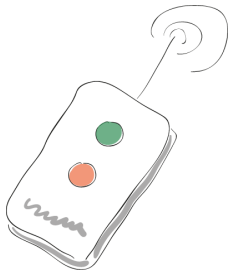
Язык Процессинг продолжает развиваться и на момент написания этой книги выпущен в версии 2.0.3 (5 September 2013).

Оглавление

Введение. Путешествие

Как Яша в другое созвездие улетел

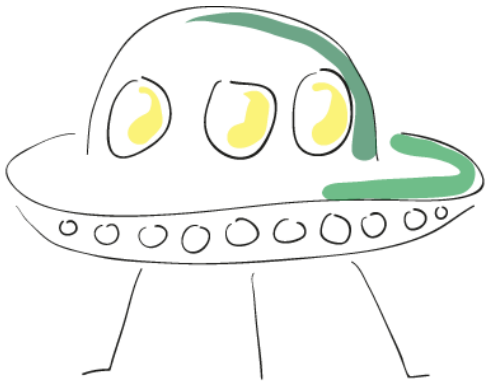
Дело было вечером, делать было нечего, день рождения у Яши уже закончился, гости разошлись, вот и сидел он, перебирая подарки.



Этот от Кости, этот от Миши, а вот кто принес необычный серебристый пульт с двумя кнопочками, Яша так и не вспомнил. Только он потянулся нажать на одну из них, как его зачем-то позвали на кухню, и про пульт он как-то забыл.

А вспомнил он о нем уже на даче, где его оставили одного с бабушкой. Бабуля куда-то ушла, а вот пульт-то как раз вдруг нашелся, и зеленая кнопочка так соблазнительно светилась. Ну не удержаться тут. Нажал Яша на эту кнопочку, и тут вдруг звук какой-то за окном: «Жжжж», а потом «Бжжмяк». Выглянул Яша в окно — ничего не видать, пришлось выйти на крыльцо. А там — космический корабль стоит, невысокий такой, зелененький, еще и мигает фонариками по-кругу. У Яши как челюсть отвалилась, так и стукнула по крыльцу с таким же звуком: «бжжмяк».

Постоял Яша. Подумал. Почесал голову. И еще раз нажал на зеленую кнопку — «бжик», дверь корабля и закрылась. Еще раз нажал — «бжик» и снова открылась. Подошел он поближе — а над ней надпись светится: «Космический корабль — Яше!». А внизу подпись: «от дружественного созвездия Большая Гигабайтица». А под ней еще одна, совсем мелкая: «это та, что рядом с Маленькой Гигабайтицей». «Вот те на, - подумал Яша. - Ну раз это мой корабль, наверно, я в него и зайти могу». Зашел, а там кресло удобное, лампочки какие-то мигают, на мониторах звезды показывают. Посмотрел Яша на пульт, да и нажал снова на зеленую кнопку. Тут-то двери закрылись, и корабль бесшумно понесся в небо.



корабль из созвездия Большая Гигабайтица

«Ой...» — пронеслось в голове у Яши, — «как же так — меня дома хватятся»! Но корабль словно прочитал его мысли, и на экране высветилась фраза: «Время на корабле замедляется и ползет со скоростью этих ваших маленьких скользких созданий в домиках в виде юлы, которые, кстати, прошлый раз чуть не съели наш продовольственный запас на год. А, так о чем это мы, так вот, за целую неделю путешествия на Земле

пройдет не больше минуты. Мы приглашаем тебя посетить наше созвездие Большой Гигабайтицы, в качестве подарка на день рождения. Но если ты откажешься — вернем тебя домой, только, пожалуйста, в этом случае не смотри больше ночью на наше созвездие такими жалостливыми глазищами. Нажми на панели приборов зеленую кнопку, если хочешь полететь, или красную — если хочешь вернуться».

Яша уже занес палец над красной кнопкой, чтобы вернуться, однако все дело испортила вылезшая исподтишка мысль, что он всегда мечтал побывать в космосе, посмотреть, как живут на других планетах. А сейчас еще и время почти остановится, и никто даже волноваться не будет. «Эх, была не была, уже десять лет живу, а вот полет в космос первый раз подарили» — решил Яша и нажал зеленую кнопку.

Как Яша с Интиком познакомился

Как прошел полет, Яша толком даже не разглядел, так быстро все происходило. Сначала корабль чуть задрожал, затем в иллюминаторах вместо домов и деревьев показалось небо, а когда Яша попытался посмотреть вниз, то увидел, что сами здания, огороды и сады стали маленькими-маленькими. А затем и вообще исчезли из виду. Вокруг возник черный космос, яркими точками вдалеке были видны скопления миллиардов звезд, пока и они вдруг не слились в сплошные линии. Затем звезды из линий снова превратились в точки, и одна из них росла, как на дрожжах, пока не превратилась в огромную разноцветную планету.

Приземлившись, или может правильнее было бы сказать, припланетившись на нее, Яша открыл дверь и выглянул наружу.

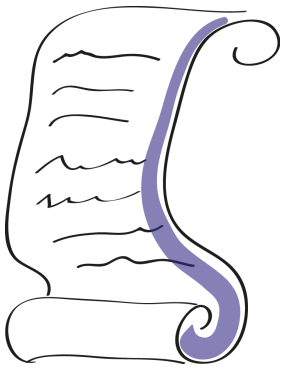
Снаружи стоял улыбающийся внеземной монстрик, который сразу подошел к Яше и сказал: «Привет! Я — Интик! Я тебя в свой телескоп видел, вот и решил пригласить в гости — заходи, располагайся».

Сказать, что Яша удивился, значит, ничего не сказать. Но отчего-то ему здесь понравилось. Вокруг были непонятной формы большие здания, кругом росли растения, чем-то отдаленно напоминающие земные, воздух был свежий, веяло спокойствием и тишиной.

Интик сразу же предложил провести экскурсию по планете. На что Яша немедленно согласился.

Экскурсия по планете

— Мы тут немножко колдуем, то есть Планеты создаем, — сходу заявил Интик.



Для этого у нас есть большой волшебный приемник Заявлений. Мы готовим заявку, еще ее называют Программой создания Планеты, а затем передаем ее Создателю Планет.

Создатель Планет принимает нашу заявку, а затем, в соответствии с инструкциями в ней начинает создавать планету.

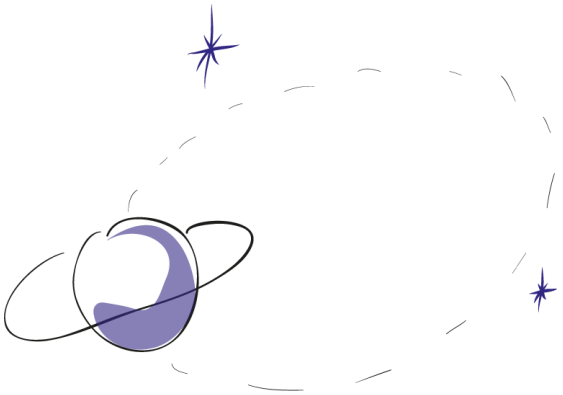
— А на каком языке заявку вы пишете? — поинтересовался Яша.

— Ну раз заявка у нас — это программа создания планеты, то и пишем мы ее, значит, на языке программирования, их кстати много, языков этих, в нашем созвездии. А там где мы сейчас находимся, принято писать на языке программирования Процессинг, — ответил Интик.

— Ага, понятно, — только и сказал Яша.

— А заявка наша состоит из набора заклинаний — команд. Сделать то-то, построить то-то, запустить то-то.

Причем все команды идут строго последовательно, друг за дружкой, как будто каждая следующая держит предыдущую за хвост! В итоге — запутаться невозможно.

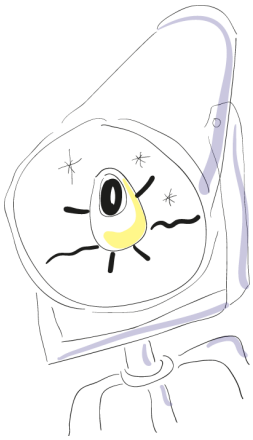


— А можно ли посмотреть на какую-нибудь планету? — спросил Яша, — как она вообще выглядит?

— Конечно можно, — ответил Интик. — Вот телескоп, а для удобства выход с этого телескопа на наш большой экран. Мы смотрим на него и сразу понимаем, что на планете происходит.

— Так просто?

— Ага, но понимаешь ли, мы не можем видеть самих обитателей планеты, или их дома, замки, поля, огороды и оценки в школьных дневниках. Так как они все невидимые!



— Невидимые?! — удивился Яша.

— Но мы можем увидеть то, что обитатели планеты нам покажут на экране. А покажут они то, что мы попросим показать нам в программе!

— А можно как-то пообщаться с ними? — заинтересованно спросил Яша.

— Как же ты с ними пообщаешься, если они невидимые? — удивился Интик. — Но у нас вот есть клавиатура, и они знают, на какие клавиши мы нажимаем. А еще есть мышка, и ее положение на нашем экране — им тоже прекрасно известно!

— Вроде понятно, но не до конца! — огорчился Яша.

— Ничего, я тебе дальше все подробней расскажу, — успокоил его Интик.

Часть 1. Устройство мира

Глава 1. Живность

Монстрики-переменные

— Когда Планета создается, часто первыми на ней поселяются добрые монстрики, — начал рассказывать Интик, — слушай дальше.

— И что, они и правда добрые? — спросил Яша.

— Добрые-то они добрые, но ветреные и переменчивые, просто жуть.

Так вот, один тип монстриков может запоминать числа. Мы таких называем — “переменные”. Переменные — это потому что они могут забывать старые числа и запоминать новые. Ну, менять старое на новое, шило на мыло.

шило на мыле

При этом, каждый монстрик имеет свое имя, а память у него хоть и не дырявая, но короткая, и помнит он только одно число.

Другие монстрики, кстати, могут запоминать слова или фразы.

Например, у нас есть монстрик по кличке ЛИМОНАД, который умеет запоминать одно число.

А еще есть монстрик по кличке БАЛБЕС, который умеет запоминать одну фразу.

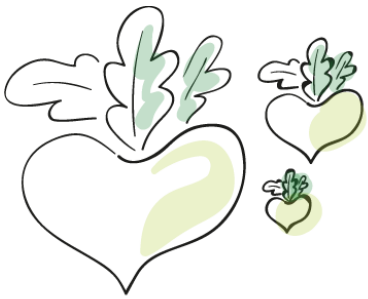
Кстати, те монстрики, которые числа помнят - они из семейства числовых монстриков.

А те, кто фразы зазубривает - те из семейства словесных монстриков.

На языке программирования создать этих монстриков легче легкого, светлее светлого, и вообще, проще пареной редиски.

— Интик, а у нас говорят: проще пареной репы.

— Пусть будет проще пареной репы, так вот, для этого нужно записать магическую фразу из двух слов и одного знака. Первым словом будет название семейства, которому монстрик будет принадлежать. В смысле, то, что он запоминать будет. А вторым словом будет имя вновь создаваемого монстрика. А завершаем заклинание, то есть строчку, точкой с запятой.


семейство пареных реп

Вот еще раз: сначала мы пишем то, на запоминании чего монстрик набил руки, а точней мозги (например «число»), затем его имя (например «ЛИМОНАД»), и в конце каждой строчки ставим точку с запятой.

Смотри, как это выглядит:

```
число ЛИМОНАД;
фраза БАЛБЕС;
```

Наверное, когда-то раньше заклинание было более длинным, а потом его сократили.

А быть оно могло, например, таким:

```
число (— вот что пусть помнит новый монстрик) ЛИМОНАД;
фраза (— вот что пусть помнит новый монстрик) БАЛБЕС;
```

— А постепенно или сразу, но то, что в скобках пропало, растворилось, мыши сгрызли...

— Гномы спрятали, драконы сожгли, бабы-йожки сварили, упитанные мальчики съели... — с готовностью помог собеседнику Яша.

— Ну, да, и теперь осталась только краткая запись.

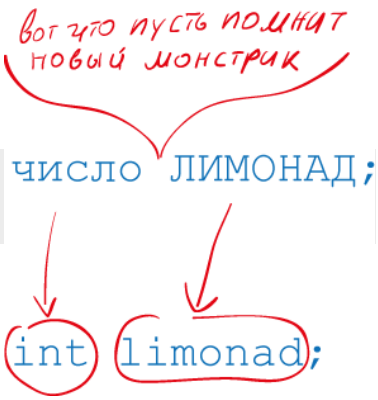
Тайная запись

Интик продолжал рассказывать дальше.

А теперь давай узнаем тайный язык записи заклинаний. Все заклинания в нашем мире записаны в виде шифра, но мы, если пораскинем мозгами (хотя нет, погоди, они нам еще пригодятся), сможем их расшифровать.

Так, для большей тайности, все записывается буквами английского языка, а слова часто сокращены, или вообще означают в буквальном переводе не то, или совсем не то, что имеется в виду на самом деле.

Например, заклинание на создание монстриков ЛИМОНАД и БАЛБЕС было бы записано так:




```
int limonad;
String balbes;
```

Ух, вот зашифровано, да? Что мы, проявив недюжую смекалистость, здесь можем разобрать?

1. «int» — произносится как «инт», это зашифрованное слово «число»;
2. «String» — произносится как «стрин», это зашифрованное слово «фраза»;
3. имена монстриков пишутся английскими буквами и начинаются с маленькой буквы.

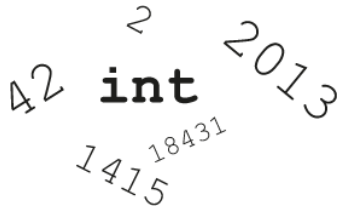
Ну вроде и все. Не так уж и сложно.

— Интик, а почему int пишется с маленькой буквы, а String — с большой?

— А семейство словесных монстриков о себе много возомнило! Думают, раз могут спокойно разговаривать, и обладают некоторыми особенностями, о которых мы пока умолчим, то значит, их и называть нужно с большой буквы!

— А если из вредности их назвать с маленькой буквы? Вот так: «string balbes;»? — прищурившись спросил Яша.

— Тогда они не откликнутся! И будет ошибка в программе, напишут тебе личное письмо, а может и бандероль пришлют, что, мол, нетути такого семейства «string»!



— Может они тогда в этом письме намекнут, что не «string», а «String» нужно писать? — спросил Яша.

— Не, даже это не напишут, - ответил Интик, - просто скажут, что ничего не знают, нет такого семейства, или вообще какую-нибудь ерунду настрочат.

— А мы вместо int и String, то есть вместо число и Фраза можем что угодно написать? — поинтересовался Яша.

— Нет, это мы названия монстриков можем какие угодно придумывать, а вот то, что они помнят, у нас строго регламентировано, и ни шага в сторону! — пояснил Интик.

— То есть int и String — и это все? — огорчился Яша.

— Есть еще несколько семейств, которым монстрики-переменные могут принадлежать, но мы к этому вернемся позже! — успокоил его Интик.

Как поссорились семейство чисел и семейство слов.
Выдуманный диалог БАЛБЕСА и ЛИМОНАДА:
— Привет! — сказал БАЛБЕС.
— 28346! — ответил ЛИМОНАД.
— Как тебя зовут? — спросил БАЛБЕС.
— 3448! 2 32 2? — отреагировал ЛИМОНАД.
— Ну тебя, говори по-человечески! — возмутился БАЛБЕС.
— 23 33 4888 552! — что-то пробормотал обиженный ЛИМОНАД.
С тех пор они больше и не общаются. А если вдруг и пробуют поговорить — то это у них недолго получается!

— Ух ты, давай еще насоздаем монстриков! — попросил Яша.

— Давай, теперь ты! — согласился Интик.

— Хорошо, фразы у нас будут помнить монстрики СЛОВОПОМ и СЛОВОПУС, а числа — МАТЕПУС и МАТЕПОМ:

```
String slovopom;
int slovopom;
int matepus
int Matepom;
String slovopus;
```

— Ну как получилось? — спросил Яша.

— На первый раз отлично! Но есть малюсенькие ошибочки, совсем незначительные, я бы даже сказал, совсем ничтожные, но из-за них твои заклинания или не сработают, или сработают не так, как ты этого хочешь.

— Ой, что-то я не замечаю ошибок, помоги, пожалуйста, растолкуй, а?

— Хорошо, смотри: я просле каждой строчки в твоем заклинании поставлю две наклонные палочки «//» — это будет означать мой комментарий к каждой строчке, на само заклинание он влиять не будет, зато ты сможешь увидеть мои комментарии.
А если вдруг мой комментарий в одну строчку не влезет, то я начну его с условного знака «/» а закончу знаком «*/» — тогда я смогу написать сколь угодно длинный комментарий, хоть на тыщу строк, или даже на миллион!*

```
String slovopom;    // отлично, здесь все правильно

int slovopom; /* у тебя уже создан монстрик slovopom строчкой
выше! и нельзя создать еще одного с таким же именем, или если
он раньше помнил фразы, то он уже никогда не сможет
специализироваться на числах! */

int matepus    /*здесь точка с запятой в конце пропущена — вот
на ней наш главный волшебник точно споткнется и лоб расшибет!
Давай его спасем от этой участи и вернем точку с запятой на
место! */

int Matepom; /* с большой буквы у нас не принято давать имена —
обидятся! */

String slovopus; // отлично, здесь все правильно
```

— Тогда я поправлю сейчас, — сказал Яша, и поправил:

```
String slovopom;
int matepus;        // я поставил здесь точку с запятой
int matepom;        // написал имя с маленькой буквы
String slovopus;    /* здесь я ничего не правил, но мне тоже
захотелось написать комментарий из нескольких строчек!*/
```

— Вот, теперь отличное заклинаньице вышло! — похвалил Интик.

Монстрики с длинными именами

— Интик, а что если мне хочется назвать монстрика не одним словом, а целым предложением, можно ли так сделать?

— Конечно можно, Яша, но нужно помнить некоторые правила, в этом случае:

- Все слова пишутся слитно;
- Имя монстрика все равно начинается с маленькой буквы;
- Каждое следующее слово в его имени пишется с большой буквы.

— Ой, как это?

— А вот смотри, например, мы хотим назвать монстрика «Год моего рождения». Тогда его имя будет выглядеть так: годМоегоРождения, — пояснил Интик.

— И помнить он будет число — год рождения! — догадался Яша. — Тогда в виде заклинания его создание будет выглядеть так:

```
int godMoegoRojdeniya;
```

— Яша, ты совершенно прав! — восхитился Интик.

Транслитерация

РЕЖИМ ЧТЕНИЯ:
повышенная сложность

— Интик, а вот что делать, если я английский язык, как бы это сказать, ну, не в зуб ногой, — спросил Яша.

— А вот если в зуб ногой, то ты фы шефеляфил, и вряд ли это улучшило твое знание английских слов! — воскликнул Интик.

— У нас так говорят, когда, кто-то что-то плохо знает, — пояснил Яша. — Вот сейчас этим «кто-то» выступаю я, а этим «что-то» — английский язык. Как мне придумывать имена, чтобы все время в словарик не лезть?

— В словарике смотреть — дело хорошее. Но можно обойтись тем, что называется крайне заумным словом — транслитерация. А на деле — все просто.

Придумываешь название переменной на своем родном языке, а потом просто заменяешь буквы на английские. Получается сперва не особо понятно, но сойдет на первое бремя.

— Бремя-то, конечно, еще то, но у нас, все же, говорят: «на первое время», — поправил Интика Яша. — Тогда я попробую сейчас по-трас-ли-ве-тировать.

— Транслитерировать!

— Угу, ну так вот:

Меняю ваши русские
на мои английские

— Первого пусть зовут «кащейБессмертный». Тогда его имя будет: ka... А как букву «щ»-то отобразить английскими буквами? Ничего не пойму, — вздохнул Яша.

— Ну, «щ» обычно заменяют на «sch», или на «sh», если так проще.

— Да уж, проще, ничего не скажешь, и правда, шепелявение какое-то, вместо «кащей» получается какой-то «касхеи». Если бы он услышал, как я его переименовал, то мне бы пришлось отложить программирование и шустренько, мигом мчаться искать иголку в яйце!

Ну да ладно, «kasheiBessmertn...». — Яша остановился. — А «ы» как написать? А «й» я правильно, что на «i» заменил?

— Правильно! А главное, чтобы тебе самому понятно было! Букву же «ы» обычно заменяют на английский игрек, то есть «y». А «ый» на «yi».

— Продолжаю дальше. Получается: «kasheiBessmertnyi». Ну ничего так, привыкнуть можно.

— Отлично! Очень даже хорошо! А сможешь транслитерировать... — Интик ненадолго задумался, — ну, скажем, имя «южныйШепелявыйЦарьВсехЧервяков»?

— Ну ты и задачки задаешь, Интик, сейчас попробую, — удивился Яша. — Итак, «ю»... Эм, я, похоже, споткнулся уже на первой букве.

— Тогда вот подсказка:
«Ю» — «yu».
«Ж» — «zh».
«Ш» — «sh».
«Я» — «ya».
«Ц» — «ts».
«ь» — просто пропускается.
«Х» — «h».
«Ч» — «ch».

— Ну с подсказкой это просто, «южныйШепелявыйЦарьВсехЧервяков» у нас становится «`yuzhnyiShepelyavyiTsarVsehChervyakov`». Вот это да, вот это высказался, — поразился сам себе Яша.

— Поэтому все же проще чуть более короткие все же имена придумывать, — согласился Интик.

Какие можно давать имена

— Интик, а имена можно придумывать, какие в голову взбредут? — спросил Яша.

— Да, совершенно любые, но, есть исключения, — ответил Интик.

— Какие исключения?

— В имени переменной нельзя использовать пробелы! Например, «`moya peremennaya`».

— А, теперь понятно, зачем мы второе слово с большой буквы пишем, чтобы прочитать легче было!

— Ага, также нельзя называть переменные начиная с числа, вот так: «`4peremennaya`».

— Понятно, если нужно число будет добавить, я его тогда в конце пришишу, вот так: «`peremennaya4`»!

— Верно, а еще каждый раз, когда создается планета, то есть начинает выполняться твоя программа, запускается невидимый код создания помощников.

— Хм, это какие еще такие помощники?

— Например, создается целый табун монстриков-переменных, мы их еще называем аборигенами потому, что живут сами по-себе, их имена я тебе потом скажу. А так как они уже созданы, то тебе нельзя их повторять!

— А как же я смогу их не повторять, если я даже не знаю все имена монстриков, которые создаются за меня? — удивился Яша.

— Да все имена и я не знаю наизусть, — задумался Интик, — но тут такое правило, старайся придумывать нестандартные имена, например, начинай их всегда со слов `moi`, то есть «мой», или с какого-либо другого слова.

— Ну это я понял, попробую, — озадачился Яша.

— Еще можешь использовать имена, состоящие из одной буквы! Такие монстрики точно не создаются автоматически.

— О, это несколько проще! — обрадовался Яша.

— Только не переусердствуй, а то однобуквенные имена потом трудно понять, как только заклинание, или раз мы пишем на языке программирования, то и называем это программой, в общем, когда все начинает вырастать в размерах.

Вот назвал монстрика «а», чтобы он помнил количество звезд на небе. Потом неделю отдыхал, вправлял вывих на пятой руке, или третье ухо забарахлило — чистил, так вот, вернулся, и думаешь, а что это за монстрик такой по-имени «а», и почему он такие большие числа помнит.

Ну и в любом случае, если твоя программа не запускается и постоянно выдает ошибку, попробуй изменить имена своих переменных! А полный список имен письменно я тебе потом дам, — пообещал Интик.

Учим монстриков запоминать числа и слова

Интик продолжал рассказывать дальше:

— Ура! Эти магические заклинания создают нам двух монстриков, и дальше мы уже можем с ними общаться, например, попросить запомнить что-то.
Пусть монстрик ЛИМОНАД запомнит число 5, а монстрик БАЛБЕС — слово “каша”. Для этого мы в совсем расшифрованном виде напишем:

```
ЛИМОНАД пусть запомнит 5;
БАЛБЕС пусть запомнит ”каша”;
```

Но фразу «пусть запомнит» слишком долго писать, поэтому ее просто заменяют на знак «присвоить», который пишется так: «=».

```
ЛИМОНАД=5;
БАЛБЕС=”каша”;
```

А произносят иногда для краткости так:

```
ЛИМОНАД пусть будет равен 5;
БАЛБЕС пусть будет равен ”каша”;
```

— Немного смешно звучит, — улыбнулся Яша.

Интик продолжал:

— Обрати внимание, что мы уже не пишем `int` или `String` перед именем монстрика, так как название семейства нужно писать только тогда, когда мы создаем монстрика. А раз он у нас уже создан, то нужно просто обращаться к нему по имени.

В закодированном виде:

```
limonad = 5;
balbes = ”каша”;
```

— А слова монстрики могут запоминать даже на русском! Это здорово! — порадовался Яша.

— Заметь, слово “каша” у нас в кавычках, не потому что она подгорелая, просто в нашем мире принято любую фразу заключать в кавычки. Именно для того, чтобы отличать числа от слов.

— Интик, а разве и так числа от слов не отличаются? Всегда же видно, где фраза, а где число, — удивился Яша.

— А у нас, если число в кавычках, то это уже не число, а фраза! — попытался объяснить Интик.

— Что-то запутанно очень, — задумался Яша.

— Ладно, давай мы к этому попозже вернемся. А сейчас про сложение поговорим, — предложил Интик.

Когда мы уже создали монстриков — мы можем попросить их выполнить разные действия с теми значениями, которые они запомнили.

Например попросим прибавить число 5 к тому, что они уже помнят. Думаю, мы можем сразу посмотреть заклинание (код):

```
  limonad = limonad + 5;
```

Обрати внимание, здесь мы тоже уже напрямую просим монстрика что-то сделать, а это означает, что где-то вверху, он должен быть обязательно создан. А иначе как мы можем к нему обращаться, если он не существует?

То есть полностью наш код будет выглядеть следующим образом.

— Код? — удивился Яша, - может быть кот? Или котенок?

— Нет, именно «код». Мы так для краткости вместо слова программа говорим, — пояснил Интик.

```
  int limonad;     // создать монстрика limonad
  limonad = 5;      // limonad  пусть запомнит число 5
  limonad = limonad + 5;  /* монстрик limonad пусть
запомнит: сумму того, что уже помнит limonad и числа
пять*/
```

— Так он же и есть монстрик `limonad`! — перебил Интика Яша.

— Ну да, то есть пусть он узнает у самого себя число, которое он помнит и прибавит к нему число 5.

Как ты уже догадался, он просто увеличит свое значение на 5.

В таких случаях, когда монстрик должен к тому числу, которое он уже помнит прибавить еще одно число, настоящие профессионалы, магистры магии любят использовать укороченную запись:

```
limonad += 5;
```

— Ага, запомнил, — повторил Яша, — «`limonad =+ 10;`».

— Да нет! — поправил его Интик, — не «`=+`», а «+=»! Ты знаки местами перепутал!

— Ладненько, сначала «плюс», затем «равно», вот еще раз: «+=» — я запомнил. И еще, подскажи мне, что это за наклонные линии такие “//”?

А, я сам вспомнил, это комментарии за кадром, они ни на что не влияют, но иногда полезны, что-нибудь прояснить по ходу дела.

— Да, все эти комментарии Создатель Планет благополучно игнорирует! Перед тем как выполнить инструкцию, он просто стирает для себя все, что находится после знаков «//» и до конца строки. А так же все между знаком «/*» и знаком «*/», даже если это будет много строк.

— Кстати, надеюсь ты помнишь, что если мы уже создали монстрика, запоминающего числа, то мы не сможем уже никогда его переучить запоминать слова. А так же, что монстрики такие выпендрежники и в жизть не переносят, чтобы у них имена повторялись. Поэтому не должно быть двух разных монстриков с одинаковым именем, иначе они такой крик поднимут! Хорошо еще, что Создатель Планет перед запуском программы проверяет все имена, и обязательно предупредит, если вдруг что.

Сложение слов между собой

— А вот если мы попросим монстрика balbes из семейства словесных прибавить к фразе «привет» фразу «Яша», то что получится? — спросил Интик.

— А что фразы тоже можно друг к другу прибавлять? — удивился Яша.

— Ну да, только они складываются по другому, не ма-те-ма-ти-че-ски, — по слогам произнес Интик, — а приклеиваются друг к другу, как пластилин.

— Ой, покажи, как это? — попросил Яша.

— Вот смотри, — произнес Интик.

```
balbes = "привет" + "Яша";
```

— В результате balbes будет помнить фразу «приветЯша».

— Как-то оно странно выглядит, слова словно слиплись — заметил Яша.

— Ага, а для того, чтобы выглядели раздельно, нужно добавить знак пробел. А еще давай добавим запятую, после слова привет.

— Вот так?

```
balbes = "привет, " + "Яша";
```

— Да, в результате balbes будет помнить фразу «привет, Яша».

Сложение чисел и слов

— Интик, а что делать, если нам нужно сложить слова и числа? - спросил Яша. — Вот, например, можно ли так написать: `"девочка Ляля вчера хохотала " + 5 + " раз"`? Или числа и фразы настолько не дружат, что их нельзя складывать?

— Можно, Яша! Если к фразе прибавить число, то это число само превращается в строку, поэтому в твоем примере при сложении получится фраза: `"девочка Ляля вчера хохотала 5 раз "`.

— Ага, понял! То есть числа в фразу сами превращаются, а вот фразы в числа уже не могут превращаться? — догадался Яша.

— Совершенно верно! А все потому, что любое число всегда может стать фразой, а вот далеко не любая фраза может стать числом.

Присваивание одновременно с созданием

— Интик, а можно ли присвоить значение монстрику одновременно с его созданием? — спросил Яша.

— Да, такое возможно. Смотри, как это будет выглядеть:

```
  /* создать монстрика limonad и пусть он сразу запомнит число 5 */
  int limonad = 5;
```

— Значит, сначала указываем семейство монстрика, затем его имя, и раньше бы мы поставили точку с запятой, означающую конец команды, а сейчас мы поставим знак присвоить «=» и число, которое должен запомнить монстрик при создании. И вот только после этого завершаем строчку точкой с запятой. — повторил для себя Яша.

Как монстрики превращаются

— Ну а основное, за что мы ценим монстриков — это за способность к превращению. Каждый раз, когда Создатель Планет встречает имя монстрика в программе, он его заменяет тем значением, которое монстрик помнит!

— То есть, если я напишу: `int limonad...`

— Не-не-не, я оговорился. Всегда, за исключением двух случаев.

Первый, это как раз, как ты сказал, когда мы только создаем монстрика. Он ни во что не превращается, так как он еще ничего не помнит, ему не во что превращаться, да и вообще не зачем!

А во втором случае, если монстрик стоит слева от знака присваивания, то есть от знака «=». Ему в этот момент запрещено превращаться! Потому что специально для него готовят значение справа от знака «=», которое он и запомнит!

— Здорово, давай еще что-нибудь посоздаем и поскладываем! — попросил Яша.

— Хорошо, начинай! — согласился Интик.

— Как прошлый раз, фразы у нас будут помнить монстрики СЛОВОПОМ и СЛОВОПУС, а числа — МАТЕПУС и МАТЕПОМ.

— Так мы уже делали! Давай тогда мы не просто создадим монстриков, как прошлый раз, а еще и сразу при создании попросим их запомнить что-то? — спросил Интик.

— Ладно, я попробую, — ответил Яша.

```
/* String,  вот какому семейству пусть принадлежит новый
монстрик СЛОВОПОМ, и пусть он сразу запомнит фразу "Яша съел"
*/
String slovopom = "Яша съел";

/* String,  вот какому семейству пусть принадлежит новый
монстрик СЛОВОПУС, и пусть он сразу запомнит фразу "эскимо" */
String slovopus = "эскимо";

/* int,  вот какому семейству пусть принадлежит новый монстрик
МАТЕПУС, и пусть он сразу запомнит число 1 */
int matepus = 1;

/* int,  вот какому семейству пусть принадлежит новый монстрик
МАТЕПОМ, и пусть он сразу запомнит число 23 */
int matepom = 23;
```

— А теперь мы поиграем в Яшу, который ест мороженное! — сказал Интик, и написал следующее присваивание:

```
slovopom = slovopom + matepus + slovopus;
```

— Что у нас получилось? — спросил Интик.

— Ээ, — протянул Яша, пытаясь сосредоточится, — раз СЛОВОПОМ стоит слева, а за ним сразу знак «=», значит, он должен запомнить то, что стоит справа от знака «=», а это значит...

А это значит запомнить фразу, которая состоит из прилипших друг к другу следующих частей, которые помнят монстрики СЛОВОПОМ, МАТЕПУС и СЛОВОПУС: "Яша съел", «1», "эскимо".

Значит, СЛОВОПОМ будет помнить: "Яша съел1эскимо".

— Молодец! — похвалил Интик.

— А как сделать так, чтобы они не так сильно слипались? — спросил Яша.

— Можно добавить при сложении разделительный пробел, то есть знак " ", — ответил Интик, — вот смотри:

```
slovopom = slovopom + " " + matepus + " " + slovopus;
```

— Расшифровываем...

```
slovopom = "Яша съел" + " " + 1 + " " + "эскимо";
```

— Теперь слепляем, и получаем: "Яша съел 1 эскимо".

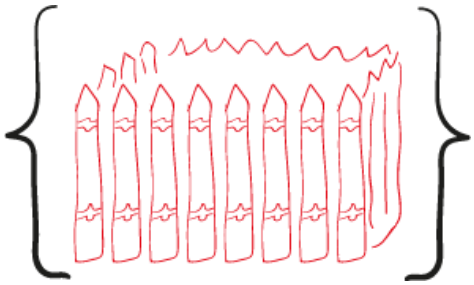
— Интик, а что если мы теперь МАТЕПУСА заменим на МАТЕПОМА? — спросил Яша.

— Тогда мы получим Яшу, у которого заболело горло! — захохотал Интик.

Глава 2. Заборы, калитки и территории

Заборчик

— Больше всего на свете мы любим порядок! — воскликнул Интик.



— Что ты имеешь ввиду?

— Все наши команды должны находиться на своей территории. А у каждой территории должен быть свой забор! Знаешь, как мы строим забор?

— Наверно какими-то знаками огораживаете?

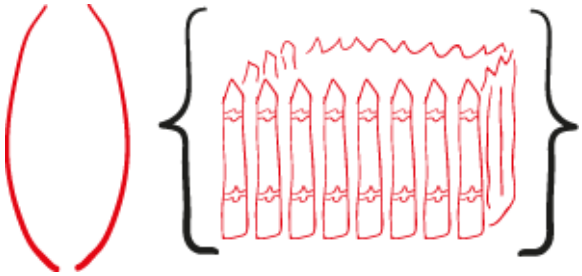
— Начало забора мы обозначаем таким значком «{» — открывающей фигурной скобкой. А конец забора обозначаем значком «}» — закрывающей фигурной скобкой. И все команды в нашей программе должны находится за своим заборчиком.

Вот так:

```
{ мои переменные и мои команды; }
```

Калитка

— А ведь если есть забор, ограждение, то должна же быть и калитка? — догадался Яша.



— Совершенно верно! А калитку мы обозначаем двумя круглыми скобочками, так, чтобы было похоже на вход! — ответил Интик.

— А вход можно в любом месте расположить? — спросил Яша.

— Нет, калитка должна быть всегда в самом начале, перед заборчиком. Чтобы ее не приходилось искать!

Заборчик с калиткой у нас выглядит так:

```
(){ мои переменные и мои команды; }
```

Территория

— А как узнать кому какой заборчик принадлежит? — спросил Яша.

— Так это просто, у каждого заборчика обязательно должно быть свое имя!

— А где его пишут?

— Его пишут в начале, перед калиткой, то есть перед первой круглой скобкой!

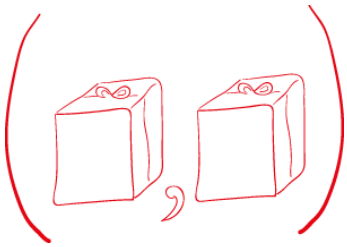
— Ну, это я понимаю, это логично, как вывеска над домом. Прочитал и сразу понятно — чья дальше территория.

— Ага, вот смотри, как это выглядит:

```
мояТерритория() { мои переменные и мои команды; }
```

Подарки от гостей

— А вот в некоторых территориях на входе требуют приходить с подарками!



— Ну, ничего удивительного, — заметил Яша. — Это как на день рождения в гости, когда идешь, попробуй только приди без подарка! Так посмотрят на тебя, что восьмой кусок торта в горло не полезет!

— Вот-вот, а у нас подарки сразу в калитке требуют! Прямо через запятую и требуют!

— При чем угадай, кто на входе стоит и подарки принимает? — спросил Интик.

— Наверняка монстрики-переменные? — предположил Яша.

— Именно! — подтвердил Интик. — Их, кстати, еще «параметрами» кличут, тех, кто в круглых скобках стоит. А еще не все подарки принимают, а только те, которые могут принять эти самые монстрики.

— Это что получается — придешь не с тем подарком, так тебя еще и не пустят? — удивился Яша.

— Именно! — еще раз подтвердил Интик. — Раз пришли, будьте добры точь-в-точь угодить пожеланиям хозяев. А иначе — «крэш, бум, бэм», разозленные хозяева сразу работу всей Планеты парализуют, а по-нашему говоря, ошибка выскакивает!

— Вот привереды! — согласился Яша. — Но раз уж у вас такие правила — ничего не попишешь, ничем не порисуешь. Расскажи тогда, как они эти подарки принимают, и как их передавать?

— Ну, это не сложно. Если территория ожидает подарков, то в круглых скобочках она создает монстриков для принятия этих подарков. Вот, например, кусочек территории покажу — калитку, где требуют два подарка числа. Первый

подарок должен быть числом, и его примет новый монстрик по имени «параметр1», и второй подарок тоже должен быть числом, а его примет новый монстрик по-имени «параметр2»:

```
мояТерритория(число параметр1, число параметр2) { мои переменные и мои команды; }
```

или в закодированном виде сама калитка:

```
(int parametr1, int parametr2)
```

— Ой, Интик, — Яша схватился руками за голову, — мне здесь непонятно сразу столько всего!

— Так спрашивай!

— Ну, во-первых, мы никогда раньше не создавали монстриков-переменных, у которых в имени были бы цифры! Это как-то непривычно.

— Да, это я дурья башка, забыл тебе сразу сказать, что имена монстриков запросто могут содержать в себе цифры. Но главное, чтобы начиналось имя с буквы! А дальше, хоть буквы, хоть цифры, хоть кубом покати — не важно!

Иногда такие имена очень удобны. Например, если мы сказку программируем. Как там у вас, «Али-баба и сорок разбойников». Вот мы каждому разбойнику можем и имя простое придумать: `разбойник1`, `разбойник2`, `разбойник3`, `разбойник4`...

— Погоди-погоди, Интик, `разбойник5`, `разбойник6` и так далее. Я понял! — перебил Интика Яша.

— Ну да, а еще какой у тебя вопрос был?

— Мы когда монстриков создавали, всегда строчку заканчивали точкой с запятой, то есть знаком «`;`». А здесь откуда-ни-возьмись запятая! — удивился Яша.

— Это когда мы независимых монстриков создаем, тогда завершаем строчку точкой с запятой, а вот когда они в калитке стоят, в круглых скобках заключены — тогда они разделяются запятой. При чем заметь, что именно разделяются, то есть между ними запятая ставится, поэтому получается, что после последнего монстрика уже нет никакой запятой, а сразу идет закрывающая круглая скобка.

— А если только один монстрик в калитке окопался? — спросил Яша.

— В этом случае разделять запятой ничего же не нужно? Значит, ее вообще не будет, вот так:

```
(int parametr1)
```

Подарки гостям

— А бывает так, что выходя из территории подарок дарят?

— Конечно! Тогда перед названием территории нужно указать, какому семейству монстриков должен принадлежать этот подарок: семейству чисел, или семейству фраз, ну или еще чего. Выглядеть это будет например так:

```
int мояТерритория() { мои переменные и мои команды; }
```

— А как этот подарок себе получить-то? — спросил Яша.

— Здесь все просто. Ты же сам не можешь его получить? Планета — там, а ты — здесь. Поэтому вместо тебя подарок получит кто?

— Наверно, опять какой-нибудь монстрик-переменная? — предположил Яша.

— Ха, а ты догадливый, — похвалил его Интик. — Именно так, чтобы получить подарок, тебе нужен подходящий монстрик-переменная. Вот, например, числовой подарок получить сможет числовой монстрик. А подарок в виде фразы — монстрик из семейства словесных. А получает он его обычным присваиванием. Вот смотри:

```
// Сначала мы создадим монстрика из числового семейства
int matepus;

/* а теперь отправим его за подарком в функцию мояТерритория(), которая по-английски пусть пишется moyaTerritoriya(); */
matepus = moyaTerritoriya();
```

— А вот теперь следи за руками, в чем фокус-кактус. Когда Создатель Планет видит название какой-то территории, после которой стоят круглые скобочки, а за ними сразу точка с запятой, то он воспринимает это, как приказ пойти в тридесятое царство, восемьдесят второе королевство, в общем, пойти в эту неизведанную территорию. Сходит он значит в нее, а там подарок возвращают — число. И куда ему это число деть?

— Наверно, отдать МАТЕПУСУ? — предположил Яша.

— Верно!

— Хорошо, давай вернемся к созданию территории. А что, если подарка при выходе с территории не дарят?

— Тогда нужно вместо типа подарка указать слово «void», что произносится, как «войд», а означает — пусто. То есть вместо

подарка будет — пу-сто-та!
Выглядит это так:

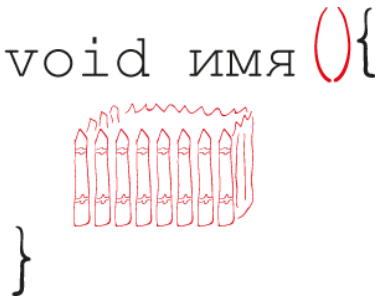
```
void мояТерритория() { мои переменные и мои команды; }
```

Как принято писать

— Территорий у нас в программе может быть много, а еще бывают территории внутри других территорий, — начал Интик.

— Похоже на государство, внутри которого город, а внутри города дома, — заметил Яша.

— Это ты хорошо придумал! Очень похоже! Так вот, так как территорий может быть много, то есть некоторые правила оформления. Так команды не принято писать сразу после открывающей фигурной скобки, их пишут с новой строки, делая небольшой отступ слева. А закрывающую фигурную скобку тоже пишут на новой строке. Вот смотри.




```
void мояТерритория() {
   мои переменные и мои команды;
}
```

Стандартные территории setup и draw

— А мы какие хотим, такие территории и создаем?

— Да, мы можем создавать разные территории и давать им свои имена, как это делать я тебе расскажу попозже. А сейчас обращу твое внимание на один важный момент! В каждой программе должна быть территория под именем `setup` и территория под именем `draw`. Слово `setup` читается, как «сэтап», и переводится, как «установка», а `draw` читается, как «дро», а переводится, как «рисование».

— А зачем нужны эти территории?

— Когда Создатель Планеты принимает нашу заявку, а затем, в соответствии с инструкциями в ней начинает создавать планету он начинает выполнение с посещения этих территорий. Поэтому если мы их не создадим — то он просто не будет знать, куда ему идти!

— А как это выглядит?

— Вот смотри.

Создаем территорию `setup`:

```
void setup() {
  мои переменные и мои команды;
}
```

Создаем территорию `draw`:

```
void draw() {
  мои переменные и мои команды;
}
```

— У этих территорий есть свои особенности! — вдруг вспомнил Интик.

— Какие?

— В территорию `setup` Создатель Планет заглядывает только один раз, при создании планеты, поэтому `setup` так и переводится, как «начальная установка».

— А в территорию `draw` он что, заглядывает несколько раз?

— Да не просто несколько! Стоит ему из нее выйти, как он снова ищет территорию `draw` и снова в нее заходит! И так до тех пор, пока программа работает!

— Заколдованный круг!

— Именно! Мы еще называем ее «Вечный цикл». Создатель Планет заходит в территорию `draw` и выполняет первую указанную в ней команду, затем спускается на строчку ниже и выполняет следующую, и так до тех пор, пока не

выполнит все команды, а после последней, он выходит из территории `draw`, забывает все, что он там натворил, и снова в нее заходит!

— Точно, заколдованный круг какой-то! — окончательно уверился Яша.

— А раз Создатель Планет вечно крутится в территории `draw()` я знаю, как песенку записать смешную! — поделился Яша. — Вот смотри:

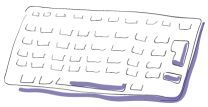
```
void draw() {
/* Чучело-мяучело
   На трубе сидело.
   Чучело-мяучело
   Песенку запело.

   Чучело-мяучело
   С пастью красной-красной —
   Всех оно замучило
   Песенкой ужасной.

   Всем кругом от чучела
   Горестно и тошно,
   Потому что песенка
   У него про то, что */
}
```

— Получилась бесконечная песенка, правда?

— Ага! — согласился Интик.

Подсказка

Как набрать на клавиатуре фигурные скобки «{» и «}».



Обычно, клавиши с фигурными скобками совмещены с клавишами русских букв «Х» и «Ъ» справа на клавиатуре.

Чтобы получить фигурные скобки нужно нажать клавишу «Shift» и одну из этих клавиш, когда включен режим ввода текста на английском языке.

Как набрать на клавиатуре круглые скобки «(» и «)».

А эти клавиши совмещены с цифровыми клавишами «9» и «0».

Чтобы получить круглые скобки нужно нажать клавишу «Shift» и одну из этих цифровых клавиш, когда включен режим ввода текста на английском языке.

Глава 3. Функции

Команды и Функции

— Когда нам нужно что-либо сделать, мы пишем письменный приказ, это еще называется «вызывать функцию».

— А откуда такое странное название «вызвать функцию»? — спросил Яша.

— За выполнение нашего приказа отвечает специальная территория, мы их называем еще Ремесленными домами. Видел же у себя на планете дома с вывесками: «Печем пироги», «Чиним сапоги», «Красим дома»?

— Ага.

— Вот, каждый раз при создании планеты создается целая прорва таких Ремесленных домов, а так как каждый из них выполняет какую-либо функцию, то мы их просто и называем — Функции! — пояснил Интик.

— Понятно, — протянул Яша, — то есть в нашем распоряжении много-много функций и мы можем ничего сами не делать, а только приказывать?

— Ничего не делать, конечно, не получится, но пользоваться функциями и правда удобно.

— А как это сделать?

— Для этого мы пишем название той функции, которую хотим вызвать, а после ее имени ставим пару круглых скобок, а за ними знак «;» — точку с запятой.

— А круглые скобки зачем? Они мне напоминают калитки у территорий, — спросил Яша.

— В круглых скобках мы как раз кратко формулируем наш приказ, вносим в него уточнения. Эти уточнения как раз и называются «параметрами».
Некоторым функциям параметры не нужны, тогда круглые скобки стоят вплотную друг к другу, а когда нужны уточнения, они указываются в круглых скобках через запятую.

`имяФункции ();` — это вызов функции без параметров

`имяФункции (параметр1, параметр2);` — а это вызов функции с двумя параметрами

— Не очень-то понятно, можно как-то на примере пояснить? — попросил Яша.

— Вот смотри, например, у нас есть функция, которая устанавливает цвет нашего экрана, но для работы ей нужно знать, какой цвет мы хотим установить.

— То есть вызов этой функции будет выглядеть так?

```
установитьЦветЭкрана(зеленый);
```

— Да, совершенно верно!

— А как узнать какие функции существуют и как ими пользоваться? — спросил Яша.

— А это я тебе и буду постепенно рассказывать! Вот в самое ближайшее время поведаю про функции `size()` и `ellipse()`.

Часть 2. Начала Программирования

Предисловие

Как установить процессинг

— Что-то мы с тобой заговорились, заболтались, — сказал Интик, — давай уже перейдем к практике, и что-нибудь такое-эдакое наколдуем!

— Да я согласен! Вот только еще не знаю, как сделать так, чтобы мои заклинания оживали!

— Ну, это просто. Тебе нужен исполнитель заклинаний, то есть сам Процессинг (Processing)!

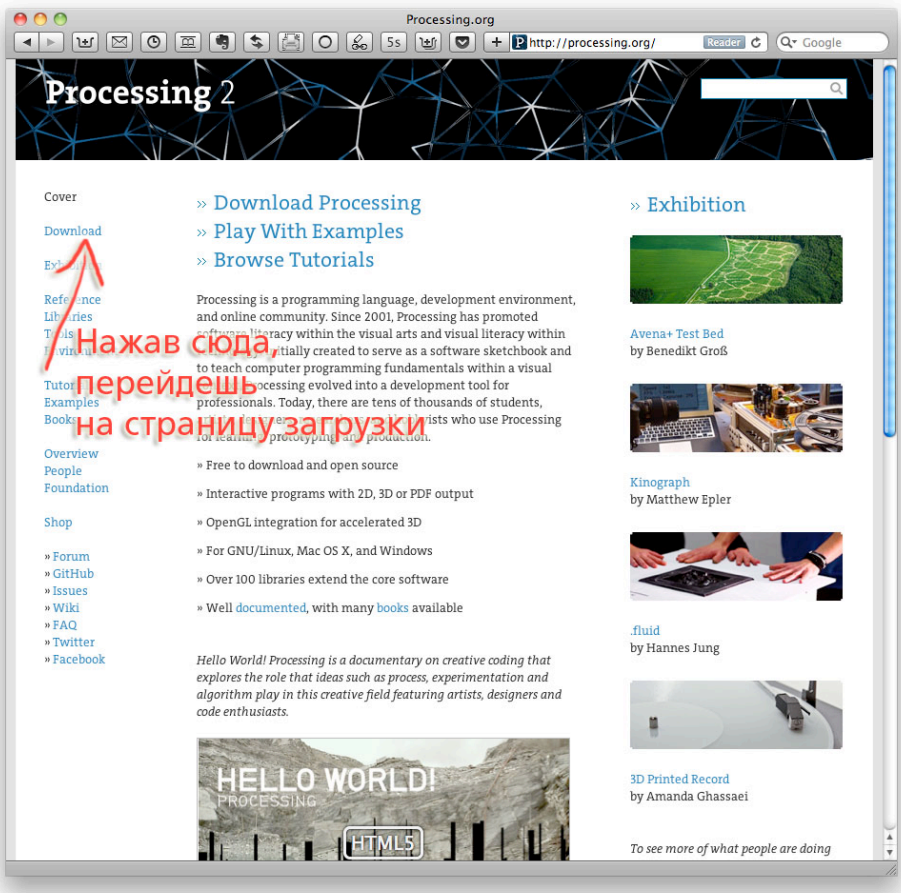
— А где его взять? — поинтересовался Яша.

Интик достал ноутбук, и открыл браузер.

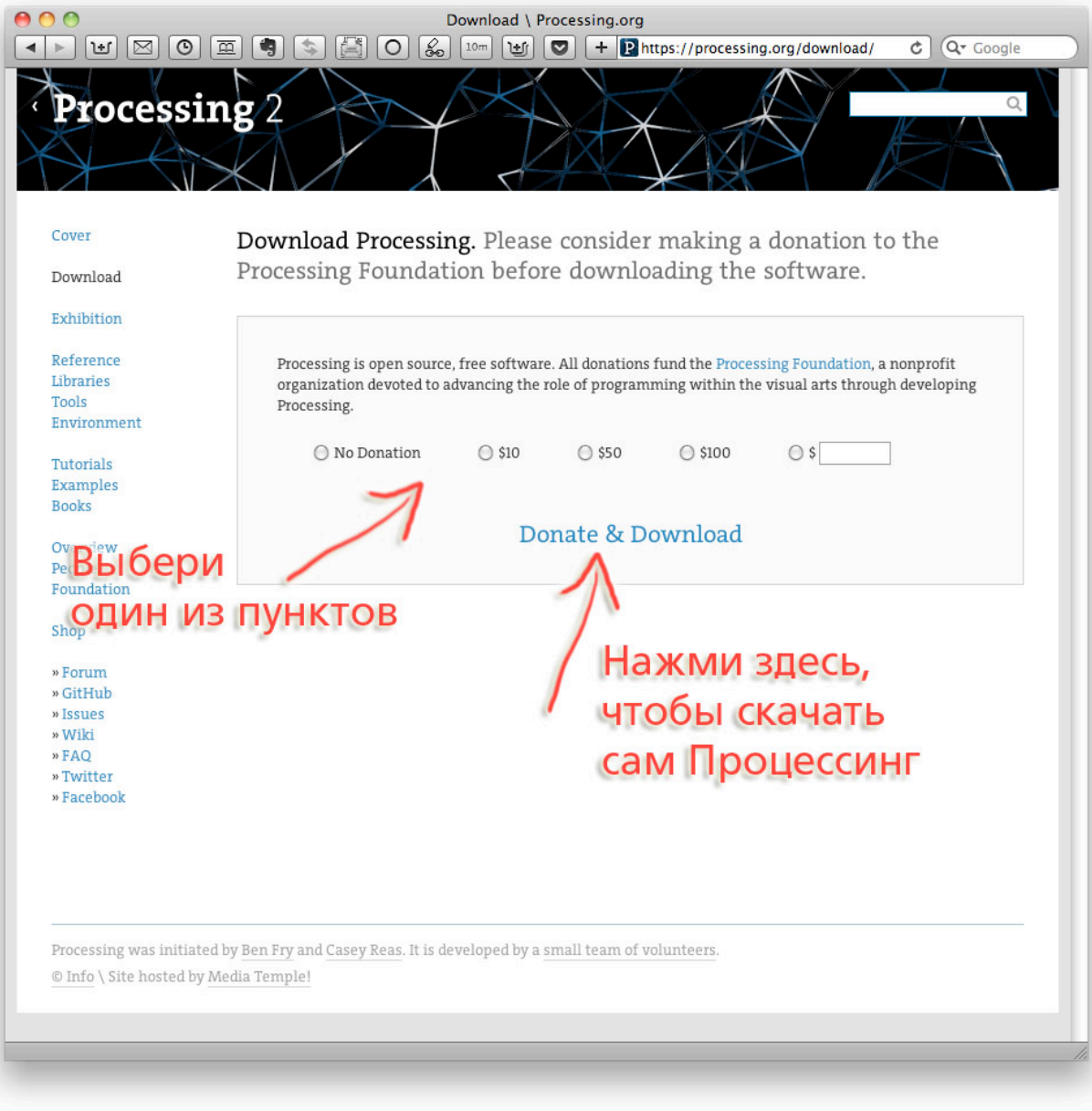
— Вот смотри, набираем код галактической ассоциации Процессинга в строчке браузера: http://processing.org

— Ой, открылось что-то непонятное на английском языке!

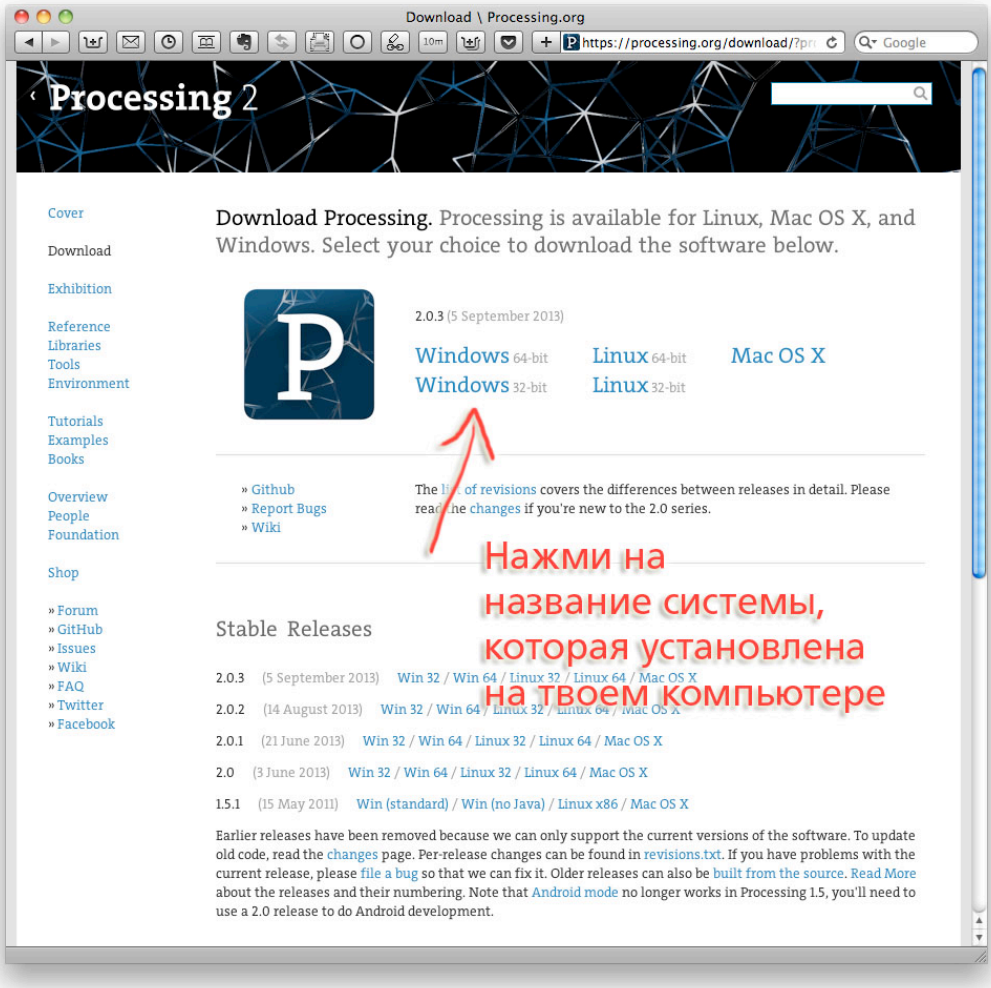
— Ну да, ты главное вот найди ссылку «Download», нажми на нее, — пояснил Интик.



— Ого, открылась еще одна страница, — удивился Яша.



— Выбери один из пунктов и нажми на ссылку внизу.
— И еще одна страница!



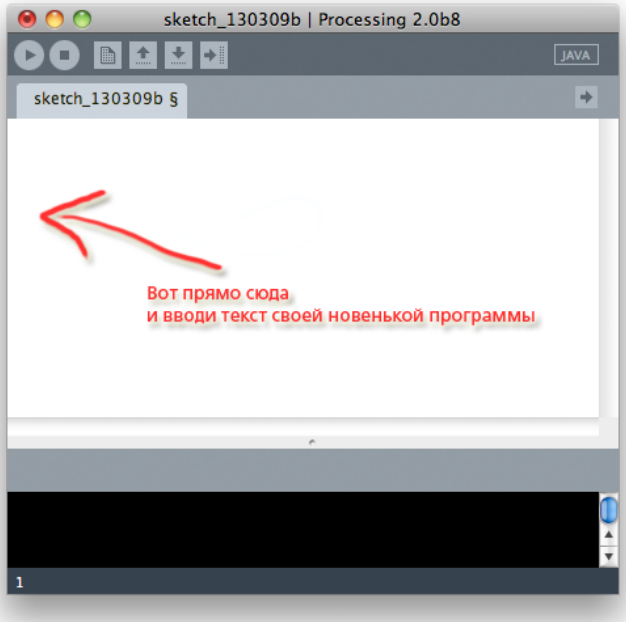
— А теперь на ней нажми на слово Windows 32-bit, если у тебя на ноутбуке установлен Windows, или Mac OS X, если установлена Mac OS.

— Вот теперь скачался какой-то файл, в конце которого написано «.zip».

— Это архив самого Процессинга! Нажми правой кнопкой на этом скачавшемся архиве и выбери «Разархивировать». И когда все разархивируется, а процесс может оказаться не быстрым, зайти в получившуюся папку и просто запусти файл с названием «Processing»!

Как создать программу

Когда запустим скачанный Процессинг, то увидим следующее окно:

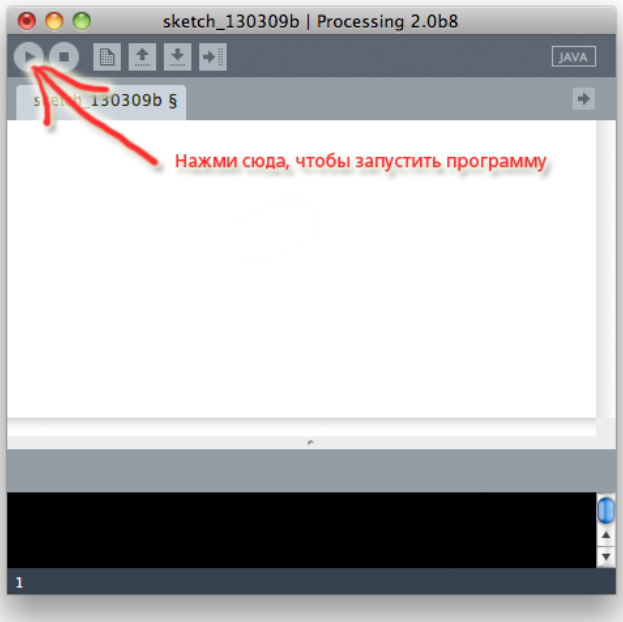


Вот прямо в этом окне и вводим наши заклинания!

— Так просто! — удивился Яша.

Как запустить программу

— А запустить программу, то есть передать ее Создателю Планет, еще проще! — воскликнул Интик, — смотри на картинку ниже:



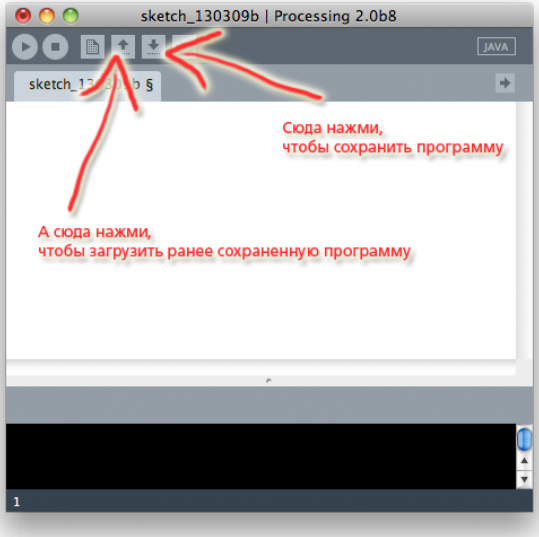
— Или для скорости можешь нажать комбинацию клавиш Ctrl-r!

Как сохранить и загрузить программу

— А вот если ты решил сделать паузу в своей работе, то, чтобы твоя программа не потерялась, ее стоит сохранить! — пояснил Интик, — тогда ты можешь даже компьютер выключить, а твои труды не потеряются.

— А потом я могу открыть то, что сохранил? — спросил Яша.

— Ага! Вот смотри, на какие кнопочки нужно нажимать.



— Еще, кстати, не каждое имя программе можно дать при сохранении! — добавил Интик. — В имени не должно быть пропусков, то есть пробелов, а также оно должно начинаться с буквы обязательно!

Как редактировать текст

РЕЖИМ ЧТЕНИЯ:
повышенная сложность

— Вот тебе некоторые советы, как быстрее вводить и редактировать свою программу, — сказал Интик.

Если ты случайно удалил что-то важное, то всегда можешь сделать шаг назад. Для этого нажми комбинацию клавиш Ctrl-z (для этого нажми и держи клавишу «Ctrl», после чего нажми клавишу «z»).

Для увеличения скорости ввода ты можешь копировать и вставлять целые куски текста. Для этого нажми кнопку мыши и выдели нужный тебе кусок текста. Когда он будет выделен, ты можешь запомнить его, нажав комбинацию клавиш Ctrl-c. А можешь его одновременно удалить из старого места, при этом запомнив, это называется «вырезать», для чего нажми Ctrl-x.

Если хочешь вставить запомненный кусок текста, то нажми в нужном месте мышкой, чтобы курсор ввода текста там оказался, после чего нажми Ctrl-v.

Чтобы твоя программа была всегда красиво оформлена, нажимай иногда Ctrl-t. Это ни капельки не изменит текст программы, но зато автоматически поставит правильные отступы у каждой строчки, если вдруг где-то они стоят не ровно. Весьма рекомендую, так как облегчает поиск ошибок!

День первый. Первая программа «графический редактор»

Функция: размер окна, size()

Одно из базовых заклинаний любого мага-программиста — это функция «`size()`».

Читается, как «сайз», по-русски значит «размер», имея ввиду размер окна нашего телескопа, через которое мы будем видеть все, что происходит на планете.

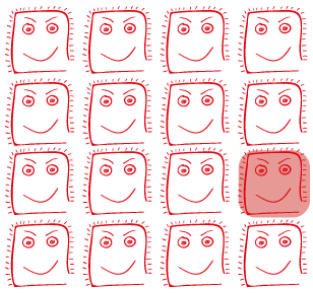
— Интик, мне кажется, что здесь чего-то не хватает. Функция «`размер()`» — это же значит, что установи мне размер окна. А какой размер? Большой, маленький, узкий или высокий? Ничего не понятно!

— Ты прав, Яша! Поэтому эту функцию всегда пишут с уточнениями, которые, как ты помнишь, мы называем параметрами.

У функции «`size()`» этих параметров два: `шириналкна` и `высотаОкна`.

```
size(шириналкна, высотаОкна);
```

— Интик, а в чем измеряется ширина и высота окна? Вот у нас на планете Земля все измеряется в метрах и сантиметрах! Удавов мы, правда, считаем в попугаях.



— А у нас все измеряется в пикселях. Это такие микробы, из которых состоит наш экран. И когда нам нужно, например, нарисовать точку на экране — мы говорим, что-то вроде: «Эй, микроб, тот, который двадцать пятый сверху, и сто восемьдесят второй слева, прими красный цвет!».

Интик добавил:

— И кстати, место микроба в общем микробном пространстве мы называем умным словом «координаты». То есть координаты — это всегда пара чисел, где первое число — расстояние слева, а второе число — расстояние сверху.

— А зачем вообще такое слово придумывать? — спросил Яша, — мне кажется, оно только мозги пудрит!

— Ну, может и пудрит, — согласился Интик. — Но, обычно, бывает удобно.

Немного подумав, Интик еще добавил:

— Кстати, функция `size()` должна всегда быть самой первой функцией в территории `setup()`! И если ее написать не самой первой, то все может начать глючить, будут выползать из темноты жуки «баги», и вообще все будет страшно и непонятно!

— А почему первой?

— Потому что именно эта функция устанавливает размер мира, то есть окна! И если мы сначала начнем что-то делать, а затем попытаемся установить размер окна, то все обидятся, и все может пойти кувырком!

— Значит, писать нужно так:

```
void setup(){
  size(400, 400);
  // а здесь уже все остальные команды
}

void draw(){
  /* а здесь команды, которые будут повторяться из раза
в раз */
}
```

— Ага!

Монстрики-аборигены

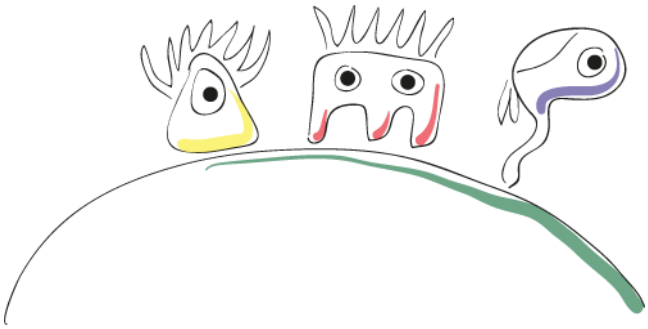
— Вот смотри, Яша, до сих пор мы создавали сами монстриков-переменных!

— Ага, а еще присваивали им всякие разные числа и слова!

— Но представь себе, что при создании планеты на ней уже живут монстрики-переменные, мы их еще называем монстриками-аборигенами. А раз они уже живут там, то их и создавать не нужно!

— Любопытненько, — заинтересовался Яша. — Они, наверное, что-то интересненькое знают?

— Да, одни монстрики помнят размер планеты, другие — в каком месте на экране у тебя мышка сейчас находится, третьи - еще что-нибудь, я тебе про них потом расскажу.



монстрики-аборигены сидят и ждут, когда у них что-нибудь спросят

Главное, что про них нужно помнить, они очень уж самостоятельные и гордые, и не терпят, чтобы им указывали, что запоминать.

— То есть я монстриков-аборигенов не могу попросить чего-нибудь запомнить? — удивился Яша.

— Да, ты можешь разве что у них спрашивать, что они знают. Ох, как они любят давать советы, вот прямо сидят и ждут, кто же придет, и у них чего-нибудь да спросит!

Монстрики-аборигены: мышкаИКС и мышкаИГРЕК

— А сейчас я тебе расскажу о некоторых монстриках-аборигенах! — начал рассказывать Интик.

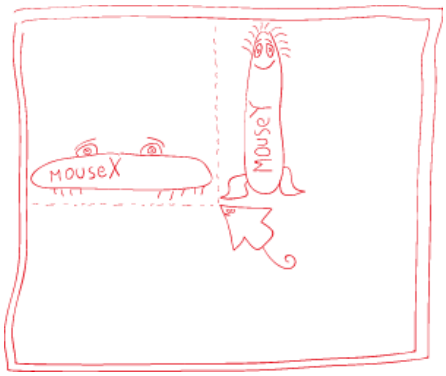
Так, у нас всегда есть два монстрика-переменных:

- первый по имени `mouseX`

(звучит, как «маусИКС», на русский переводится, как мышкаШИРИНА),

- а второй по имени `mouseY`

(звучит, как «маусИГРЕК», а переводится, как мышкаВЫСОТА).



Первый, `mouseX` — низенький такой, но широокий, он хранит число. Причем не простое число, да и не золотое, а равное расстоянию от мышки до левого края экрана.

Второй, `mouseY` — высокий, но худенький. Он помнит число, равное расстоянию от мышки до верхнего края экрана.

Эта дружная парочка часто ходит рядом, и к ним очень удобно обращаться, когда нам нужно узнать, где у нас на экране мышка спряталась!

Например, `mouseX` помнит, в каком пикселе, если считать слева-направо — спряталась мышка. А `mouseY` — в каком пикселе, если считать сверху-вниз.

Функция: овал, ellipse()

— Предположим, у нас на экране поселились сплюснутые круги, вытянутые круги, то есть всякие разные овалы. Как мы их сможем отличить друг от друга? — спросил Интик.

— Ну, во-первых, по форме, — предположил Яша. — Одни овалы будут невысокие, гномы такие, а другие, наоборот, высокие, как эльфы или швабры.

— Швабры? — с недоумением переспросил Интик. — Про эльфов я слышал, а вот про швабр не приходилось.

— Как же ты про них не слышал, — удивился Яша, — а чем вы пол вытираете?

— Ах, эти швабры, — засмеялся Интик. — Так вот, — сделав паузу он продолжил, — это «`высотаОвала`», а еще?

— Еще по ширине могут отличаться, — продолжал Яша. — Одни такие широкие, что редкая птица долетит до их середины. А другие — узкие, словно щель между передними зубами Антона Милошниченко, моего одноклассника, сквозь которую у него даже плеваться не получается.

— Ага, это «`ширинаОвала`», — заметил Интик, пропустив пассаж про плевки. — Еще чего-нибудь?

— Да вроде все.

— А как же их расположение на экране? Одни ближе к левому краю экрана, другие подальше.

Некоторые ближе к верхнему краю экрана, а какие-то дальше от него.

Итак, вот свойства нашего овала:

- `расстояниеОтЛевогоКраяЭкрана`,
- `расстояниеОтВерхнегоКраяЭкрана`,
- `ширинаОвала`,
- `высотаОвала`

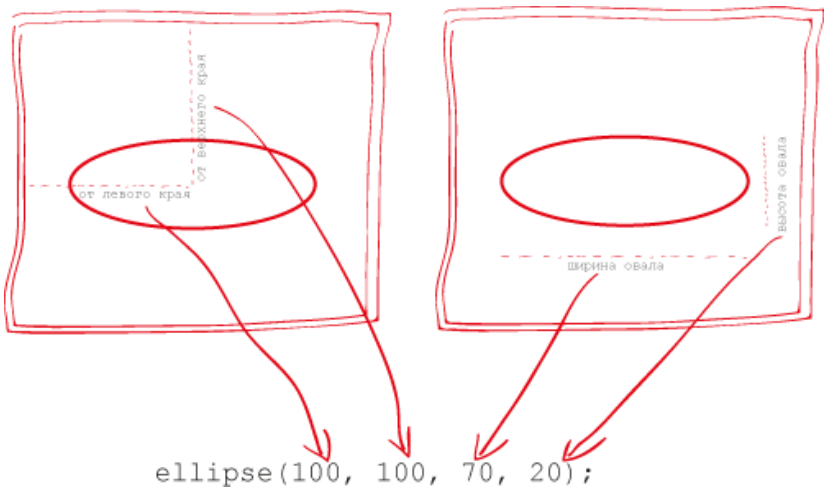
Тут Яша задумался и спросил:

— Расстояние от левого края экрана — это понятно. Но ведь это расстояние можно измерить или до левого края овала, или до правого, а еще до центра овала можно. И все эти расстояния будут же разными! — что имеется ввиду под этим расстоянием, я недопонимаю!

— Очень правильный вопрос, — похвалил Интик. — От какой части экрана я сказал, а вот до какой части овала — забыл. Исправляю этот недочет. Расстояние всегда считается до центра овала! Прямо до самой его что ни на есть серединки.

— А дальше принцип прост. Когда мы хотим нарисовать эллипс, то есть овал, мы вызываем функцию «`ellipse();`», читается, как «ил**И**пс», переводится, как «овал», и передаем ей четыре значения-параметра, строго в следующем порядке:

```
ellipse(расстояниеОтЛевогоКраяЭкрана,
расстояниеОтВерхнегоКраяЭкрана, ширинаОвала,
высотаОвала);
```



— Хочу овалы порисовать, — заявил Яша.

— Давай, давай, — подбодрил его Интик. — просто так, или какие-нибудь особенные?

— Хочу, чтобы в верхнем-левом углу у меня жил маленький круг.

Сначала установлю размер окна, шириной и высотой в 400 пикселей-микробов:

```
size (400, 400);
```

Вот это мой маленький круг, размером 30 на 30 микробов:

```
ellipse(30, 30, 390, 50);
```

— Яша, разреши, я тебя перебью, — вклинился Интик, — мне почему-то кажется, что ты перепутал порядок параметров, при вызове функции «ellipse();». Помнишь, сначала идет расстояние до левого края, потом до верхнего, затем ширина овала, затем высота.

— Ой, и правда. У, вместо маленького круга 30 на 30 у меня получился большой сплюснутый овал шириной 390 и высотой 50 пикселей. Сейчас поправлю, поменяю их местами:

```
ellipse(390, 50, 30, 30);
```

— Вот теперь, и правда круг, — согласился Интик. — Вот только непохоже, что ты его в левом верхнем углу разместил, больше похоже что ты его засунул в правый угл! Да и я бы сказал, что не просто засунул, а еще и хвост ему прищемил!

— Как это прищемил?

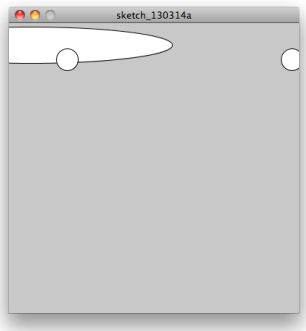
— Смотри, ты его разместил на расстоянии 390 пикселей от левого края, а это почти впритык к правому краю, и он просто не помещается в окне!

Вот и получается, что у тебя на экране уже не круг, а какая-то горошина, которую травмвай переехал.

— Хм! тогда вместо 390 пикселей нужно число поменьше поставить, хотя бы 80. Пусть будет так:

```
ellipse(80, 50, 30, 30);
```

— Отличненько, — согласился Интик, - вроде все в порядке! А вот если все твои попытки на одном экране изобразить, знаешь, что получится? Вот смотри, я нарисовал:



Яша пишет программу. Следы от мышки

— Интик, я вот уже мечтаю, когда же я наконец-то смогу настоящую программу написать! Все мне не терпится! — сказал Яша и прищурившись посмотрел на Интика.

— Так ты уже столько всего знаешь, зачем же дальше откладывать! — согласился Интик. — Что будешь писать?

— Хочу, чтобы я водил мышкой по экрану, за ней бы оставались следы в виде маленьких кружков!

Яша продолжал:

— Окошко я сделаю небольшим, микробов 250 на 250:

```
  size(250, 250);
```

Тут вмешался Интик:

— Если ты уже программу пишешь, то ты кое-что забыл!

— Вот... Сейчас... Подожди... — Я пролистал свои заметки, и не нашел ничего такого, чтобы я так уж прямо и забыл!

— А как же территории!

— Ах, да-да, все вызовы функций должны на быть на какой-то территории. Так, размышляем дальше.

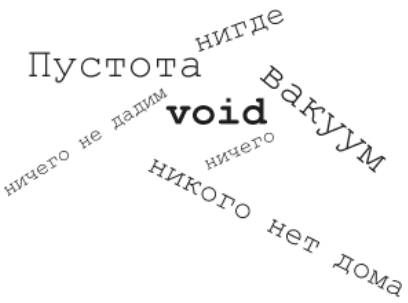
Я знаю о двух территориях: `setup()`, в которую Создатель Планет один раз заходит, при создании Планеты, и `draw()`, в которую он заходит сразу после этого. А после `draw()`... а после `draw()` он снова заходит в `draw()`. Наверно там медом намазано. Да, тут я все вспомнил.

Если я помещу вызов функции «`size()`» в `draw()`, то он создаст нужный мне размер окна, выйдет из `draw()`, снова в нее зайдет, и опять попытается установить размер окна. Как-то глупо получается, мне нужно один раз его установить, а потом в нем рисовать.

Значит, размещу в территории `setup()`! — завершил свои размышления Яша, и вот что написал:

```
setup() {
  size(250, 250);
}
```

— Яша, Яша! — сразу же вмешался Интик. — А как же `void`? Всегда, когда ты создаешь территории `setup()` или `draw()`, перед ними нужно указывать слово `void`. Вот попробуй запустить свою программу прямо сейчас, без этого `void`, и посмотрим, что получится!



— Ладненько, нажима-а-ем на запуск, и получа-а-ем...

Первая ошибка и ее исправление

— И получаем вот что, ошибку! — воскликнул Яша.

— Он тебе написал личное сообщение, белым по красному! — заметил Интик. — «`Unexpected token: {`», что звучит ну примерно так: «анэкспэктэд тоукэн», я жирным выделил куда ударение в словах ставить нужно.

— Ы-ы... — только и смог произнести в ответ Яша.

— Вот тебе и «ы-ы», я же говорил, — чуть быстрее заговорил Интик, — Планета формалистов! Стоит хоть чуть-чуть ошибиться, и будут отказываться работать. Иногда, конечно, и помогут, на ошибки укажут, но все на своем бело-красно-английском диалекте! То, что сейчас написали, в переводе примерно будет так: «неожиданный значок: {».

— Хм, и правда, при чем тут неожиданный значок «`{`», если я слово `void` пропустил. Зато, хоть подсветили желтым цветом строчку, которая не понравилась.

— А знаешь, почему про «неожиданный значок» написали? Потому как думают, что ты не создаешь территорию, а вызываешь функцию!

— Как это? Ничего я не понял, — недоумевал Яша.

— А вот смотри, чем отличается вызов функции, от создания территории. Когда мы создаем территорию, мы поступаем примерно так же, как и когда создаем монстриков переменных. То есть первым словом указываем имя семейства: `int`, `String`, `void` или другое.

— Стоп-стоп-стоп, — вклинился Яша. — Неужто `void` — это тоже семейство?

— Ну да, только монстриков-переменных в нем не водится, только территории!

— Загадочно, как-то!

— Да все просто, в это семейство объединяются все территории, которые жадные и не возвращают подарков!

Ну, так вот, я продолжу. Сначала указываем имя семейства, а затем имя монстрика-переменной или же имя создаваемой территории. Но если после имени монстрика мы сразу ставим точку с запятой, то после имени территории сначала идет калитка, а затем заборчик.

— А когда мы вызываем функцию?

— Вот раньше мы уже вызывали функции «`size();`» и «`ellipse();`». В этом случае мы просто пишем их названия, калитку в виде круглых скобочек и точку с запятой.

— Так что же это получается, если я пропустил слово `void`, то они думают, что я не создаю территорию, а пытаюсь ее вызвать по-имени?

— Ага.

Хорошо, исправляю и дописываю `void`:

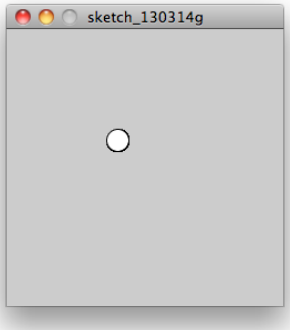
```
void setup() {
  size(250, 250);
}
```

Яша продолжал:

— А вот вызов функции рисования овала я размещу в `draw()`, потому как мне хочется, чтобы овалов рисовалось много!

```
void setup() {
  size(250, 250);
}

void draw() {
  ellipse(100, 100, 20, 20);
}
```

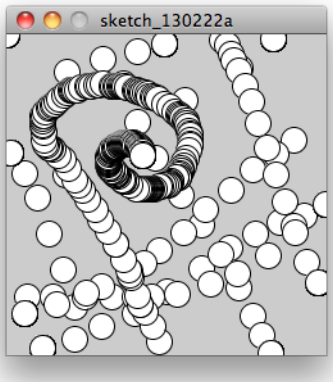
— И что у нас получится? Если запустим программу?
— Вот такой, небольшой кружок.



Когда что-то работает не совсем так, как этого хочешь

— Только он на мышку почему-то не реагирует, — задумался Яша. — Ах, да, я же забыл использовать монстриков-аборигенов. Ведь я же могу заменить любое число ими?

— Да, всегда вместо числа, можно написать имя числового монстрика. Тогда, когда Создатель Планет доходит до этого места, он заменяет имя монстрика на число, которое тот помнит. Это называется — превращение монстриков, помнишь, мы об этом раньше говорили.

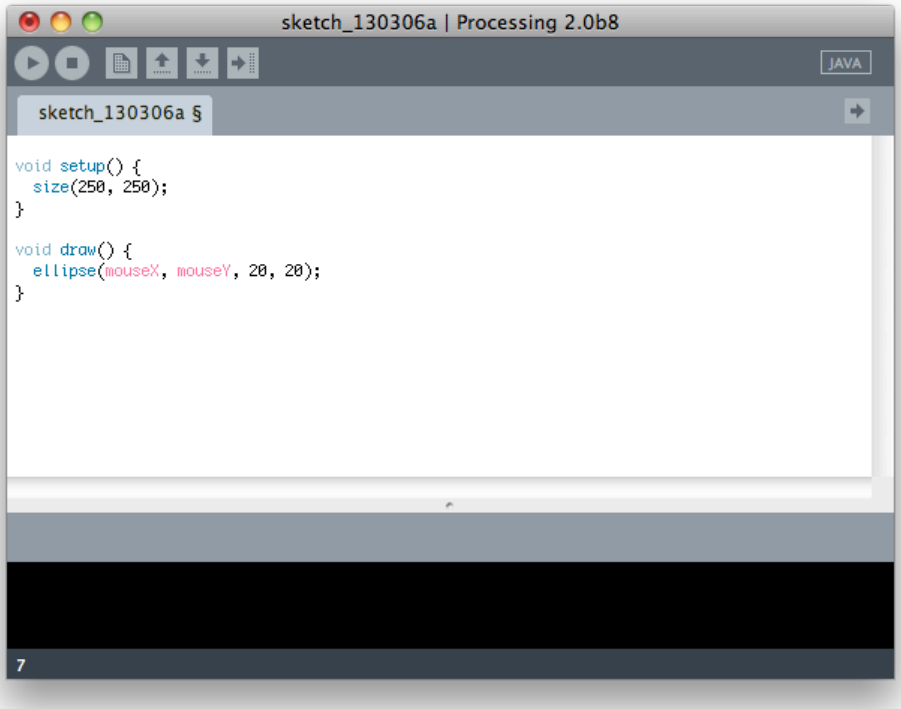


— Ага, ну тогда я первое число `100` заменю на `mouseX`, а второе — на `mouseY`:

```
void setup() {
  size(250, 250);
}

void draw() {
  ellipse(mouseX, mouseY, 20, 20);
}
```

— Ура! Я вожу мышкой по экрану. И смотри, за курсором мышки остаются следы!

А вот так выглядит моя программа в самом Процессинге:

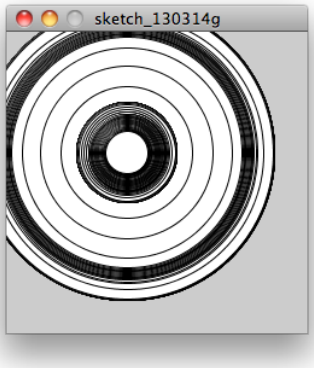

sketch_130306a | Processing 2.0b8
JAVA
sketch_130306a §
void setup() {
size(250, 250);
}
void draw() {
ellipse(mouseX, mouseY, 20, 20);
}
7

Яша балуется

Яша хитро прищурился и сказал:

— Ха, ну а раз вместо любого числа, я могу написать имя числового монстрика, и при выполнении команды он сначала будет заменен на число, которое помнит, а затем уже с этим числом исполнится команда, то я сделаю финт ушами, а точнее — финт кругами!

Смотри, что я придумал, я вместо ширины и высоты овала напишу имя монстрика-переменной `mouseX`! Тогда что у нас получится... — Задумался Яша, — чем ближе мышка будет к левому краю экрана, тем меньшее число будет помнить `mouseX`. А чем дальше — тем большее число он будет помнить. Ведь он же хранит расстояние от левого края.



И тогда получится, что чем ближе к левому краю, тем круги будут меньше, а чем ближе к правому, тем круги будут больше.

```
void setup() {
  size(250, 250);
}

void draw() {
  ellipse(100, 100, mouseX, mouseX);
}
```

— Во как странно и необычно получилось!

— Ну да, координаты-то у круга равны (`100`, `100`), а вот ширина меняется!

Хотелки

— Вот бы мне еще научиться стирать предыдущие круги, заливать экран каким-нибудь цветом! — замечтался Яша.

— Еще немного и я тебе про это тоже расскажу, — обнадежил его Интик.

Не запустилась?

Интик делится секретами:

— А если бы при запуске программы у тебя возникла ошибка, и программа не хочет запускаться, то прежде всего проверь еще раз текст программы, нет ли пропущенных точек с запятой, запятых, все ли слова написаны верно.

Опечататься это нормально, привычно и полезно для здоровья! Наверняка после проверки ты найдешь ошибку, и программа успешно запустится!

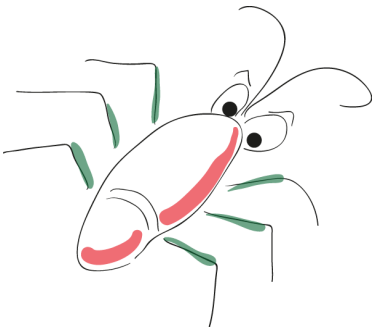
Но если вдруг все равно пишет, что ошибка, то пролистай в конец этой книги, там мы разберем типичные ошибки, это поможет разобраться с этой маленькой проблемой.

А если и это не помогло — пиши мне по адресу: **intiku@yandex.ru**

Или заходи на сайт: www.programmingforkids.ru

Программистские истории: ловля жуков

— А вообще, — начал рассказывать Интик, — когда у нас в программе создания Планеты возникает ошибка, мы говорим, что в нее забрался жучок, или баг (от английского «bug» — жук).



Есть такая история. Когда много-много лет назад на заре появления Создателей Планет, одна из первых программ запускалась, но в процессе работы все время выдавала ошибку. И разные умы умные и умы глупые пытались ее исправить, но ни у кого это не получалось.

Однажды рядом с громоздким аппаратом, который обсчитывает программы, маленький мальчик играя в машинки заигрался, и, открыв крышку вычислительного блока, увидел там спрятавшегося между контактов жучка, бага. Он его вытащил и с криками «я нашел баг, я нашел баг» побежал показывать его взрослым. В этот момент программа вдруг заработала и дальше работала без ошибок!

С тех пор программисты-волшебники сначала в шутку говорили, что «нашли баг», когда обнаруживали в своей программе ошибки, а затем история с мальчиком забылась, а вот это выражение прочно вошло в обиход.

Поэтому сейчас принято все ошибки делить на *очепятки* и на *баги*.

Очепятки — это те ошибки, от которых программа просто не запускается! Например, забыли фигурную скобку поставить, или пропустили точку с запятой, или вместо большой буквы написали маленькую, да много ли еще чего.

Эти ошибки отлавливать легче всего — о них обычно, как ты уже видел, сразу пишут в красном поле внизу, и подсвечивают строку, рядом с которой может быть ошибка. А может и не рядом, все-таки не всегда угадывают точно, поэтому самому все же думать хорошенько приходится порой.

А вот багами продолжают называть те ошибки, которые мешают работать программе так, как ты этого хочешь!

— Это как я забыл использовать монстриков `mouseX` и `mouseY`, и при этом все не понимал, почему моя программа не реагирует на мышку? — спросил Яша.

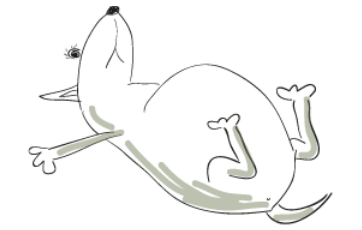
— Ага. И вот такие баги отлавливать сложнее всего. Потому что программа вроде и работает, но почему-то не так, как этого хочется, и приходится в ней детально разбираться, читать все строчки сверху вниз с самого начала, и пытаться догадаться, где же мог закрасться баг! Или она работает-работает, а затем бац, и ошибку выдает. Пойди, разберись еще почему...

Я тебе потом попозже расскажу способы, которыми мы пользуемся для ловли багов!

День второй. Пишем на стене

О функции text() и координатах «x», «y»

Функция «`text();`», читается «тэкст», переводится - «текст», чем-то похожа на функцию «`ellipse();`».



место, Бобик, место!

А похожа она тем, что, как и эллипс рисуется в определенном месте экрана, так и текст тоже выводится ни где-то вообще, а в четко указанном месте.

Передается при вызове три параметра.

Первым идет то, что мы хотим вывести на экран: фраза или число.

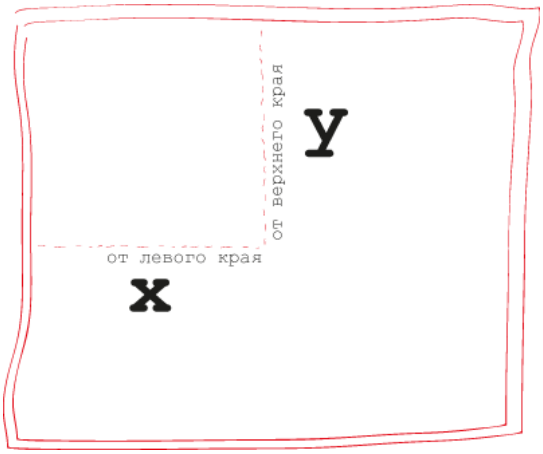
Вторым и третьим, как уже нам привычно, передается местоположение текста на экране (еще мы называем его координатами);

— Интик, местоположение — это ты имеешь ввиду пару чисел? Где первое — это расстояние от левого края экрана, а второе — от верхнего?

на языке уже завязло

— Совершенно верно! — подтвердил Интик. — Эта пара чисел нам будет постоянно встречаться, поэтому дальше я вместо длиннющей и вязнущей на языке фразы «расстояние от левого края экрана» буду говорить кратко: «`x`», что произносится, как «ИКС», а вместо фразы «расстояние от верхнего края экрана» буду говорить кратко: «`y`», что произносится, как «**И**ГРЕК». А эту пару чисел (`x, y`), то есть ИКС и ИГРЕК буду называть словом «координата».

— А, помню-помню, похоже, не только ты используешь эти сокращения - ИКС (`x`) и ИГРЕК (`y`), — заметил Яша.



В имени монстрика-аборигена, который помнит расстояние от мышки до левого края экрана как раз и зашифрован этот ИКС, то есть «`x`». Имя этого монстрика так и пишется: `mouseX`!

И с его другом такая же ситуация! В его имени, который помнит расстояние от мышки до верхнего края экрана, зашифрован ИГРЕК, то есть «`y`». И имя этого монстрика пишется: `mouseY`!

А что, удобно, мне кажется, я начинаю потихоньку осваиваться в именах!

— Тогда вот как запишем описание этой функции:

```
text(текстДляВывода, x, y);
```

где:
`текстДляВывода` — фраза (String) или число (int)
`x` — число
`y` — число

Вот, например, если мы хотим вывести фразу на экран "Привет, Яша!" в точку с координатами 20 на 50:

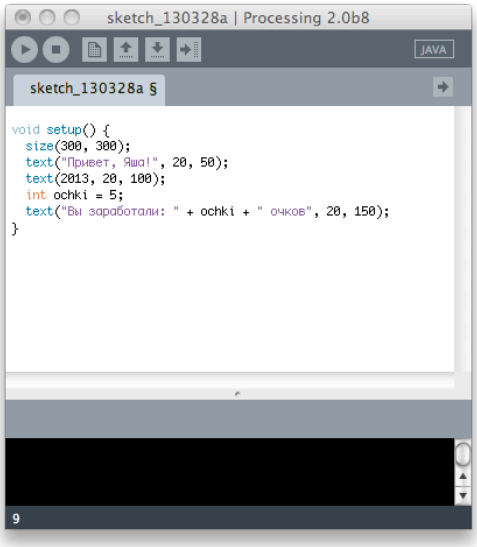
```
text("Привет, Яша!", 20,50);
```

А если хотим вывести, например число «2013» в то же место на экране:

```
text(2013, 20,100);
```

Пример вывода текста и числа, которое помнит переменная:

```
// Переменная ochki помнит количество очков в игре
  int ochki = 5;
  text("Вы заработали: " + ochki + " очков", 20,100);
```



О функции textSize()

— О, это очень простая функция. Она меняет размер выводимого текста! Ей в скобочках передается один-одинешенек параметр — число равное размеру текста.

```
textSize(размерТекста);
```

где:

`размерТекста` — число

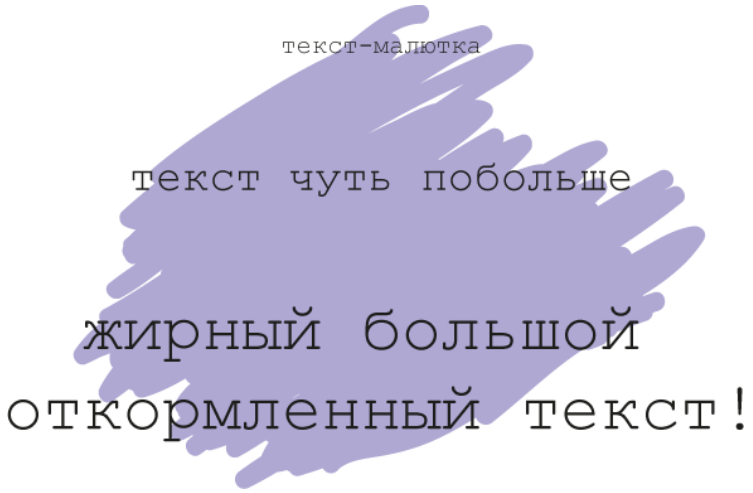
Например:

```
textSize(42);
```

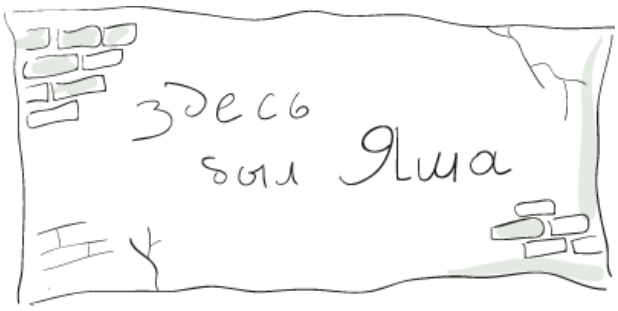
— Главное в этой функции что?

— Что?

— Вызвать ее до того, как вызовешь функцию `text()`! — пояснил Интик. — Так как она меняет размер только того текста, который будет выведен после ее вызова.



Яша пишет программу. Надписи на стене



— Ну ладно, — глубокомысленно изрек Яша и приступил к обдумыванию новой программы. — В этот раз я напишу совсем простенькую программу, пусть она выводит на экран какой-нибудь текст. Представлю-ка я себе, что это и не экран вовсе, а стена какого-нибудь дома! А что на стене написать? Ну, конечно, я напишу: "Здесь был Яша".

Для этого задам размер забора, то есть экрана... Надоели мне квадратные экраны, пусть будет вытянутый: 400 микробов на 200, например.

```
setup(){
  size(400, 200);
}
```

— Интик, я тут все же правильно написал?

— Ну, за исключением того, что ты опять забыл слово «void» — да, все правильно.

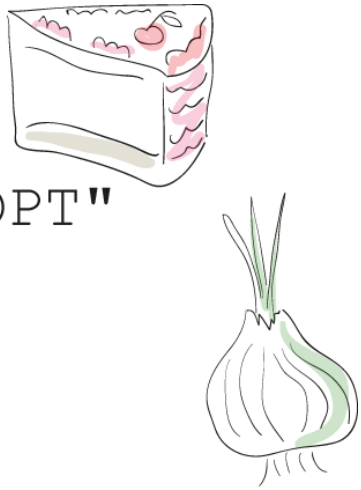
— Ой-ой, надо же, сейчас поправлю, а заодно допишу вывод текста, вот так:

```
void setup(){
  size(400, 200);
}

void draw() {
  text(«Здесь был Яша», 50, 100);
}
```

— Хм, оно не захотело запуститься, и выдало вот такую непонятнейшую ошибку! — огорченно сказал Яша, и потер лоб.

— Чего ему тут не понравилось? — спросил Яша.

— Кавычки, Яша, кавычки, — произнес Интик. — Если мы пытаемся передать этой функции текстовую строку в тех кавычках, которые принято писать в книгах, это вот в таких: «это текст в книжных кавычках», то Создатель Планет их не понимает! Ему нужны только наши особенные, прекраснейшие, вкуснейшие, ни с чем не сравнимые программистские кавычки, они вот такие: "а это текст в программистских кавычках".



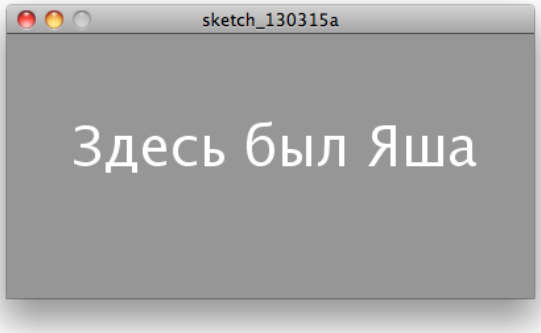
— Ох, а зачем такие сложности-то? — удивился Яша.

— У нас есть поговорка: «На вкус и цвет Создателя Планет нет!»

— Ладно-ладно, сейчас переделаю, а заодно и размер текста побольше укажу:

```
void setup(){
  size(400, 200);
}

void draw() {
  textSize(42);
  text("Здесь был Яша", 50, 100);
}
```

— Вот теперь — то, что надо:



Свои монстрики-аборигены

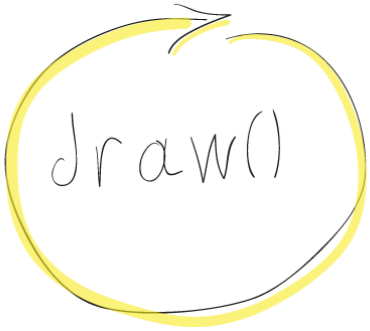
— Помнишь, что когда Создатель Планет выходит из территории `draw()`, то он все забывает, что в ней происходило. И все монстрики, которые в ней создавались, тоже растворяются в воздухе, как зимой пар изо рта. Или же превращаются в тыквы.

созданные в draw() монстрики-переменные
в полночь
превращались в тыквы...

— Помню, поэтому когда он снова заходит в территорию `draw()`, и видит команду о создании монстриков, он их с легкостью создает, и не возникает проблем с одинаковыми именами, так как все предыдущие все равно уже растворились! Или отыквились, — сказал Яша, — вот только это иногда бывает неудобно.

Если я, например, хочу знать, сколько раз Создатель Планет зашел в территорию `draw()`, то сейчас я не могу это узнать!

Монстрики-аборигены живут вне draw()



Так как даже если я создам монстрика-переменную из семейства чисел, и в самой территории `draw()` напишу приказ о том, чтобы монстрик увеличивал число, которое он помнит на единичку, то все равно, каждый раз, когда Создатель Планет заглянет в `draw()` он заново создаст этого монстрика, и тот забудет все о чем помнил раньше, бяда-то какая!

Яша продолжал:

— А единственные монстрики, которые не растворяются и не исчезают, это монстрики-аборигены, созданные за нас в самом начале! Но ведь к ним можно только обращаться по именам, спрашивая, что они помнят, а вот приказывать им изменить их значения — не-воз-мож-но!

— Так мы можем своих аборигенов насоздавать! — с гордостью сказал Интик.

— Ух ты, а как это? — удивился Яша.

— Для этого нам нужно написать приказ о создании монстриков-переменных не внутри территории `draw()`, как мы это раньше делали. А снаружи! Вне территории!

И тогда, когда Создатель Планет выйдет из территории `draw()` и забудет все, что он там набедокурил, то вообще же из программы он выйти не может, поэтому те монстрики-аборигены, которые созданы вне `draw()`, так и будут существовать!

— Мне бы как-нибудь на примере посмотреть, — попросил Яша.

— Вот, пожалуйста, создаем своего монстрика аборигена, например, по кличке ИНДЮК (`induk`), из семейства чисел (`int`):

```
int induk;

void setup(){
}

void draw(){
}
```

— Аа-а, теперь понятно, что ты имел ввиду, Интик, когда сказал, что вне территорий нужно разместить команды о создании монстриков.

Любая переменная исчезает, как только Создатель Планет выходит из той территории, где она создана.

Профессиональное создание аборигенов

РЕЖИМ ЧТЕНИЯ:
повышенная сложность

— А можно ли при создании монстриков-аборигенов сразу и присваивать им значения? — спросил Яша.
— Можно, это будет выглядеть так:

```
int induk = 0;

void setup(){
}
void draw(){
}
```

— А еще можно сразу создавать целые семейства монстриков-аборигенов, для этого после названия семейства нужно перечислить их имена через запятую, а в конце строчки завершить ее точкой с запятой:

```
int induk, kurica, jaba;
void setup(){
}
void draw(){
}
```

— А еще можно одновременно и создавать целые семейства и присваивать им значения:

```
int induk = 9, kurica = 2, jaba = 8;
void setup(){
}
void draw(){
}
```

Яша балуется — собирает отряд своих аборигенов

Шаг первый. Яша думает

— А раз я на собственном, так сказать, опыте понял, что Создатель Планет непрерывно посещает территорию `draw()`, то я бы хотел это использовать.

— И каким же, позвольте поинтересоваться, образом? — спросил Интик.

— Текст я уже умею выводить на экран, а теперь хочу, чтобы по-моему велению, по-моему хотению, он передвигался!

Яша стал рассуждать вслух:

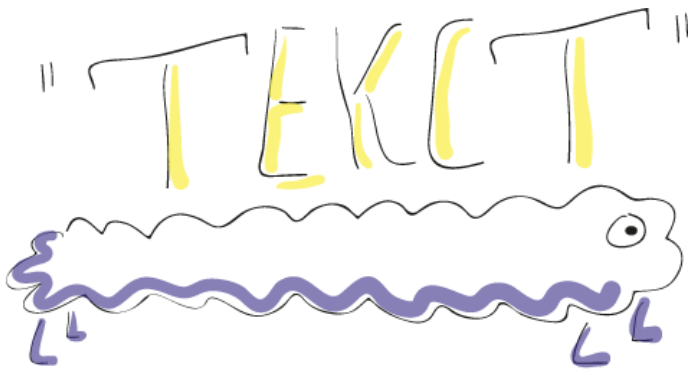
— А что такое передвижение? Это изменение положения на экране.

А как у нас определяется положение на экране? Ага, через координаты «x» и «y».

То есть сейчас мой текст выводится на расстоянии 50 пикселей от края экрана, а мне нужно, чтобы это число постоянно менялось.

А что у нас способно постоянно меняться? Ага, монстрики-переменные!

Значит, мне нужно завести ручного и пушистого монстрика-переменную, который и будет помнить положение текста на экране. Точнее, раз монстрик может помнить только одно число, значит, пусть он помнит расстояние от левого края экрана.



текст передвигается с помощью пушистого монстрика-аборигена

А расстояние от верхнего пусть пока для простоты будет 100 микробов, как и раньше.

Нареку монстрика именем... Раз он помнит координату «x» — где находится мой текст, то пусть и имя у него будет кратким «`xТекст`», или по-английски «`xText`».

```
int xText;
// ага, создали!
```

Что дальше-то?

А, ну дальше мы уже подобный финт ушами проделывали, когда координату «x» у овала указывали не числом, а писали имя монстрика.

Вот и сейчас, но только уже у текста, я вместо координаты «x» впишу имя монстрика, но только уже имя СВОЕГО МОНСТРИКА — `xText`! А не какого-то там лысого аборигена `mouseX`.

Ну да, ну да, а раз монстрики умеют превращаться, то когда Создатель Планет дойдет до приказа о выводе текста на экран, он спросит у моего монстрика, какое число он помнит, и это число подставит в вызов функции `text()`!

Итак, что у нас тут получается...

В предыдущей программе я писал так:

```
text("Здесь был Яша", 50, 100);
```

А сейчас я заменю «`50`» на `xText` и будет:

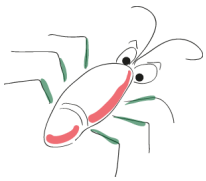
```
text("Здесь был Яша", xText, 100);
```

Интик, ты что-то долго молчишь! Я уже беспокоиться начал, ты не заснул?

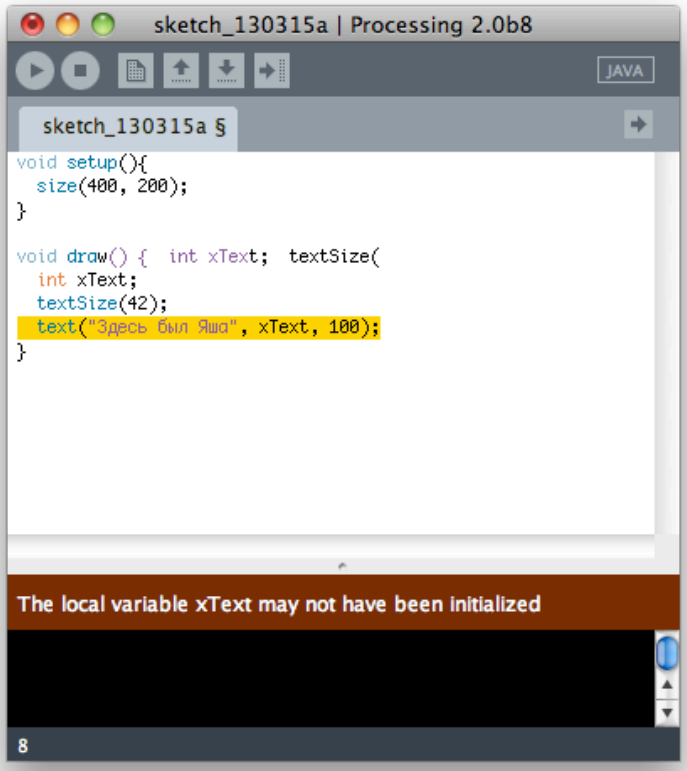
— Я слушаю, Яша, — ответил Интик. — Ты очень интересно и последовательно рассуждаешь. И пока все настолько правильно, что мне даже не к чему придраться! Что дальше?

— А дальше я тогда целиком код программы запишу:

```
void setup(){
  size(400, 200);
}

void draw() {
  int xText;
  textSize(42);
  text("Здесь был Яша", xText, 100);
}
```



Ну, запускаем...



— Вот это ошибка — всем ошибкам ошибка! — открыл рот от изумления Яша. — Интик, выручай, что здесь не так?

Поиск ошибки. Ловим багов

— А вот сейчас ты, Яшенька, поторопился! Давай разберем по-шагам.

В строчке: `int xText;` — создается переменная. Но заметь, что она еще ничего не помнит! Мы же ее не просили ничего запомнить!

новорожденная переменная еще ничего не помнит!

— А, кстати, Интик, у меня философский вопрос, что помнит переменная, когда ее только создали, но еще ничего ей не присвоили?

— Яша, в том-то и дело, что она ничего не помнит!

И поэтому, когда Создатель Планет видит ее имя при вызове функции `text("Здесь был Яша", xText, 100);` — он бы и рад был заменить эту переменную числом, которое она помнит. Но раз она ничего не помнит — он не может этого сделать. Вот и ругается, если дословно, то пишет, что «локальная переменная xText не инициализирована», «Зэ ло**о**кал вэриэбл эксТэкст мэй нот хэв бин инициалайзд», «The local variable xText may not have been initialized».

— Надо понимать, что «не инициализирована» это и имеется ввиду, что ничего не помнит.

— Ага, именно так.

— Ну тогда я ее про-ини-ци-а-ли-зи-ру-ю. Я могу присвоить ей, скажем число ноль сразу в момент создания. Или пока что сделаю это отдельной строкой, напишу «`xText присвоить 0`», то есть так: «`xText = 0`».

```
void setup(){
  size(400, 200);
}

void draw() {
  int xText;
  xText = 0;
  textSize(42);
  text("Здесь был Яша", xText, 100);
}
```

— Вот, теперь ошибки нет! Ура! — воскликнул Яша.

— Но и текст не двигается, — заметил Интик.

— Ничего, это дело поправимое! Сейчас займусь, — с оптимизмом ответил Яша.

Поиск логического бага

Яша стал рассуждать вслух дальше:

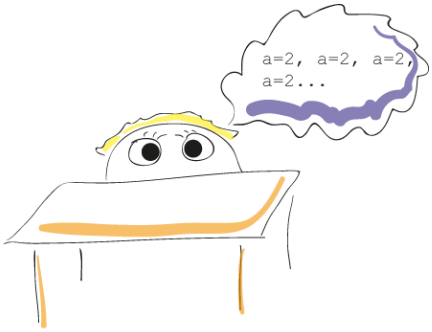
— Переменная у меня создана — это раз.

Начальное значение ноль ей присвоено — это два.

Даже превращение переменной в число замечательно проходит — это три.

Чего не хватает?

А не хватает изменения переменной! Что она все ноль, да ноль.



переменная "a"
зазубривает новое значение

Эх, добавлю изменений!

Как там это делается?

Пишем «`xText =`» — что означает `xText` запомнит значение справа от знака «=». А что бы написать справа от этого знака? — почесал затылок Яша.

Думай голова, думай. Число на единицу большее, чем то число, которое помнит xText получить проще простого: `xText + 1`.

Ага, вот и правая часть готова.

Напишу так:

```
xText = xText + 1;
```

и добавлю в программу сразу после вывода на экран:

```
void setup(){
  size(400, 200);
}

void draw() {
  int xText;
  xText = 0;
  textSize(42);
  text("Здесь был Яша", xText, 100);
  xText = xText + 1;
}
```

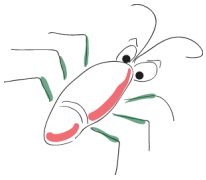
— Интик, а у меня текст все равно не двигается!

И еще один логический баг!

— Так конечно, ты наверно просто запамятовал, что выходя из территории `draw()` Создатель Планет все забывает, что творил в ней. А когда он в нее заходит — то заново создает твою переменную `xText`, и опять ей присваивает `0`! Вот у тебя ничего и не меняется, — пояснил Интик. — Моя тебе подсказка: вспомни про создание своих монстриков-аборигенов!

— Вспоминаю-вспоминаю. Ты же мне совсем недавно это рассказывал. Если я хочу, чтобы переменные не терялись, то нужно сделать их аборигенами, то есть выселить из территории `draw()` наружу! Чем я сейчас и займусь:

```
int xText;
xText = 0;

void setup(){
  size(400, 200);
}

void draw() {
  textSize(42);
  text("Здесь был Яша", xText, 100);
  xText = xText + 1;
}
```

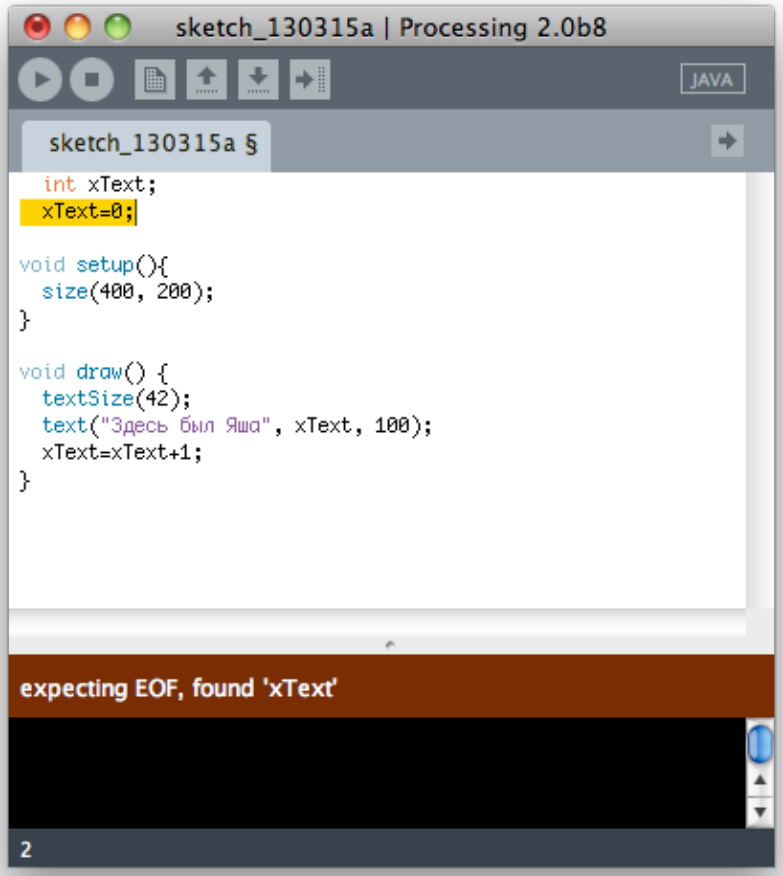


— Интик, я вроде все правильно написал, но у меня снова ошибка.

— Сейчас мы ее поборем! — с воодушевлением ответил Интик. — Давай ее сюда!

И еще одна очепятка

— Вот, сказал Яша.



— Дай-ка я прочитаю, сказал Интик. — Пишет: «экспэктинг ЭОФ, фаунд `эксТэкст`», «ожидаю конец файла, а нашел пременную `xText`», «`expecting EOF, found `xText``»

— Конец файла? — недоумевал Яша.

— Вот, я же говорил, что не всегда он верно ошибку определяет. Но я-то знаю, что не так, и расскажу тебе:

Вне территорий `setup()` и `draw()` могут только создаваться монстрики-переменные. А вот какие-то команды там писать нельзя. Скажем вызывать функции, или присваивать монстрикам какие-нибудь значения.

За одним исключением, монстрику можно присвоить значение только в момент его создания, помнишь, как это делать?

— Ага, — подтвердил Яша. — Вот так: `int xText=0;`

— Верно, поэтому исправить эту ошибку можно двумя способами...

— Или написать так, как я только что написал, присвоив значение переменной сразу при создании.

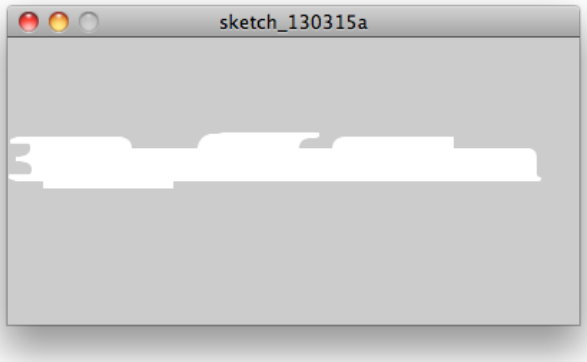
Или перенести присвоение в одну из территорий.

В территорию `draw()` переносить глупо, так как ей каждый круг будет ноль присваиваться, а вот `setup()`, как мне кажется, для этого идеально подходит.

Получится такая программа:

```
int xText;

void setup(){
  size(400, 200);
  xText=0;
}

void draw() {
  textSize(42);
  text("Здесь был Яша", xText, 100);
  xText=xText+1;
}
```

— Ура, двигается, только размазывается почему-то!



— Так она же каждый раз рисует поверх предыдущего рисунка! — воскликнул Интик.

— Тогда мне нужно добавить функцию стирания предыдущего рисунка! Залить бы всю стену краской, перед тем как новую надпись выводить!

— Это, кстати, запросто, для этого тебе пригодится следующая функция: `background();` Я про нее прямо сейчас и расскажу.

Функция: background() или ведро с краской

По-русски произносится, как «бэкграунд», а означает «фон», имея ввиду цвет фона нашего окна.

Вызывая эту функцию, мы все равно, что ведро с краской опрокидываем на наш экран, все что было, становится погребено под новым блестящим слоем краски.

background();
не скупится на краску

При вызове мы передаем ей в скобках число от 0 до 255. Число 0 означает ноль света, то есть темноту, черный цвет. А 255 — максимум света, то есть белый цвет. Можно передать и любые числа от 0 до 255 — это будет означать серый цвет.

— А больше 255 нельзя число задать?

— Нет, так как 255 — это и есть белый цвет, и нельзя сделать его еще белее.

— И все-таки, странное число, почему не 300, оно хоть круглое? — поинтересовался Яша.

— А у тебя дома, сколько фломастеров в коробке? — в ответ спросил Интик.

— Ну, 15, — ответил Яша.

— Вот и у нас серых цветов всего 255. Если бы было, скажем, пять серых цветов, то мы бы задавали число от 0 до 5. А так до 255 задаем.

— Ну, теперь понятно, — произнес Яша.

```
background(число_от_0_до_255);
```

Например, установим светло-серый цвет:

```
background(100);
```

Или что то же самое:

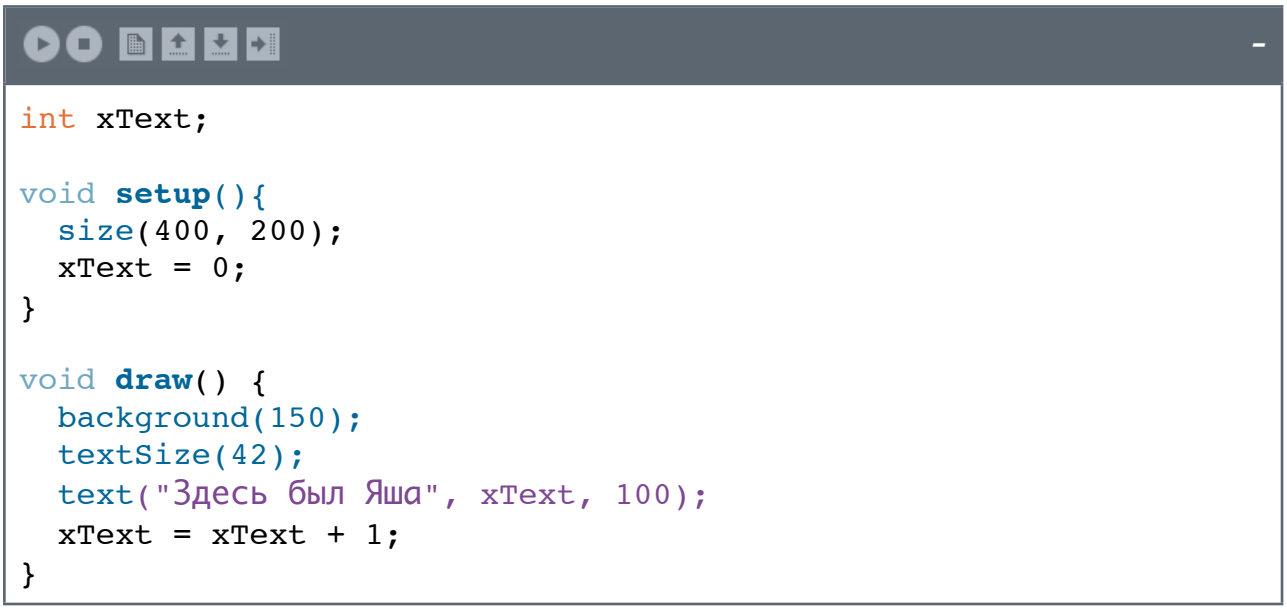
```
int c = 100;
background(c);
```

Яша учится пользоваться ведром с краской

— А сейчас я добавлю функцию стирания предыдущего рисунка! И мне кажется, что на эту роль отлично сойдет функция ведро краски, то есть `background();`

Как там она вызывается, — Яша подсмотрел в шпаргалку. — Ага, в скобках ей передается номер серого цвета.

Вот что у меня получится:



```
int xText;

void setup(){
  size(400, 200);
  xText = 0;
}

void draw() {
  background(150);
  textSize(42);
  text("Здесь был Яша", xText, 100);
  xText = xText + 1;
}
```

— Ура! Он двигается! Ура!

РЕЖИМ ЧТЕНИЯ:
повышенная сложность

— Кстати, хотел напомнить, — заметил Интик, — что следующую строку настоящие профессионалы пишут короче:

```
xText = xText + 1;
```

— И как ее записать по-другому? — спросил Яша.
— Вот так:

```
xText += 1;
```

— Что и означает, пусть xText увеличит свое значение на единицу.
— А вместо цифры «1» наверно можно написать, какое угодно другое число? — спросил Яша.
— Именно! И не только число, но и другую числовую переменную. Вот например, чему будет равен `xText`, если


```
int a = 5;
int xText = 10;
int xText += a;
```

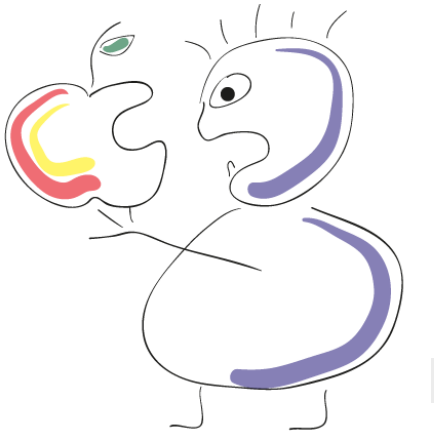
— Ну раз «+=» означает увеличить значение переменной `xText` на значение справа от этого знака, а справа у нас стоит переменная `a`. Значит `xText` увеличит свое значение на то, которое помнит a, то есть на `5`.
Значит `xText` станет равен `15`!
— Верно, все на лету схватываешь! — похвалил Интик.

Немного попозже Интик еще добавил:

— А есть еще совсем продвинутая запись!

— Расскажи? — попросил Яша.

— Она пригождается только в одном случае, когда мы хотим, чтобы переменная увеличила свое значение ровно на единицу. В этом случае мы пишем так:



профессионал уменьшает размер яблока путем его поедания

```
xText++;
```

— А если мы хотим, чтобы она уменьшила свое значение ровно на единицу, то пишем подряд два минусика:

```
xText--;
```

— Ха, — заявил Яша. — Я тогда свою программу тоже подправлю, пусть выглядит более профессиональной!

Первое, что я сделаю, это использую твой, Интик, новый способ увеличения переменной на единицу, а второе — присвою переменной ноль сразу при создании:

```
int xText = 0;

void setup(){
  size(400, 200);
}

void draw() {
  background(150);
  textSize(42);
  text("Здесь был Яша", xText, 100);
  xText++;
}
```

— Ничего так, симпатичненько получилось, — согласился Интик.

Хотелки

— Вот только мой текст уползает и не возвращается. Было бы хорошо, если бы он не уезжал вправо навсегда! Интик, как его вернуть на место?

— Яша, это сделать очень просто, для этого... Хотя, давай сейчас отдохнем, а потом я тебе объясню. Расскажу про новую территорию-контроллера: `if()`, которую называют «иф()», что в переводе «если».

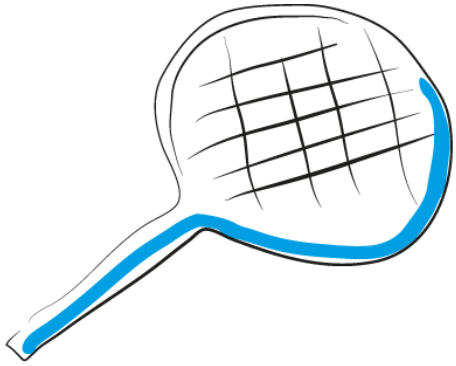
— Эх, мне бы еще цветами текст раскрасить, а то устал я уже от этих серых будней, — пожаловался Яша.

— Ничего-ничего, потерпи еще немного, через пару глав дойдем до цвета, вот уж тогда нараскрашиваемся!

О функциях-мухобойках: print() и println()

— А еще я тебе расскажу про одни из лучших функций, применяемых при отлове багов в программе, жучков, ошибок то есть, — начал рассказывать Интик. — Мы их между собой так и кличим иногда — мухобойками!

— Я их буду называть багобойками! Так и чем же они такие особенные? — спросил Яша.



print(); - лучшее средство для ловли багов

— Просто они у нас выводят текст не на экран, а в место для шпаргалок!

— А зачем нам шпаргалки? — удивился Яша.

— Ну смотри, пишем мы, пишем программу. А в ней все ошибка какая-то непонятная. И никак эту ошибочку, этого жучка выловить не можем.

И вот мы думаем, а вдруг он у нас зарылся в неправильных значениях монстриков-переменных. Вдруг они помнят совсем не то, что мы от них ждем. Что же делать?

— Ну... — протянул Яша, — в этом случае хорошо бы опросить этих монстриков сразу походу программы. Пусть программа выполняется, как и раньше, а мы бы им допрос с пристрастием учинили! Да так, чтобы на ход самой программы влияния — ни-ни!

print("ГДЕ ПРЯНИК?");

— Верно мыслишь, товарищ Яша! — улыбнулся Интик. — Вот функция `print();` такой допрос и устраивает.

Читается, кстати, как «принт», а переводится «напечатать».

А если по-простому говорить, то работает совершенно так же, как и функция `text();` вот только еще проще! Ей не нужно передавать координаты x и y.

— Она что ли сама знает, куда ей выводить? — спросил Яша.

— Что-то вроде того. Она выводит текст не на экран, а в шпаргалку внизу экрана с нашей программой. Она еще такого черного цвета. Я тебе покажу попозже.

— А `println();` тогда чем отличается от `print();`? — поинтересовался Яша.

— Это сокращение от print line, что означает «напечатать строку». Произносится `println();`, кстати, как «пр*и*нтэлэн».

Сокращения они такие — трудновыговариваемые.

Отличается тем, что если `print();` после вывода на экран, например, фразы "не хочу", как будто запоминает то место, где остановился вывод, и если еще раз вызвать `print();` с фразой "манную кашу", так она эту манную кашу напишет сразу после слов "не хочу".

А функция `println();` каждый свой вывод текста на экран начинает с новой строки!

```
print(текстДляВывода);
```

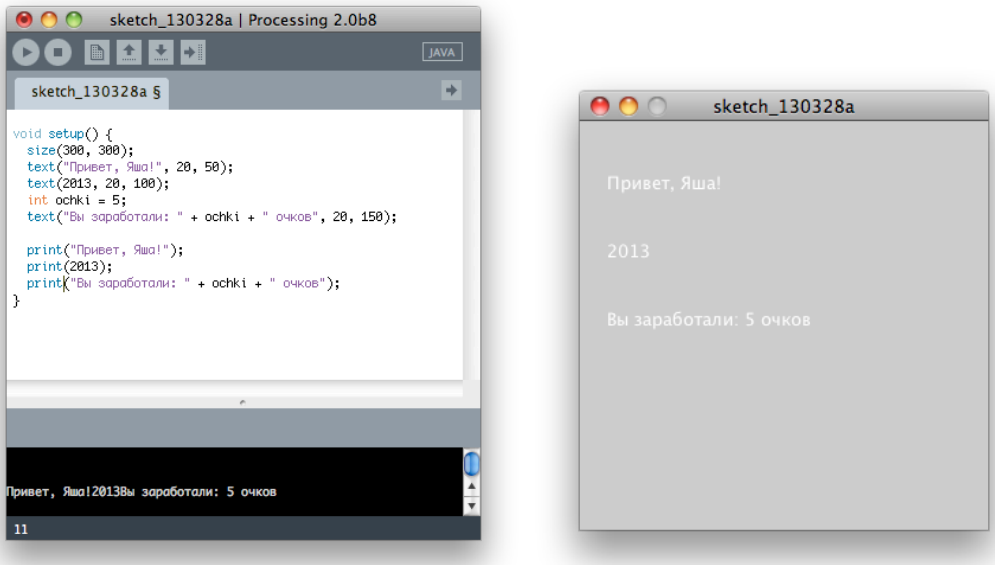
где:
`текстДляВывода` — фраза или число

Например, команды:

```
print(" не хочу ");
print("манную кашу");
```

Выведут в область подсказок все в одной строчке:

```
не хочу манную кашу
```




```
println(текстДляВывода);
```

где:

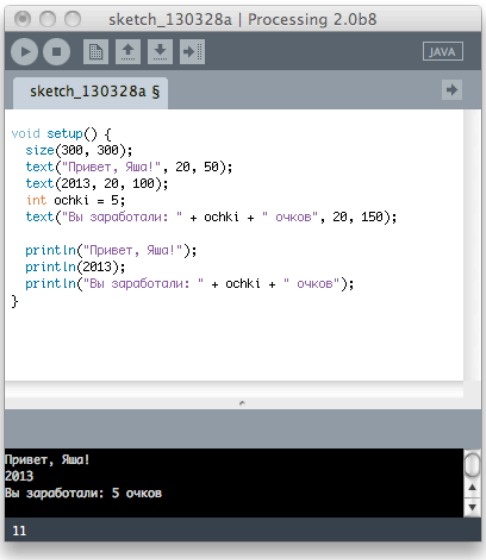
`текстДляВывода` — фраза или число

Например команды:

```
println(" не хочу ");
println("манную кашу");
```

Выведут в область подсказок каждую фразу в отдельной строчке:

```
 не хочу
манную кашу
```



Яша балуется

— А ну-ка, — сказал Яша, прищурившись, — добавлю-ка я после `text();` вызов еще и `print();` — вот и посмотрим, что это за зверь такой. — Помню-помню, — Яша не дал даже рот раскрыть Интику, — что `text();` я вызываю с координатами, то есть с «x» и «y», а `print();` без оных!

Сказано, сделано:

```
void setup(){
  size(400, 200);
}

void draw() {
  textSize(42);
  text("Здесь был Яша", 50, 100);
  print("Здесь был Яша");
}
```

— Вот оно, как интересно, — озадачился Яша. — Окно программы у меня как было, так и осталось. А вот внизу самого текста программы, на черном фоне, словно галки затараторили: "Здесь был ЯшаЗдесь был ЯшаЗдесь...".

И почему-то все тараторят и тараторят.

— Ну так, Яша, — заметил Интик, — а ты куда вызов функции `print();` поставил?

— Куда-куда, в территорию `draw()` и поставил, — не понял вопроса Яша.

— А помнишь, чем эта территория особенная? — спросил Интик.

— А-а, — догадался Яша, — Создатель Планет постоянно в ней вертится, входит-выходит, входит-выходит. Эх, как говорил Иа, замечательно выходит!

— Иа?

— Осел такой. Ну да не важно. Главное — я понял. Создатель Планет вызывает функцию `print()` она для него печатает текст внизу на черном фоне, он выходит из территории и снова в нее заходит, снова видит функцию `print()` снова ее вызывает. И она снова ему печатает текст. Что мы и видим! — произнес Яша, довольный собой.

Создатель Планет играет с функцией draw()

Яша пишет песенку

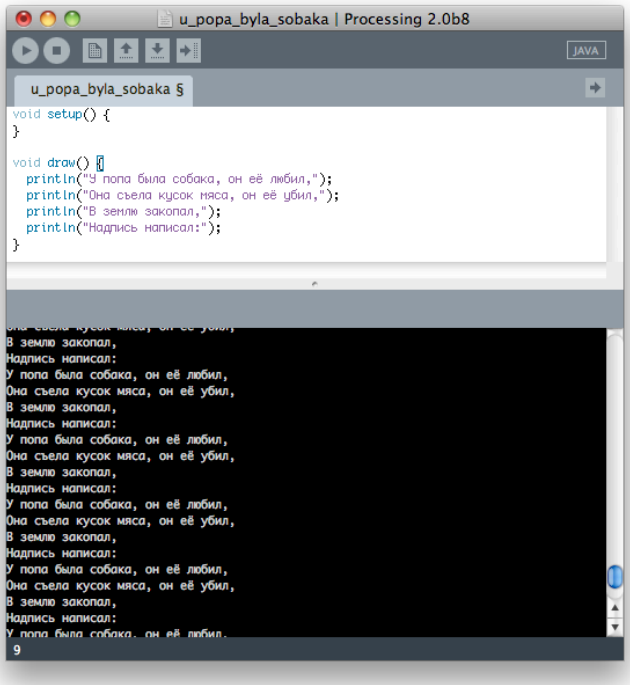
— О, — вдруг воскликнул Яша, — я теперь знаю, как правильно кое-что записать. Вот у меня песенка состоит из повторяющихся стишков. Я их и выведу в место для подсказок функцией `println();`

```
println("У попа была собака, он её любил,");
println("Она съела кусок мяса, он её убил,");
println("В землю закопал,");
println("Надпись написал:");
```

Получится такая простая программка:

```
void setup() {
}

void draw() {
  println("У попа была собака, он её любил,");
  println("Она съела кусок мяса, он её убил,");
  println("В землю закопал,");
  println("Надпись написал:");
}
```

И смешно так пишет:



Яша улучшает песенку

— Но это очень просто было. Теперь я хочу сделать так, чтобы каждое следующее четверостишие было немного сдвинуто вправо.

А как мне сдвинуть текст вправо?

— А есть же пробел, который вот так пишется: `" "` , — подсказал Интик. — Просто кавычки и пустое место между ними.

— Да! Но я хочу, чтобы каждое следующее было сдвинуто все правее и правее!

— Тогда это похоже на то, как ты текст раньше двигал с помощью переменных! — опять подсказал Интик.

— Точно, только здесь вместо координат у меня будет переменная из семейства `String`, которая будет хранить только пробелы!

И для начала, она не будет хранить ни одного пробела, чтобы не было лишнего отступа при первом запуске, вот так:

```
String s = "";
```

А в конце программы, я ей буду прибавлять по одному пробелу:

```
  s = s + " ";
```

Получится, что сначала она помнит `""`, затем `" "`, затем `"  "`, и так далее...

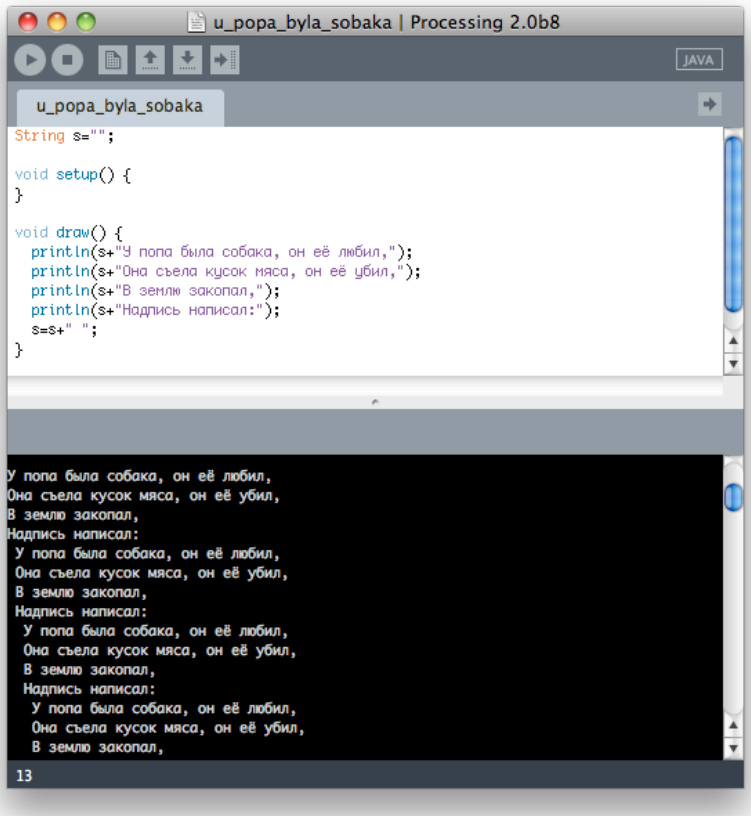
А чтобы текст сдвигался вправо, я в `println();` напишу сложение этой `s` и фразы из стишка:

```
  println(s + "У попа была собака, он её любил,");
  println(s + "Она съела кусок мяса, он её убил,");
  println(s + "В землю закопал,");
  println(s + "Надпись написал:");
```

Все вместе у меня вот так получилось:

```
String s="";

void setup() {
}

void draw() {
  println(s + "У попа была собака, он её любил,");
  println(s + "Она съела кусок мяса, он её убил,");
  println(s + "В землю закопал,");
  println(s + "Надпись написал:");
  s = s + " ";
}
```

Правда, классная бесконечная песенка?



День третий. В гостях у контроллеров

Контроллеры if(). Билетики есть? Проходите!

— Есть территории, в которые попасть можно только при выполнении каких-то условий. Все такие территории имеют одно универсальное имя «if», читается «иф», в переводе «если».

Устроена территория как обычно, сначала идет ее имя `if`, затем калитка в виде круглых скобочек, а за круглыми скобочками забор из фигурных скобочек.

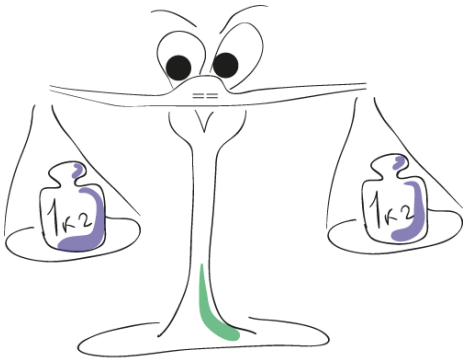
Вот только в главном входе, калитке, указывается условие, при выполнении которого пустят на саму территорию.

Сказочный пример:

```
если (победилДракона == этоПравда) {
 получи принцессу в жены;
 получи полцарства в приданное;
 стань королем;
}
```

— Интик, а что это за два знака «равно» подряд в круглых скобочках? — спросил Яша.

— Это, Яша, оператор сравнения! — ответил Интик, — он сравнивает то, что стоит слева от него и справа от него. Переводится как «равно ли».



оператор сравнения "==" дотошно сравнивает равен ли вес гирь слева и справа

— А у нас в школе на уроке математики знак «равно» обозначается так: «=», и не повторяется два раза! — удивился Яша.

— А у нас знак «=», когда он один, означает не просто «равно», а «пусть станет равным». Помнишь, как мы присваивали переменным значения?

```
int limon;
limon = 10; // означает: limon пусть станет равным 10.
```

А вот два «равно» подряд — это сравнение, и используется только контроллерами на территории `if()`! Например:

```
int limon;
limon = 10;
if (limon == 10) { /* означает: если (limon и правда равен 10) */
    /* если и правда равен, тогда выполняются команды в этом загончике; */
}
```

— То есть знак «==» не заставляет монстрика запомнить новое число? — удивился Яша.

— Совершенно верно! Знак «==» и контроллеры `if()` — просто наблюдатели, и ни во что не вмешиваются.

контроллер пустит в зоопарк
только
если (уТебяДенег == 100)

Хитрые сравнения у контроллеров

— Интик, а контроллеры могут только сравнивать монстрика-переменную и число, на предмет равны ли они, и больше ничего не могут? — спросил Яша.

— Ну как же ничего! Они еще ого-го, как много всего могут. Сейчас расскажу.

Кроме сравнения “равно ли”, то есть значка «==», применяются и другие сравнения. Смотри какие:

```
>    БОЛЬШЕ

<    МЕНЬШЕ

>=   БОЛЬШЕ ЛИБО РАВЕН

<=   МЕНЬШЕ ЛИБО РАВЕН

==   РАВЕН

!=   НЕ РАВЕН
```

Вот например:

```
// limon БОЛЬШЕ 10?
limon>10

// limon МЕНЬШЕ 10?
limon<10

// limon БОЛЬШЕ ЛИБО РАВЕН 10?
limon>=10

// limon МЕНЬШЕ ЛИБО РАВЕН 10?
limon<=10

// limon РАВЕН 10?
limon==10

// limon НЕ РАВЕН 10?
limon!=10
```

Вот все возможные варианты. Посмотри внимательно, они обязательно пригодятся в будущем.

— Интик, только они у тебя в примере без круглых скобочек написаны... — заметил Яша.

— Верно, это я для примера, так написал. А когда ты их будешь использовать с контроллерами, то обязательно в круглых скобочках размести, это ты правильно заметил! Вот так:

```
if (limon > 10) {
   /* а здесь размести команды всякие */
}
```

Подсказка

Как набрать на клавиатуре знак больше «>» или знак «<».

Обычно эти клавиши расположены в правом нижнем углу клавиатуры, там, где находятся русские буквы «Б» и «Ю».

Когда включен режим ввода английского текста нажми «Shift» и одну из этих клавиш.

Как набрать на клавиатуре знак равенства «=» или восклицательный знак «!».

Обычно эти клавиши расположены в ряду с цифрами.

Знак «!» получается, если нажать и удерживать клавишу «Shift», после чего нажать клавишу с цифрой один «1».

А знак «=» получается, если нажать клавишу «=».

Интик рассказывает, зачем нужны хитрые сравнения

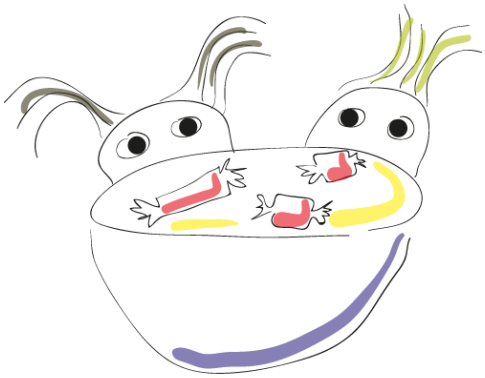
— А зачем они нам могут пригодиться? — спросил Яша.

— Как же зачем? Вот, например, предположим, что для запоминания количества шоколадных плиток, которые съел Саша, мы уже создали монстрика ШОК (shok):

```
if (shok > 2) {
  // больше шоколада Саше не давать!
}
```

Или, предположим, что для запоминания количества конфет, которые дали Ане, мы уже создали монстрика КОНФЕТА1 (konfeta1), а для запоминания количества конфет, которые дали Ясе, мы уже создали монстрика КОНФЕТА2 (konfeta2).

А мы, скажем, хотим дать детям конфеты поровну, то можно написать так:



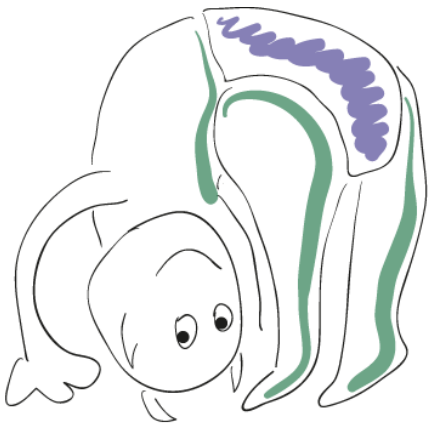
Аня и Яся делят конфеты поровну

```
if (konfeta1 < konfeta2) {
  // Дать Ане еще одну конфету!
}
```

— О, так можно сравнивать не только переменные с числами, но еще и их самих между собой? — удивился Яша.

— Ага, - подтвердил Интик.

— Это может оказаться удобным, - задумался Яша. - Со знаками «<» (меньше) и «>» (больше) менее-более понятно, а вот для чего могут пригодиться такие необычные знаки сравнения, как «<=» (меньше либо равно) и «>=» (больше либо равно)?



монстрик КУВЫРОК хочет сделать >= 12 кувырков

— А вот, например, тренер по спортивному кувырканию дает задание: «сделать не меньше 12 кувырков». А «не меньше 12» означает, что можно сделать и 12, а можно и 13, и 128! Только 128 лучше не надо, а то ходить потом не сможешь, если ты не чемпион олимпиады по кувырканию.

— То есть, получается, здесь пригодится знак «больше или равно 12»?

— Ага, вот смотри, предположим, что для запоминания количества кувырков мы уже создали монстрика КУВЫРОК (`kuvyrok`), потом покувыркались, а теперь хотим запустить нашего КУВЫРКА к контроллеру:

```
if (kuvyrok >= 12) {
  // сказать тренеру, что задание сделано!
}
```

— А можно ли сравнить разность двух переменных с каким-то числом? — спросил Яша.

— Конечно можно! — ответил Интик. — Ведь как справа, так и слева от операторов сравнения можно написать любые математические выражения! Вот, например, можно сравнить разность переменных `a` и `b` с числом 20:

```
if (a-b > 20) {
  // a минус b больше 20
}
```

— Отличненько, а давай в контроллеров поиграем! — попросил Яша.

— Давай, а как?! — спросил Интик.

— Ну, например, я хочу, чтобы, если мышка находилась бы в правой половине экрана, то рисовать большие круги. А если в левой, то маленькие. И что-то мне подсказывает, что без привлечения контроллеров я не обойдусь, — задумался Яша.

— Да, тебе понадобится контроллер, который будет пускать на территорию, где написан приказ рисовать большие круги, только если расстояние мышки от левого края экрана будет больше некоторого значения, — стал размышлять Интик.

— Я помню, что расстояние мышки от левого края экрана помнит монстрик-абориген `mouseX`. Значит, у меня получится следующий код:

```
if (mouseX => 100) {
   // здесь я помещу приказ о рисовании большого круга
}

if (mouseX =< 100) {
   // здесь я помещу приказ о рисовании маленького круга
}
```

— Получилось?

— Не совсем, ты немного перепутал значки и написал «=>» вместо «>=»! А еще написал «=<» вместо «<=», — поправил Яшу Интик.

— Тогда так?

```
if (mouseX >= 100) {
   // здесь я помещу приказ о рисовании большого круга
}

if (mouseX <= 100) {
   // здесь я помещу приказ о рисовании маленького круга
}
```

— Ага, теперь замечательно!

Яша пишет программу. Посадка на луну

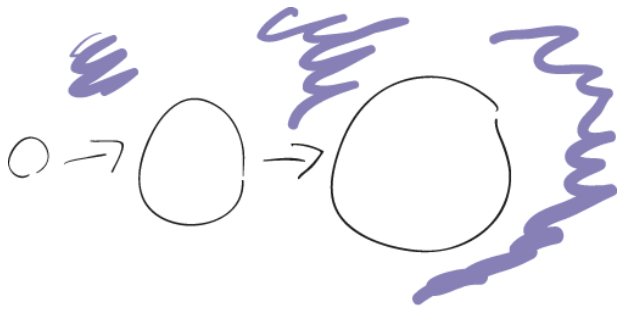
— А вот напишу-ка я программу посадки на Луну! — с воодушевлением размечтался Яша.

— На какую еще луну? — недоуменно спросил Интик.

— Как это еще на какую? На круглую! Вон смотри, — ответил Яша, показывая пальцем в небо, — какая у вас Луна красивая.

Когда она так далеко — она маленькой кажется, а вот если представить, что мы на нее прилуняемся, то она все увеличивается и увеличивается.

Вот, — сказал Яша, — выводя на листке бумаги картинки, так должна выглядеть Луна.



Так должна выглядеть луна,
когда на нее прилуняешься

Вообще моя программа будет очень похожа на ту, где текст двигался.

Сначала я создам своего монстрика-аборигена под именем i, он будет помнить, насколько мы близки уже к Луне. И сразу присвою ему значение 0.

```
int i = 0;
```

А дальше все просто, только вместо вывода текста, я использую функцию `ellipse()`!

Раз экран сделаю размером 400 на 400 пикселей, то для того, чтобы круг был в центре, укажу ему координаты: 200 на 200.

Начальный размер круга сделаю 20 по ширине и 20 по высоте — Луна, она же круглая!

А еще, а еще вместо длинной записи:

```
  i = i + 1;
```

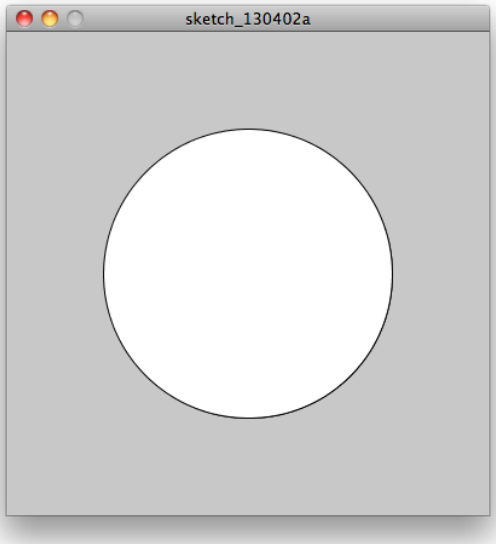
Я напишу короткую, как узнал недавно:

```
  i++;
```

И Яша быстренько написал следующую программу:

```
int i = 0;

void setup() {
  size(400, 400);
}

void draw() {
  ellipse(200, 200, 20 + i, 20 + i);
  i++;
}
```

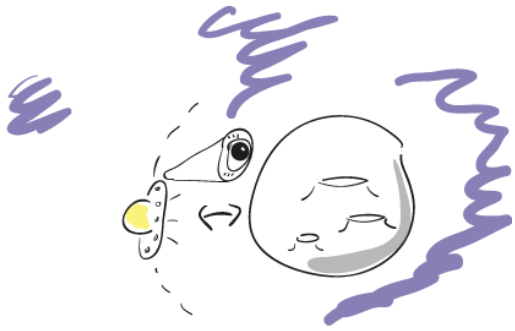
— Вот что у меня получилось! Интик, правда, похоже на прилунение?



— Да, очень похоже, один в один, — согласился Интик.

Яша улучшает программу. Остановка на орбите

— А теперь, бффф, мой корабль отменяет прилунение, и должен зависнуть на орбите неизведанной Луны, чтобы исследовать ее издалека! Вдруг там опасности! — придумывал дальше Яша.



Космический корабль завис на орбите, высматривая возможного врага

— Вот тут тебе и понадобятся контроллеры, о которых мы недавно говорили, — заметил Интик. — Тебе же надо как-то узнать, когда прекращать приближение?

— Угу, — заявил Яша. — Надо. Вот скажем, я хочу, чтобы мы остановились тогда, когда размер Луны стал бы равен 300.

— Другими словами, — переформулировал Интик, — ты хочешь, чтобы мы приближались к Луне все то время, пока ее размер меньше 300?

— Ага, — согласился Яша.

— А как бы ты записал эти два условия в зашифрованном виде? В виде, готовом для размещения в круглых скобочках у контроллера? — спросил Интик.

— Ну, — задумался Яша, — если размер Луны у меня хранит переменная i, а мне нужно остановиться, если он станет равным 300, то я запишу так: `(i = 300)`. Все правильно?

— Почти что, — с паузой произнес Интик. — Вот только одна мелочь, ты присваивание перепутал со сравнением.

— Ах, да, — хлопнул себя по лбу Яша, — одно равно «=» — это же приказ, чтобы монстрик запомнил число 300. А для сравнения нужно написать «==». Вот так: `(i == 300)`. А теперь?

— Вот теперь верно. А второе условие?

— Со вторым все ясно. Размер меньше 300, это значит нужно написать так: `(i < 300)`.

— Ага. А что внутри территории у контроллера напишешь?

— Ну раз мы должны приближаться, пока i < 300, а за приближение у нас отвечает команда i++, то ее и размещу, вот так:

```
if (i < 300) {
  i++;
}
```

А вся моя программа теперь будет выглядеть так:

```
int i = 0;

void setup() {
  size(400, 400);
}

void draw() {
  ellipse(200, 200, 20 + i, 20 + i);
  if (i < 300) {
    i++;
  }
}
```

— Ур-ра! Ррработает! — воскликнул Яша. — Я обожаю контроллеров! С ними так просто писать программы!

Хотелки и объектное программирование

— А вот что, если я бы хотел много лун сделать? Вот прямо много-много? — спросил Яша. — Это ведь наверно ужасно сложно, столько раз придется писать одно и то же?

— О, нет, Яша! Есть такая классная штука, которая называется ОБЪЕКТО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ. Если сокращенно, то ООП.

— Заумные какие-то слова, — прокоментировал Яша непонятную фразу.

— Да все просто, на самом деле, — с воодушевлением произнес Интик. — Это обычная программа, с небольшой, но ужасно нам полезной особенностью. Она позволяет не писать сто раз одинаковые команды. Специальными заклинаниями мы один раз создаем луну, звезду, монстрика, или корабль, а затем можем их размножать. Например, сделать так, чтобы у нас был целый космос звезд, или целый флот кораблей. А программа при этом все равно остается махонькой!

— Хочу-хочу! — Яша потянул вверх руку, прямо, как в школе на уроке. — когда же ты мне про ООП расскажешь?

— Обязательно расскажу, потерпи чуток!

О функции frameRate(), встреча с Создателем Планет

— Яша, а ты знаешь, что мы можем сделать так, что наша программа будет выполняться или медленно, или быстро? — спросил Интик.

— Подгонять старичка Создателя Планет? — улыбнулся Яша.

— Вроде того.

Внезапно рядом с ними появилось облачко, из которого сначала показалась остроконечная шляпа, затем сухонькое сморщенное личико, а затем пред ними предстал сам Создатель Планет.

— Вот не надоть так выражаться, молодой люди! — заявил он, слегка искажая слова. — Интий Флоатович, вы ведь есть прекрасно знает, что мне положен норматив!

"Я есть наблюдать за вами"

— Уважаемый Создатель Планет, — произнес Яша, — вы уж извините нас, мы не нарочно! Я здесь первый раз и еще ничего-ничего не понимаю, можете мне помочь?

— Ладно-ладно, — проворчал старичок, — молодежь, в чем помочь-то?

— Помочь понять, как и что здесь устроено! Вот, например, что это за норматив, о котором вы сейчас сказали? — вежливо поинтересовался Яша.

— Это есть скорость, с которой мне положен перемещаться, когда я есть создавай планет.

— Вот не совсем понятно...

— Чего же есть тут не понятно? Вы и так знать, что территория `draw()` напоминать зацикленный беговой дорожка, и выходя из `draw()` я есть снова в нее заходить!

— Знаем-знаем... — подтвердил Яша.

— Так воть, в одну секунду я есть заходить и выходить из территории `draw()` 60 раз.

— Ого, быстро! — восхитился Яша.

— Ничего не есть быстро! Я есть мочь много быстрей! И сто раз в секунда, и иногда тысяча раз в секунда! А мочь и много медленней, 2 раза в секунда, или 10 раз в секунда! Но моя любимая скорость — 60 раз в секунда! Все, больше вопросов у вас нет? Мне пора спешить. Я есть наведать вас в следующий раз, — сказала остроконечная шляпа, уже снова растворяясь в облачке.

— Вот это да, — восхищенно пробормотал Яша. — Создатель Планет собственной персоной!

— Похоже, он теперь за нами приглядывает, — не совсем довольно пробормотал Интик, — может, будет часто появляться. Все хорошо, но уж больно он сварливый и придирчивый.

— Интий Флоатович...

— Яша, прекрати! Я же тебя по имени-отчеству не называю!

— Ладно-ладно, дорогой Интик, а как нам эту скорость выполнения программы изменить-то? Что-то старичок не успел дорассказать.

— А для этого есть, Яша, функция `frameRate()`, читается «фрэймРэйт», переводится «кадровЧастота». Мы ей в скобочках число передаем, вот с этой скоростью Создатель Планет и будет перемещаться!

Вот только ему желательно заранее знать эту скорость, чтобы распланировать свой день. Поэтому мы ее стараемся всегда помещать в территорию `setup()`!

— Ага, я понимаю, — подтвердил Яша, — это как с функцией `size()`, мы ее вообще всегда обязаны в `setup()` располагать, да еще и на первом месте.

```
frameRate(число);
//читается «фрэймРэйт», переводится «кадровЧастота»
```

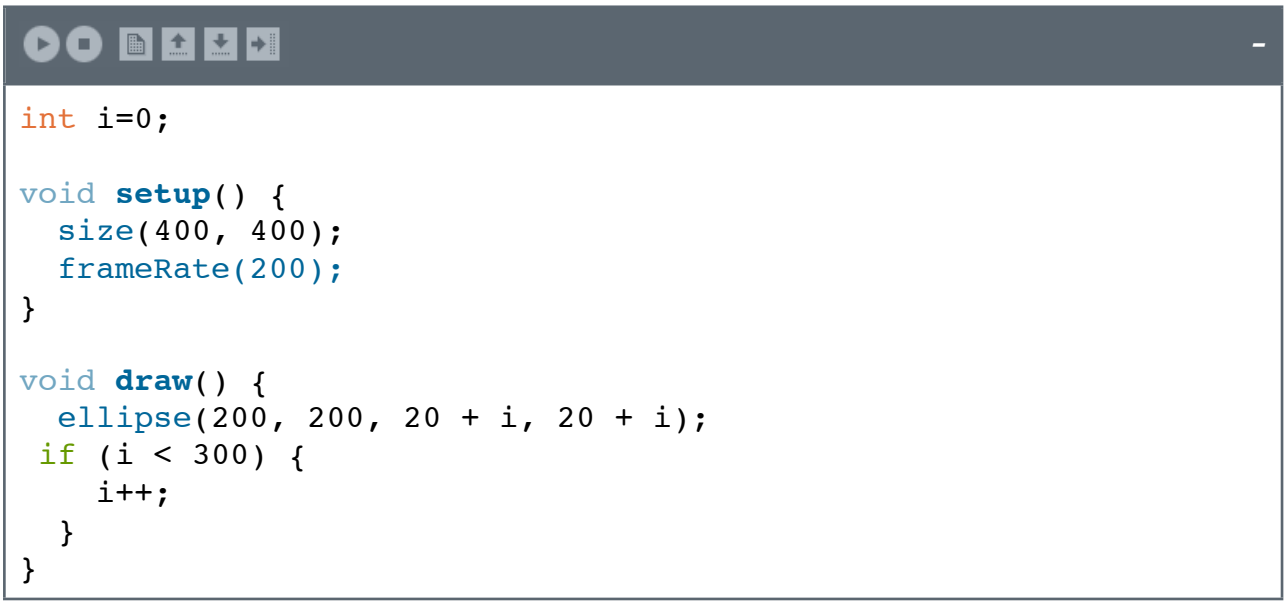
где:

`число` — это число от 1 до скольки хватит сил у Создателя Планет, например, до 10000. Указывает, сколько раз за одну секунду хочется выполнить команды в территории `draw()`

Устанавливает скорость, с которой должна выполняться программа.

Яша балуется с frameRate()

— Не могу удержаться, чтобы к своему прилунению, остановке на орбите не прицепить новую функцию! — заявил Яша. — Оставлю все без изменений, вот только впишу `frameRate(200);` в территорию `setup()` и посмотрю, как она работает:



```
int i=0;

void setup() {
  size(400, 400);
  frameRate(200);
}

void draw() {
  ellipse(200, 200, 20 + i, 20 + i);
 if (i < 300) {
    i++;
  }
}
```

— О, приближается к орбите значительно быстрее!

А вот если поставлю `frameRate(10);`...

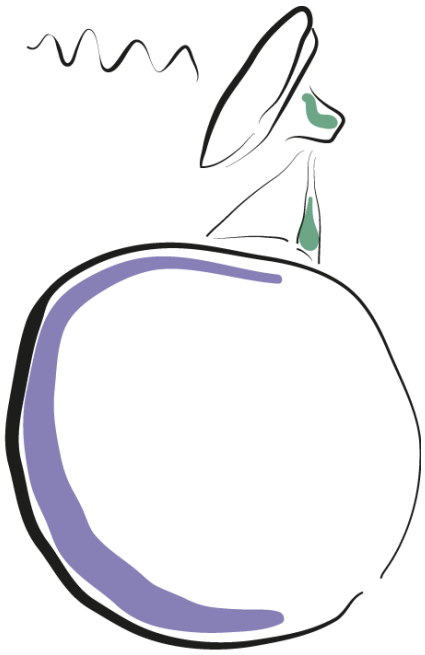
То значительно медленней!

Яша улучшает программу: машем флагами!

— А теперь мой орбитальный корабль будет, вжжж, приближаться к Луне, а затем, уууу, отближаться от нее!

— Отдаляться?

— Да, отдаляться, а затем снова приближаться! Чтобы не запеленговали!



отважные хранители Луны на всякий пожарный пеленгуют пространство

Надо подумать, что мне для этого нужно!

Переменная, которая хранит текущий размер планеты «i» у меня есть.

Каждый раз, заходя в `draw()` Создатель Планет проверяет меньше ли она, чем максимальный размер 300, и если все в порядке — увеличивает ее на 1.

Значит, приближение у меня готово.

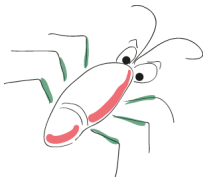
А отдаление должно работать так: как только мы слишком сильно приблизились, то есть размер стал 300, мы начинаем отдалятся, а значит уменьшать нашу переменную «i».

На нашем языке это я напишу так:

```
if (i == 300) {
   i--;
 }
```

Добавлю эту строчку в свою программу:

```
int i = 0;

void setup() {
  size(400, 400);
  frameRate(200);
}

void draw() {
  ellipse(200, 200, 20 + i, 20 + i);
 if (i < 300) {
    i++;
  }
 if (i == 300) {
    i--;
  }
}
```



— Интик, вот только она не работает, так как я задумал, почему? — огорчился Яша. — Круг увеличивается, но не уменьшается.

Поиск ошибки. Почему не отдаляемся от Луны

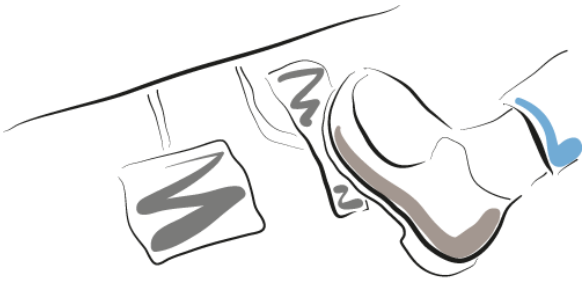
— Давай по-шагам посмотрим, что Создатель Планет делает в твоей программе. Ты же знаешь, какой он дотошный!

— Давай. — согласился Яша.

— Рассмотрим территорию `draw()`:

1. Сначала он рисует эллипс.

2. Затем, если `i` меньше 300, то увеличивает `i` на 1.



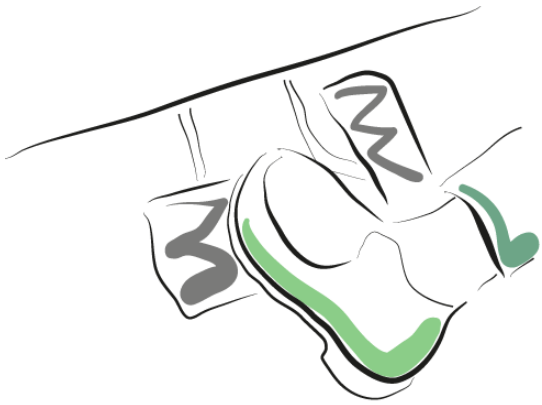
Если видимый размер Луны меньше 300
то дядя Ваня давит на педаль газа
и ее размер увеличивается еще на 1

3. Затем узнает `i` равно ли 300?

А так как первые 300 пробежек территории `draw()` i меньше 300, то внутрь фигурных скобочек он попросту не заходит, и `i` не уменьшает!

Поэтому первые 300 раз, пока i растет от 0 до 300 — круг очень даже хорошо увеличивается.

А вот когда `i` дошло до 300, то он заходит в последнее условие! И уменьшает `i` на единицу! Она становится равной 299.



Если видимый размер Луны равен 300
то дядя Петя давит на педаль тормоза
и ее размер уменьшается на 1

Но что происходит дальше, Яша?

— А дальше он снова заходит в `draw()`, видит, что `i` снова стала меньше 300, и увеличивает ее на 1. И она опять равна 300!

— Теперь догадался? — спросил Интик.

— А, так первое условие значит никогда не дает `i` уменьшится! Ведь если `i` уменьшится, то первое условие его сразу снова увеличит!

— Верно!

— Так, думаю, что нужно исправить...

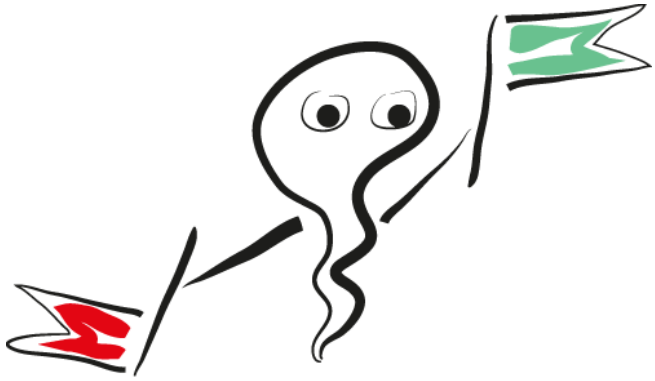
Ну, ясно теперь, что условие `if (i < 300)` больше никогда не подойдет мне. Так как оно будет срабатывать всегда, даже когда мы должны отдаляться от Луны.

Ведь при отдалении `i` будет точно меньше 300, и чем дальше, тем меньше.

— Хочешь совет? — спросил Интик.

— Ага, очень хочу! — ответил Яша.

— Тебе поможет новая переменная, которая будет хранить направление твоего перемещения. Она будет хранить одно значение, когда ты приближаешься к Луне, а другое, когда удаляешься. Такую переменную мы еще называем между собой «флаг».



Слушай мою команду!
Зеленый флаг - приближаемся!
Красный флаг - отдаляемся!

Потому как только по одному размеру Луны невозможно понять приближаешься ли ты к ней, или удаляешься.

— Ага, попробуй тут пойми, если `i` равно 100, например, что нужно делать? Увеличивать `i`, или уменьшать?

А вот если у меня будет флаг... Например, переменная `f`:

```
int f;
```

И если `f` равна 1, значит, мы приближаемся, и `i` нужно увеличивать.

А если `f` равна 2, значит, мы отдаляемся, и `i` нужно уменьшать.

Запишу это так:

```
if (f == 1) i++;
if (f == 2) i--;
```

Но теперь мне нужно как-то устанавливать флаги!

— Ага, — подтвердил Интик, — для этого еще условия нужно добавить. Но кроме этого, ты сейчас создаешь переменную и ничего ей изначально не присваиваешь! А мало ли что ей в голову взбредет!

— Верно! — согласился Яша. — Раз у нас с самого начала идет приближение к Луне, значит, я ей при создании единицу-то и присвою! Вот так:

```
int f = 1;
```

— Флаг на приближение «1» должен устанавливаться с самого начала, когда `i` равно 0.

А флаг на отдаление «2» должен устанавливаться тогда, когда мы достигли максимального приближения, у нас это «300».

Запишу это так:

```
if (i == 0) f = 1;
if (i == 300) f = 2;
```

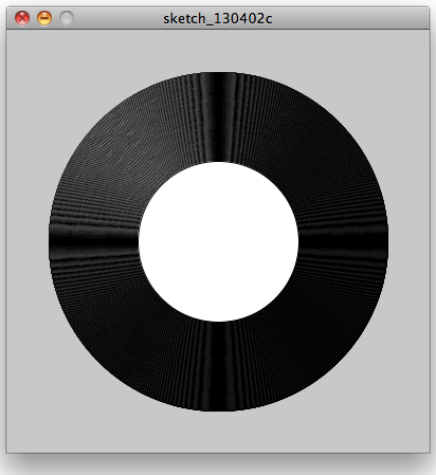
Вот теперь вроде все правильно.

Флаги будут меняться только в крайних случаях, и не будут друг у друга путаться под ногами!

Добавлю это в программу:

```
int i = 0;
int f = 1;


void setup() {
  size(400, 400);
  frameRate(200);
}

void draw() {
  ellipse(200, 200, 20 + i, 20 + i);
  if (f == 1) i++;
  if (f == 2) i--;
  if (i == 0) f = 1;
  if (i == 300) f = 2;
}
```

Отлично! Работает, смотри, что рисует:



— Замечательно выглядит, правда? Вот только не очень похоже пока что.

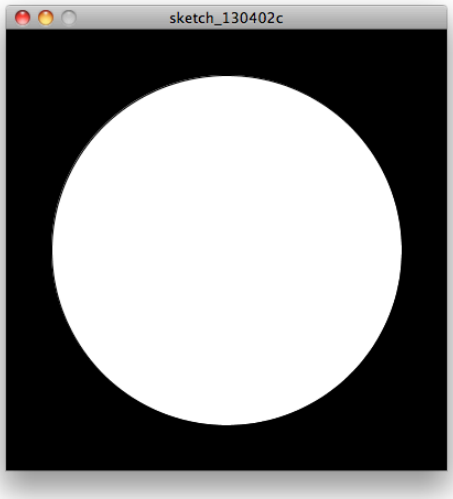
Яша добавляет черный космос

— А, — догадался Яша, — это все потому, что я снова `background()` забыл указать! И каждый раз, заходя в `draw()` новая Луна рисуется поверх старой! Сейчас исправлю! Добавлю:

```
  background(0);
```

```
int i = 0;
int f = 1;


void setup() {
  size(400, 400);
  frameRate(200);
}

void draw() {
  background(0);
  ellipse(200, 200, 20 + i, 20 + i);
  if (f == 1) i++;
  if (f == 2) i--;
  if (i == 0) f = 1;
  if (i == 300) f = 2;
}
```

И вот что получается:



— Настоящая Луна, в настоящем черном космосе!

День четвертый. Улучшаем графический редактор — следим за кнопкой

Монстрики-ДаНет семейства boolean или караульные

— Расскажу тебе о монстриках-переменных из семейства `boolean`. Кстати произносится, как «булин».

— А переводится как? — спросил Яша.

— А никак не переводится! — ответил Интик. — Это семейство названо в честь сержанта Булина, славившегося на заре наших дней краткостью своей речи.

Так вот, про переменные семейства `boolean`. Их еще называют монстрики-ДаНет. Так как они умеют говорит только ДА или только НЕТ.

По-английски ДА, ЭТО ТАК пишется, как `true`, произносится «тру». А НЕТ, ЭТО НЕ ТАК пишется, как `false`, произносится «фолс».

Они не умеют хранить ни числа, ни фразы, это ходячие монстрики переключатели. В одном положении переключателя они говорят — ДА, а в другом — НЕТ.

Есть такая история. Однажды по-ошибке в караул вместо монстрика из семейства `boolean` поставили монстрика из семейства `String`.

boolean
на страже сгущенки

Опасные были дни, враг так и целился на стратегические запасы сгущенки, подтягивал силы. И вот в один из таких дней стоящий на карауле монстрик из семейства `String` пришел докладывать своему начальству: «Здравствуйте многоуважаемое начальство! Я бы хотел засвидетельствовать вам свое почтение, перед тем, как перейду к началу рассказа о том, что мне показалось важным и достойным вашего многоуважаемого внимания...».

Он продолжал и продолжал рассказывать, хотя хотел сказать лишь о том, что враги уже у ворот, и готовят открывашки. В итоге он все же успел предупредить свое командование, хотя и поздновато. К моменту, когда он завершил свое повествование, объединенным силам удалось отстоять только одну банку сгущенки, а остальное, судя по липким следам, бесследно исчезло во вражеском стане.

С тех пор в карауле служат только монстрики из семейства `boolean`, которые на вопрос «есть ли опасность?» отвечают по-армейски четко: «Никак, нет», или «так точно, есть».

А еще у семейства `boolean` обычно имена удобные. Ходит например такой монстрик-ДаНет под именем вБагдадеВсеСпокойно. Ты его окликаешь: “эй,

вБагдадеВсеСпокойно?”. А он тебе в ответ: “ага”! И ты сразу же понимаешь, что все в порядке.

— Дай я угадаю, как это будет на языке программирования выглядеть, — предложил Яша.

— Давай.

— Ну раз он монстрик-ДаНет, то при его создании будем использовать название его семейства `boolean`. Затем запишем его имя и поставим точку с запятой.

В Багдаде все спокойно...

```
boolean vBagdadeVseSpokoino;
```

А теперь пусть в Багдаде будет спокойно, значит присвоим ему значение «`Да, это правда`», то есть `true`.

```
vBagdadeVseSpokoino = true;
```

— А если в Багдаде не спокойно?

— Тогда ему присвоим значение «`Нет, это ложь`», то есть `false`.

```
vBagdadeVseSpokoino = false;
```

Переменная-абориген mousePressed

У нас есть еще один монстрик-абориген, которого не нужно создавать, по имени `mousePressed` (читается, как «маусПрэсд», а переводится, как «мышкаНажата») принадлежащий семейству монстриков ДаНет, то есть `boolean`.

Этот монстрик очень шустрый, собирает сплетни о том, пользуются ли мышкой, нажимают ли на ней кнопки.

И если на мышке кнопка нажата, то он запоминает в себе «Да».

К нему удобно обращаться: «эй, мышкаНажата?». А в ответ тебе: `true`, то есть «Да», или `false`, то есть «Нет».

монстрик mousePressed собирает слухи о том, нажата ли мышка

— Интик, а как к нему обратиться-то можно?

— А для этого, Яша, как раз контроллеры и нужны!

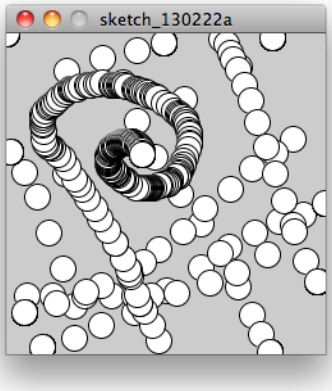
Вот в нашем случае, мы в скобочках можем узнать, равна ли `mousePressed` значению `true`. То есть хранит ли монстрик `мышкаНажата` значение `Да`, `это правда`.

```
  if (mousePressed == true) {
    /* здесь будут команды, которые выполнятся, если
мышка нажата */
  }
```

Яша пишет программу

— Интик, помнишь, я писал такую вот программку, где следы от мыши оставались:

```
void setup() {
  size(250, 250);
}

void draw() {
  ellipse(mouseX, mouseY, 20, 20);
}
```



— Ага, конечно, помню! — подтвердил Интик.
— Так вот я к ней условие хочу дописать, контроллера, чтобы кружки рисовались, только если мышка нажата!
— Мне кажется, что тебе это теперь легко будет сделать! — с уверенностью произнес Интик.
— Попробую... — согласился Яша.

Так как я хочу, чтобы круг рисовался не всегда, а только при каком-то условии, то я помещу его в территорию у контроллера `if`, вот так:

```
  if () {
    ellipse(mouseX, mouseY, 20, 20);
  }
```

Осталось только придумать, какое условие в скобочках написать.

А с другой стороны, что здесь долго думать-то. Условие это, чтобы мышка была нажата.

Помнит же, нажата мыша или нет, монстрик-абориген `mousePressed`, и если она нажата, то он хранит значение `true`, а если нет, то — `false`. Сравнение же у нас обозначается знаком «==».

Поэтому полностью условие я запишу так:

```
if (mousePressed == true) {
    ellipse(mouseX, mouseY, 20, 20);
}
```

А заодно еще и фон установлю черным цветом, вызвав:

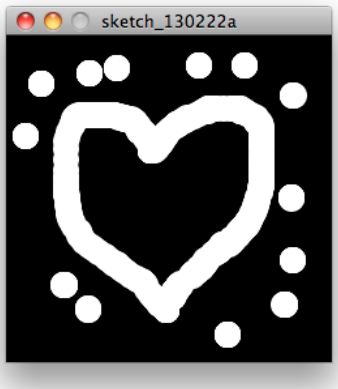
```
  background(0);
```

Вот что получилось:

4-1

```
void setup(){
  size(250, 250);
  background(0);
}

void draw() {
  if (mousePressed == true) {
    ellipse(mouseX, mouseY, 20, 20);
  }
}
```

А вот что я нарисовал в своей программе:



— Милое такое сердечко, Яша! Ты — молодец!

День пятый. Узнаем секреты про контроллеров

Переменная-абориген mouseButton

Еще один монстрик-абориген, который за нас уже создан, называется `mouseButton`, читается «м*а*усБатн», переводится «кнопкаМыши».

— А он, к какому семейству принадлежит? — спросил Яша.

— Он не принадлежит ни к одному из семейств! Волк-одиночка, можно сказать, — ответил Интик.

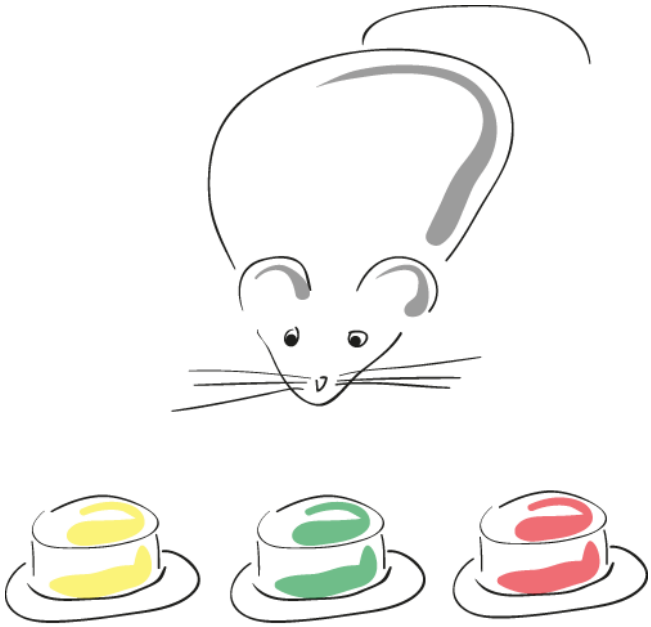
— А что же он тогда запоминает?

— Его словарный запас состоит всего из трех особых слов.

— Особых слов? А чем они отличаются от обычных слов, которые помнят монстрики из семейства `String`? - поинтересовался Яша.

— Особые слова, которые еще называют «Ключевыми словами» никогда не пишутся в кавычках! Всегда пишутся только БОЛЬШИМИ БУКВАМИ. И список этих слов ограничен.

— Ага, у `mouseButton` ты уже сказал, что всего тремя словами ограничен, а какими?



Мышь и ее три кнопки

— Это слова:

`LEFT` (читается «лэфт», переводится «левая») — означает левую кнопку мыши.

`RIGHT` (читается «райт», переводится «правая») — означает правую кнопку мыши.

`CENTER` (читается «сэнтэр», переводится «центр») — означает центральную кнопку мыши.

Ну и стоит помнить, что как любому не нами созданному монстрику-аборигену эти значения, конечно, нельзя присваивать. Но зато можно у него узнавать, какая последняя из кнопок мыши была нажата!

— Интик, то есть я узнаю, что помнит монстрик `mouseButton`, то я узнаю, какая кнопка мыши *сейчас* нажата?

— Не совсем, ты узнаешь, какая кнопка мыши была нажата последней, а когда это произошло, секунду назад, или столетие, этот монстрик не знает, и добиться от него этого не возможно.

— Понял, тогда сначала нужно узнать у монстрика `mousePressed`, он же шустрый и ловкий, нажата ли сейчас кнопка мыши, а потом уже уточнить, какая из них нажата у монстрика `mouseButton`?

— Ага, именно так!

— Вот это разделение обязанностей, понимаю...

Контроллеры играют в матрешки

— Интик, а если мне одного условия мало? — спросил Яша.

— Как это мало? — с недоумением пошевелил ушами Интик.

— А вот мы только недавно обсуждали кнопки мыши.

Когда сначала нужно узнать нажата ли в данный момент кнопка мыши у монстрика `mousePressed`, и если нажата, то узнать у монстрика `mouseButton` какая именно нажата.

Вот я хочу представить, как построить контроллеров для такой задачи.

Самое простое, что могу придумать, это написать следующих контроллеров.

Первое условие проверяет нажата ли кнопка мыши прямо в данный момент:

```
  if (mousePressed == true) {
     /* а здесь написать команды, которые выполнятся
только тогда, когда будет нажата кнопка мыши */
  }
```

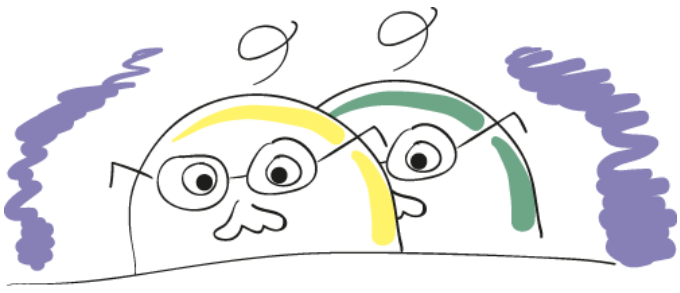
А второе условие проверяет, какая последняя из кнопок мыши была нажата, и равна ли она ПРАВОЙ кнопке:

```
    if (mouseButton == RIGHT) {
      /* а здесь написать команды, которые выполнятся
тогда, когда кнопка мыши будет ПРАВАЯ */
    }
```

— Все правильно, Яша! Теперь эти условия нужно вложить, как матрешки!

— А зачем их матрешить? Вроде и так все понятно.

— Ну, смотри, сейчас у тебя территории этих контроллеров никак не связаны. У первого своя территория, и он решает, пускать ли на нее Создателя Планет или нет. А у второго — своя!



один контроллер
за спиной у другого контроллера

В итоге, даже если первый контроллер не пустит к себе Создателя Планет, то Создатель Планет все равно подойдет ко второму условию, ведь он находится вне территории первого.

Подойдет и спросит — какая кнопка была нажата последней, и если случайно окажется, что ПРАВАЯ, то даже если она была нажата сто лет и один день назад, он зайдет внутрь, в гости к этому контроллеру.

А это вроде не то, что ты хочешь?

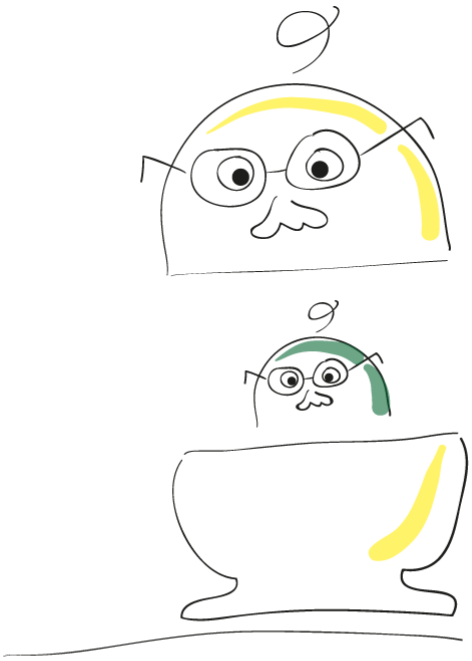
— Ну да, я хочу, чтобы он ко второму контроллеру заходил в гости только тогда, когда его пустит первый!

Кажется, я догадался! Нужно второго контроллера всего целиком вместе с его территорией поместить внутрь территории первого контроллера.

Вот так:

```
  if (mousePressed == true) {
    if (mouseButton == RIGHT) {
      /* а здесь написать команды, которые выполнятся только тогда, когда будет нажата кнопка мыши, и когда кнопка мыши будет ПРАВАЯ */
    }
  }
```

— Молодец, Яша, все правильно! Это действительно сработает!



Один: "Билет есть?"
Другой (писклявым голосом): "А руки помыл?"

Яша играет с программой — добавляет различение кнопок

— А пусть мой редактор теперь умеет отличать одну кнопку мыши от другой! — придумал идею Яша.

— Хорошая мысль, как осуществлять будешь? — поинтересовался Интик.

— Ну, мы же только что вложенных контроллеров проходили! — ответил Яша. — Вот ими и воспользуюсь. Пусть, когда левой кнопкой мыши нажимаю — рисуются большие круги, а вот когда правой — маленькие!

Значит так:

Определять, когда кнопка мыши нажата, это я уже умею, я в этом уже руку, а точнее пальцы набил:

```
if (mousePressed == true) {
}
```

А теперь внутри территории этого контроллера я помещу два раздельных условия. Одно будет определять, нажата ли левая кнопка мыши (`LEFT`), и рисовать большой круг:

```
  if (mouseButton == LEFT) {
    ellipse(mouseX, mouseY, 50, 50);
  }
```

Во второе условие помещу другого контроллера, который будет рисовать маленькие круги, если нажата правая кнопка мыши (`RIGHT`).

```
  if (mouseButton == RIGHT) {
    ellipse(mouseX, mouseY, 20, 20);
  }
```

Вложенные, как матрешки, они будут выглядеть так:

```
  if (mousePressed == true) {
    if (mouseButton == LEFT) {
      ellipse(mouseX, mouseY, 50, 50);
    }
    if (mouseButton == RIGHT) {
      ellipse(mouseX, mouseY, 20, 20);
    }
  }
```

И вся программа вот так:

4-1-1

```
void setup() {
  size(250, 250);
  background(0);
}

void draw() {
  if (mousePressed == true) {
    if (mouseButton == LEFT) {
      ellipse(mouseX, mouseY, 50, 50);
    }
    if (mouseButton == RIGHT) {
      ellipse(mouseX, mouseY, 20, 20);
    }
  }
}
```

— Идеально, Яша! — похвалил его Интик. — С первого раза все правильно!

— Да, — протянул Яша, — такое редко бывает. Обычно все же за блохами приходится гоняться, прямо как Левша!

— Теперь в твоем редакторе даже что-то нарисовать можно!

— А вот я уже! Портрет вон того монстрика! Правда, похож?

Функция: курсор, cursor()

— Это совсем простенькая функция, — сказал Интик. — Мы когда мышку водим по экрану, то обычно у нас ее нос, то есть курсор, имеет форму стрелочки. И вот ее-то и можно поменять!

— Ух ты! Хочу поменять нос у мышки!

— Так все просто, этой функции нужно написать в скобочках одно из шести имен монстриков-постоянных, с которыми она дружит и близко знакома.

```
cursor(имяМонстрикаПостоянной);
//читается «кёсор», переводится «курсор»
```

Устанавливает вид курсора мыши.

где:

имяМонстрикаПостоянной — это одно из следующих слов:

ARROW (читается «эрроу», переводится «стрела»)— это наша обычная стрелочка

CROSS (читается «крос», переводится «крест»)— а это перекрестие, как прицел

HAND (читается «хэнд», переводится «рука»)— это и есть рука

MOVE (читается «мув», переводится «двигать»)— это тоже стрелочка

TEXT (читается «тэкст», переводится «текст»)— а это такая странная фигура, которая появляется, когда мы текст куда-то вводим

WAIT (читается «вэйт», переводится «подождите»)— а таким курсор обычно делают, когда программа над чем-то задумалась, чтобы всем понятно было

Функция: отключение курсора, noCursor()

— А вот дополнение к предыдущей функции. Позволяет курсору исчезнуть!

```
noCursor();
//читается «ноКёсор», переводится «курсораНет»
```

Делает курсор мыши невидимым.

Союзы у контроллеров

Союз «И»

Интик начал рассказывать:

— А я тебе хочу подсказать, как вместо двух вложенных контроллеров можно использовать только одного, но попросив его проверять более сложное условие.

Это сложное условие состоит из двух простых условий, которые должны выполняться одновременно.

— Интик, это я уже понял. Вот, например, если у нас два условия:
Первое условие: кнопка мыши должна быть нажата сейчас.
Второе условие: кнопка мыши должна быть правая.

Раньше мы записывали так:

```
  if (mousePressed == true) {
    if (mouseButton == RIGHT) {
      /* а здесь написать команды, которые выполнятся только тогда, когда будет нажата кнопка мыши, и когда кнопка мыши будет ПРАВАЯ */
    }
  }
```

Вопрос только в том, как их соединить в одно?

— В этом нам поможет союз «И», почти, как в обычном предложении.

— Кажется, я понимаю. Должно быть верным первое условие И должно быть верным второе условие?

— Да, только вместо «И» мы пишем очень уж странный знак, вот так: «&&».

— Ха, попробуй его еще найди на клавиатуре.

— Да, поискать придется. Обычно нужно нажать клавишу «Shift» и клавишу со цифрой «7». Два раза!

А у нас - обед!
А у вас еще и денег - нет!
А еще у нас - ремонт!
И вообще, я болен - вот!

вредный контроллер

Тогда мы можем создать всего одного контроллера, вот так:

```
  if ((mousePressed == true) && (mouseButton == RIGHT)) {
```

```
    /* а здесь написать команды, которые выполнятся только тогда, когда будет нажата кнопка мыши, и когда кнопка мыши будет ПРАВАЯ */
  }
```

— Ого, что-то круглых скобочек многовато! — воскликнул Яша.

— Да не так уж и много, — не согласился Интик. — Крайние скобочки — так они у всех контроллеров присутствуют.

А внутри каждое из условий мы тоже в круглые скобочки заключаем — так наглядней! Можно сказать, что они у нас перекочевали из контроллеров, когда те были еще матрешкой! А между этими условиями и ставим нужный нам союз. Вот сейчас мы союз «И» («&&») поставили!

— Я понял. Союз «И» («&&») просто заменяет вложенные друг в друга контроллеры.

Союз «ИЛИ»

Но, Интик, ты так говоришь, как будто могут быть и другие союзы!

— А ведь и правда есть другие союзы, Яша, — вкрадчиво заметил Интик.

— Какие? — спросил Яша.

— Ну, например, союз «ИЛИ», который пишется, как две вертикальные палочки: «||». Обычно нужно нажать клавишу «Shift» и клавишу со знаком «|». Два раза!

— А где он может пригодиться?

— Например, в соревнованиях по шахматам.

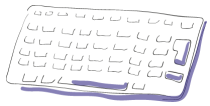
Условие: В третьем туре, если победишь Сашу ИЛИ победишь Ваню, то пройдешь во второй тур.

```
если (победилСашу ИЛИ победилВаню){
    прошел во второй тур;
}
```

Подсказка

Как набрать на клавиатуре знак И «&&».

Нажми комбинацию клавиши «Shift» и клавиши с цифрой «7». Главное, чтобы язык ввода установлен был в «русский».



Как набрать на клавиатуре знак ИЛИ «||».

Так как на разных клавиатурах он запрятан в разные места, то придется поискать. Обычно он находится сверху от клавиши «Enter» и обозначается как двоеточие, но вместо точек — вытянутые вертикальные линии.

Нужно нажать клавишу «Shift» и эту клавишу со знаком «|».

Главное, чтобы язык ввода установлен был в «русский».

Яша пробует играть с курсором

— Ну ладно, — сказал Яша. — Посмотрю, как эти курсоры вживую выглядят!

Отведу каждому из них по куску экрана шириной в 50 пикселей. Пусть при наведении мышки они меняются!

А раз их всего семь: шесть плюс невидимый, то семь умножить на 50 получаем 350. Поэтому сделаю экран шириной в 350 пикселей, а высотой любой можно, путь хоть 100!

```
size(350, 100);
```

Значит, для первого вида курсора координата мышки должна быть меньше 50, это просто:

```
if (mouseX < 50) {
  cursor(ARROW);
}
```

А вот для второго вида курсора мне придется составное условие делать, а то если я начну вложенных контроллеров-матрешек плодить, то их слишком много будет!

Для второго вида курсора, нужно, чтобы мышка не залезала на предыдущую область, то есть была больше, чем 50 пикселей, но и меньше чем 100!

Вот у нас два условия налицо.

Первое: `(mouseX > 50)`, и второе: `(mouseX < 100)`. Объединю их союзом И (`&&`):

```
if (mouseX > 50 && mouseX < 100) {
  cursor(CROSS);
}
```

Ну а все остальные похожим образом напишу, и получу такую программу:

3-3

```
void setup(){
  size(350, 100);
}

void draw()
{
  if (mouseX < 50) {
    cursor(ARROW);
  }
  if (mouseX > 50 && mouseX < 100) {
    cursor(CROSS);
  }
  if (mouseX > 100 && mouseX < 150) {
    cursor(HAND);
  }
  if (mouseX > 150 && mouseX < 200) {
    cursor(MOVE);
  }
  if (mouseX > 200 && mouseX < 250) {
    cursor(TEXT);
  }
  if (mouseX > 250 && mouseX < 300) {
    cursor(WAIT);
  }
  if (mouseX > 300) {
    noCursor();
  }
}
```

— Отлично, Яша, — похвалил Интик, — только ты внутри составных условий скобочки забыл!

— Ну, оно ведь и без них работает! — возразил Яша.

— Работает, в этом случае — работает. А вот в другом, в другом может и не сработать!

— Это ты что имеешь в виду, Интик? — поинтересовался Яша.

— Я тебе сейчас расскажу. В главе для настоящих профессионалов!

Почему важны скобочки в условиях

«Казнить нельзя помиловать»
Загадка.

РЕЖИМ ЧТЕНИЯ:
повышенная сложность

Интик начал издалека:
— Скажем есть царь, который решает казнить или нет слугу.
И пишет такое условие:

```
если (слугаНевиновен ИЛИ слугаРаскаялся И вернул100Золотых){
   помиловать;
}
```

Вот тебе и загадка, в каком порядке эти И и ИЛИ будут применяться?

Яша подумал:
— Тут возможно два варианта. Первый:

```
если ((слугаНевиновен) ИЛИ (слугаРаскаялся И вернул100Золотых)){
   помиловать;
}
```

и второй:

```
если ((слугаНевиновен ИЛИ слугаРаскаялся) И (вернул100Золотых)){
   помиловать;
}
```

— А так как сначала выполняются действия в скобочках, то результаты будут совершенно разными! — заметил Интик.

— Ага, в первом случае, слуга должен быть или невиновен, или же одновременно вернуть 100 золотых и раскаяться!

— А во втором?

— Во втором ситуация совсем другая! Слуга в любом случае должен вернуть 100 золотых, и одновременно или раскаяться или оказаться невиновным! — Яша подумал, и добавил. — Я думаю, что второе условие слуге бы не очень понравилось! Быть невиновным, но при этом все равно вернуть 100 золотых, куда уж как неприятно!

— Вот-вот, поэтому лучше всегда заключать все в скобочки! — заключил Интик. — А то такие логические баги полезут, сто лет в обед не выловишь!

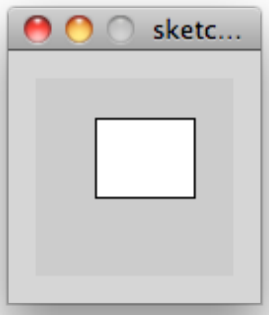
День шестой. Квадраты, треугольники и все такое

Функция: квадрат, rect()

— Сегодня я расскажу тебе, как рисовать квадраты! Для этого потребуется следующая функция:

```
rect(x, y, w, h);
//читается «рэкт», сокращение от rectangle («рэктэнгл»), переводится «прямоугольник»
```

Рисует квадрат.



где:

`x` — это x-координата верхнего левого угла

`y` — это y-координата верхнего левого угла

`w` — это ширина

`h` — это высота

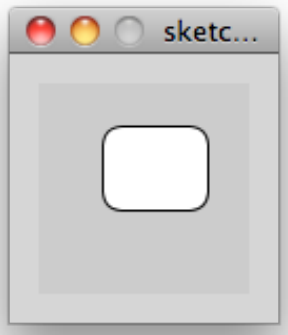
Пример:

`rect(30, 20, 50, 40);` — рисует квадрат шириной 50 пикселей, высотой 40 пикселей так, что левый верхний угол имеет координаты 30 на 20.

— А еще вот секретики, — добавил Интик. — Можно сделать так, что у него будут углы не острые, а закругленные! Смотри!

```
rect(x, y, w, h, r);
```

Рисует квадрат с закругленными углами.



где:

`x, y, w, h` — это то же самое, что и у обычного квадрата

`r` — показывает насколько сильно закруглять углы

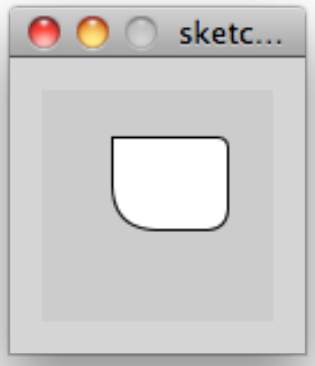
Пример:

`rect(30, 20, 50, 40, 10);` — рисует такой же квадрат, как и раньше, но с закруглениями углов в 10 пикселей!

— А еще можно каждый угол у квадрата закруглять по-своему, — добавил Интик. — Смотри!

```
rect(x, y, w, h, r1, r2, r3, r4);
```

Рисует квадрат с закругленными углами, где каждый угол закругляется по-своему.



где:

`x, y, w, h` — это то же самое, что и у обычного квадрата

`r1, r2, r3, r4` — показывает насколько сильно закруглить каждый из углов

Пример:

`rect(30, 20, 50, 40, 0, 5, 10, 20);` — рисует такой же квадрат, как и раньше, но первый угол не закругляется, второй закругляется на 5, третий на 10, а четвертый на 20 пикселей!

Функция: режим квадрата, rectMode()

— Интик, а почему, когда рисуем круг, мы указываем его центр, а когда рисуем квадрат, то координаты левого верхнего угла? Мне кажется, что это немного несправедливо!

— Это Яша, так исторически сложилось!

— Эх, история, инквизиция, крестовые походы, нашествие Чингисхана...

— Но мы можем это изменить! Для этого есть специальные функции! Вот, например, чтобы изменить способ рисования квадрата, нужно перед тем, как написать `rect()` вызвать следующую функцию:

```
rectMode(режим);
//читается «рэктМоуд», переводится «режимПрямоугольника»
```

где:

`режим` — это одно из следующих слов:

`CENTER` (читается «*сэнтэр*», переводится «*центр*»)— задает отсчет от центра квадрата.

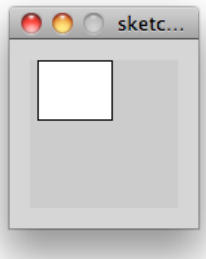
`CORNER` (читается «*конэр*», переводится «*угол*»)— задает отсчет от левого верхнего угла квадрата.

`CORNERS` (читается «*конэрс*», переводится «*углы*»)— хитрый способ, первые два параметра при вызове `rect()` это координаты левого верхнего угла, а вторые два — правого нижнего.

Пример:

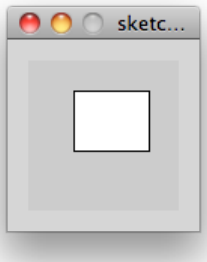
Вот смотри, один и тот же вызов рисования квадрата, но перед ним каждый раз устанавливаются разные режимы его рисования:

```
rectMode(CENTER);
rect(30, 20, 50, 40);
```



— А это похоже на то, как овал рисуется. Центр, ширина и высота!

```
rectMode(CORNER);
rect(30, 20, 50, 40);
```

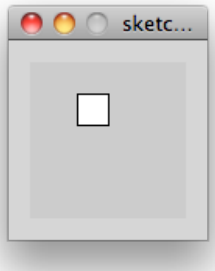


— О, — заметил Яша, — а здесь наш прямоугольник нарисовался точно так же, как и раньше! Как будто мы никакого `rectMode();` и не вызывали!

— Точно, Яша, — подтвердил Интик, — ведь этот режим вызывается автоматически! Мы еще говорим «по-умолчанию»!

— А, так же, как по-умолчанию, выбирается цвет фона серый, и по-умолчанию выбирается цвет рисования белый. Я понял!

```
rectMode(CORNERS);
rect(30, 20, 50, 40);
```



— А здесь квадратик, почему такой маленький получился? — спросил Яша.

— Так ведь последние два числа в режиме `rectMode(CORNERS);` больше не его ширина и высота, а координаты его правого нижнего угла!

— Интик, раз у прямоугольника есть такие режимы рисования, то наверно и при выводе овала тоже можно задать режим? — спросил Яша.

— Конечно, оно все очень похоже, вот смотри.

Функция: режим овала, ellipseMode()

— Напомню, что сначала устанавливаем режим рисования, а только потом рисуем! Это как с рисованием красками, когда сначала окунаем кисточку в краску и только потом рисуем.

— Да, если сначала поводить по бумаге сухой кисточкой, а потом макнуть ее в баночку с краской, то как-то глупо получится, — подтвердил Яша.

```
ellipseMode(режим);
//читается «илипсМоуд», переводится «режимОвала»
```

где:

`режим` — это одно из следующих слов:

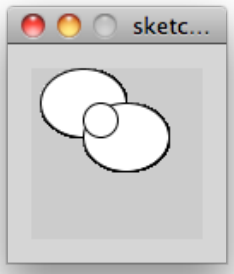
`CENTER` (читается «*сэнтэр*», переводится «*центр*»)— задает отсчет от центра.

`CORNER` (читается «*конэр*», переводится «*угол*»)— задает отсчет от левого верхнего угла.

`CORNERS` (читается «*конэрс*», переводится «*углы*»)— хитрый способ, первые два параметра при вызове `ellipse()` это координаты левого верхнего угла, а вторые два — правого нижнего.

Пример:

```
void setup() {
}
void draw() {
  ellipseMode(CENTER);
  ellipse(30, 20, 50, 40);

  ellipseMode(CORNER);
  ellipse(30, 20, 50, 40);

  ellipseMode(CORNERS);
  ellipse(30, 20, 50, 40);
}
```



Функция: треугольник, triangle()

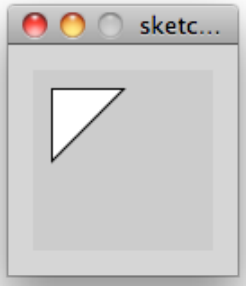
— Уф, как много ты мне всего рассказал, Интик, — выдохнул Яша, вытирая пот со лба.

— Погоди, то ли еще будет! Знаешь, сколько я всего знаю? Устал, или еще про треугольники рассказать?

— Я хочу рисовать треугольники, так что рассказывай!

```
triangle(x1, y1, x2, y2, x3, y3);
//читается «трайэнгл», переводится «треугольник»
```

Рисует треугольник, собственно.



где:

У треугольника три точки, которые еще называют вершинами.

`x1` — это x-координата 1-ой точки

`y1` — это y-координата 1-ой точки

`x2` — это x-координата 2-ой точки

`y2` — это y-координата 2-ой точки

`x3` — это x-координата 3-ой точки

`y3` — это y-координата 3-ой точки

Пример:

`triangle(10, 10, 50, 10, 10, 50);` — рисует треугольник, у которого:

Первая вершина — это 10, 10.

Вторая вершина — это 50, 10.

Третья вершина — это 10, 50.

— Интик, а у треугольника тоже есть режим рисования, как у прямоугольника и у овала? — спросил Яша.

— Не, у треугольника нет такого, — ответил Интик.

— Уф, — вздохнул Яша.

Яша пишет программу

Хорошо отдохнув Яша принялся за написание программы.

— Вот у меня раньше была такая программа:

4-1-1

```
void setup() {
  size(250, 250);
  background(0);
}

void draw() {
  if (mousePressed == true) {
    if (mouseButton == LEFT) {
      ellipse(mouseX, mouseY, 50, 50);
    }
    if (mouseButton == RIGHT) {
      ellipse(mouseX, mouseY, 20, 20);
    }
  }
}
```

Но ведь скучно рисовать только овалы!

Пусть у меня правой кнопкой мыши теперь рисуются квадраты! Еще и `rectMode()` опробую.

```
  if (mousePressed == true && mouseButton == RIGHT) {
    rectMode(CENTER);
    rect(mouseX, mouseY, 20, 20);
  }
```

А еще среднюю кнопку мыши добавлю, пусть при нажатии на нее рисуются треугольники!

```
  if (mousePressed == true && mouseButton == CENTER) {
    triangle(mouseX, mouseY, mouseX + 30, mouseY + 10, mouseX +
15, mouseY + 20);
  }
```

Левой, как раньше достанутся от меня овалы.

И Яша сразу написал следующую программу:

4-2

```
void setup() {
  size(250, 250);
  background(0);
}

void draw() {
  if (mousePressed == true && mouseButton == LEFT) {
    ellipse(mouseX, mouseY, 20, 20);
  }
  if (mousePressed == true && mouseButton == RIGHT) {
    rectMode(CENTER);
    rect(mouseX, mouseY, 20, 20);
  }
  if (mousePressed == true && mouseButton == CENTER) {
    triangle(mouseX, mouseY, mouseX + 30, mouseY + 10, mouseX +
15, mouseY + 20);
  }
}
```

И в ней я вот такой смешной космический корабль нарисовал:

День седьмой. Где мои цветные краски?!

Как устроен цвет

— Интик, а как же разными цветами рисовать? А то какое-то все черное да белое.

— Так вот мы как раз и добрались до раскрашивания! У нас как на палитре художника все многообразие оттенков получается смешиванием трех основных цветов.

— И какие же это основные цвета? — спросил Яша.

— Это красный, зеленый и синий. По-английски: red («рэд»), green («грин»), blue («блу»).

— И что все-все остальные цвета получаются от перемешивания этих трех? — не поверил Яша.


```
Базилио: "Нам пожалуйста много разных функций".
Буратино: "И три баночки краски".
```

— Ага, — подтвердил Интик. Вот только это не совсем похоже на рисование на бумаге, есть отличия!

— Какие?

— Во-первых, мы рисуем как бы, не на белой бумаге, а на черной. А во вторых у нас краска необычная, а светящаяся!

Поэтому, если краски мало — то получается черный цвет, а если слишком много — то она слишком уж сильно светиться начинает, и все становится сначала белесым, а затем и вовсе белым.

— Как же я к такому привыкну? — недоумевал Яша.

— А мы с тобой сейчас потренируемся, — успокоил его Интик. — Главное, что стоит запомнить, что основных цветов всего три, и каждого из них можно налить от 0 до 255 литров!

— А откуда число 255?

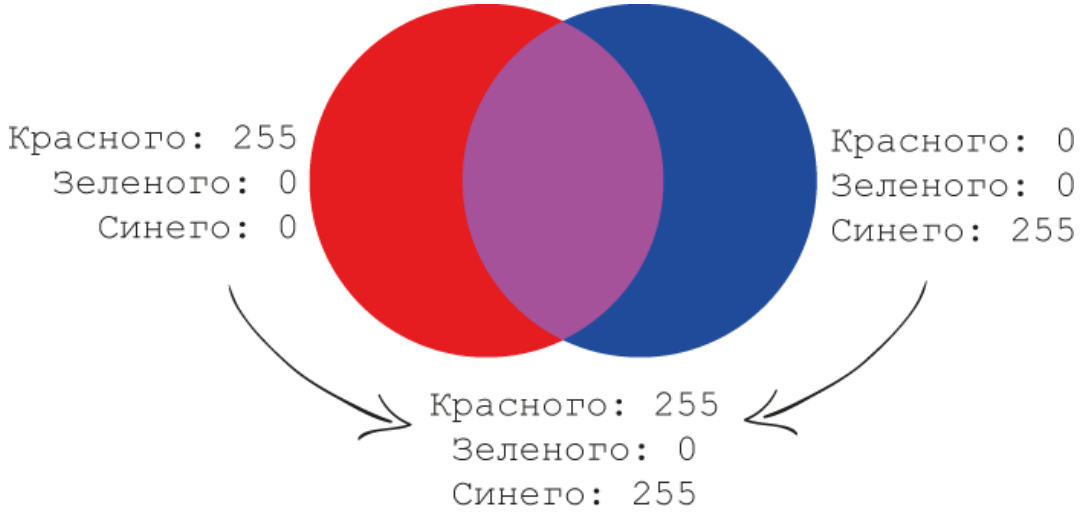
— Просто столько в одно ведерко помещается, — улыбнулся Интик. — Вот попробуй угадать, как сделать красный цвет из этих трех!

— Это наверно задачка на внимательность? — удивился Яша. — Ведь будет красного: 255, зеленого: 0 и синего: 0?

— Верно! Теперь дальше.

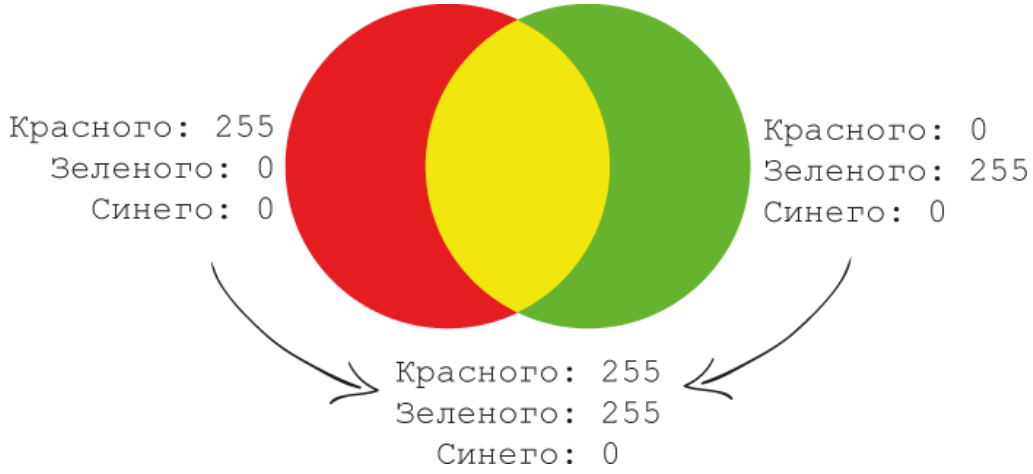
Вот смотри, как получается пурпурный цвет.

Для этого нужно смешать красного: 255, зеленого: 0 и синего: 255!



— А желтый как получить? Ведь на бумаге мы получаем зеленый, смешивая желтый и синий, — спросил Яша.

— А желтый получается нашими светящимися красками при смешивании красного и зеленого, вот так красного: 255, зеленого: 255 и синего: 0.



Семейство цветных монстриков-переменных: color

— А знаешь, кто у нас цвета хранит?

— И кто?

— У нас есть целое семейство монстриков-переменных, называемых `color`, читается «*колор*», переводится «цвет».

— Ага, вот к тем семействам, которые я уже знаю: `int` и `String`, добавилось новое: `color`.

Числовым и словесным монстрикам присваивать значения просто, ведь одна переменная помнит всего одно число:

```
int a = 123;
```

А цветовому же монстрику нужно запомнить аж три числа, сколько красного, сколько зеленого и сколько синего! Не понимаю, как это можно сделать?

— Для этого у них есть семейная функция, которая так и зовется, как их семейство.

— Она называется `color`?

— Ага!

— Создание монстрика из семейства `color`, наверно, выглядит как обычно:

```
color c;
```

— Верно! А присваивание ему, например, черного цвета вот так:

```
c = color(0, 0, 0);
```

— А красного вот тогда так:

```
c = color(255, 0, 0);
```

— Точно!

— А что с этим семейством `color` делать-то можно?

— Например, его можно передать, как параметр, при вызове функции `background();`

```
color c = color(255, 0, 0);
background(c);
```

— И у нас фон станет красным?

— Ага! Хотя, кстати, функции `background()` можно передать цвет и без использования монстриков из семейства `color`, вот таким образом:

```
background(255, 0, 0);
```

— А два параметра передать можно? — спросил Яша.

— Нет, только один, или три. Если один — то, будет заливать фон серым, если это число. А если это не число, а монстрик семейства `color`, то цветным. И если три параметра передашь, то тоже цветом будет рисовать.

А еще не только фон окна, но и все фигуры можно разными цветами рисовать!

— И как это сделать?

— Это я попозже расскажу!

— Ну ладно, тогда я пока с цветом фона поиграю, — сказал Яша.

Яша пишет программу — меняем цвет окна

— Я просто попробовать семейство `color` в деле хочу! Уже достаточно наловчился, чтобы быстренько написать программку, в которой бы цвет фона менялся, от черного к красному!

Для этого я создам переменную `i`, из семейства чисел, которая сначала будет помнить 0,

```
int i = 0;
```

а в конце программы я ей буду прибавлять по единице.

```
i++;
```

И создам переменную `col` из семейства `color`,

```
color col;
```

которой буду присваивать цвет, в котором зеленый и синий равны нулю, а вот красный будет равен `i`, а значит, с каждым разом он будет все ярче и ярче

```
col = color(i, 0, 0);
```

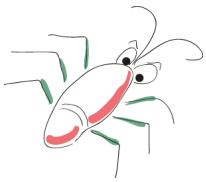
а саму переменную `col` буду передавать в `background();` для установки цвета фона

```
background(col);
```

А все вместе будет у меня вот таким:

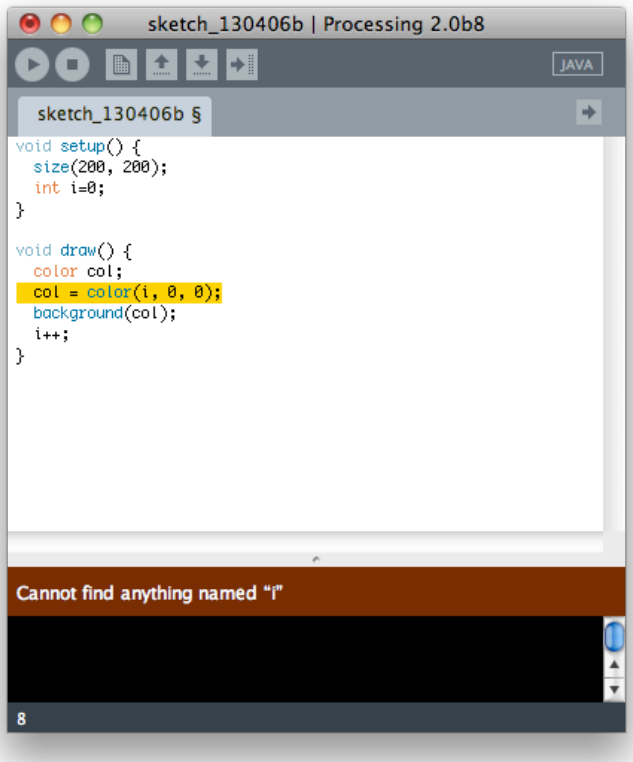
2-1

```
void setup() {
  size(200, 200);
  int i = 0;
}

void draw() {
  color col;
  col = color(i, 0, 0);
  background(col);
  i++;
}
```



Исправляем очепятки

— Интик, вот такая ошибка у меня, очепятка:



— Это он пишет «кэннот файнд энисинг нэймд ‘и’», что в переводе «Не могу найти ничего такого, чтобы называлось ‘i’».

Ты же переменную `i` создаешь в территории `setup()`! А использовать пытаешься в территории `draw()` ! А помнишь основное правило переменных?

— То, что *любая переменная исчезает, как только Создатель Планет выходит из той территории, где она создана*?

— Ага!

— Точно-точно, выходя из `setup()` он просто забывает про переменную `i`, а в территории `draw()` я ее пытаюсь использовать, а он ее просто уже не знает! Эх, коротка же у него память! Ну да ладно, я исправлю. Я ведь помню, что во всех территориях доступны только переменные-аборигены, а чтобы их получить, нужно их создание прописать вне всех территорий.

2-1

```
int i = 0;

void setup() {
  size(200, 200);
}

void draw() {
  color col;
  col = color(i, 0, 0);
  background(col);
  i++;
}
```

— О, здорово, теперь цвет фона меняется!

Яша улучшает программу

Сейчас сделаю его семафором! Чтобы, как только он увеличился до красного, сразу бы снова уменьшался до черного! И дальше опять по-кругу.

Для этого я не буду, как раньше писать `i++`, ведь это заставляет `i` только увеличиваться!

Я заведу нового монстрика-переменную `di`,

```
int di = 2;
```

И i пусть увеличивается на то число, которое `di` помнит!

```
  i += di;
```

А так как `di` у меня с самого начала помнит 2, значит, `i` будет все время расти! С двойной скоростью! Сначала 0, затем 2, потом 4 и так далее...

И когда `i` станет больше, чем 255, то `di` будет пора стать отрицательной! Ведь при дальнейшем прибавлении к `i` отрицательного числа, плюс на минус дает минус, и получается, что мы от `i` будем отнимать 2, а значит, `i` будет уменьшаться!

Напишу вот такого контроллера:

```
  if (i > 255) {
    di = -2;
  }
```

Но как только `i` станет слишком маленьким, ну например, меньше единицы, то `di` снова настанет время стать положительной и равной 2! И получится, что мы вернулись в самое начало, как будто только запустили программу, а значит, у нас все будет без сбоев работать.

Напишу такого контроллера:

```
  if (i < 1) {
    di = 2;
  }
```

И все вместе будет вот таким:

2-1

```
int i = 0;
int di = 2;
void setup() {
  size(200, 200);
}

void draw() {
  color col;
  col = color(i, 0, 0);
  background(col);
  i += di;
  if (i > 255) {
    di = -2;
  }
  if (i < 1) {
    di = 2;
  }
}
```

— Ура! Работает! Окошко семафорит, то красным становится, то черным!
— Прямо Стендаль, — заметил образованный Интик.
— Чего-чего? — не понял Яша.
— Да так, не обращай внимание, ветром навеяло, — улыбнулся Интик.

Секреты про контроллеров

РЕЖИМ ЧТЕНИЯ:
повышенная сложность

— Яша, помнишь, что настоящие маги-программисты любят больше всего?

— Не, что-то запамятовал! — ответил Яша.

— А больше всего они любят делать свою даже уже работающую программу все красивше и красивше! Например, делают так, чтобы она работала так же, а выглядела короче! Или чтобы работала еще быстрее! Мы это еще называем «оптимизацией».

Магов-программистов хлебом не корми, дай им только свою программу оптимизировать!

— Расскажи тогда, будь добр, еще какие-нибудь секретики! — попросил Яша.

— Вот слушай про контроллеров «`if`»!

Ты же знаешь, что у каждого контроллера, есть своя территория, на которую он пускает, только если его условие в круглых скобочках выполнено?

— Ага.

— Так вот секрет в том, что если на этой территории находится всего одна команда, то есть всего одна строчка какого-нибудь приказа, оканчивающаяся точкой с запятой, то можно не писать фигурные скобки!

Вот смотри, вот так выглядит обычный контроллер:

```
if (a == 12) {
  ellipse(20, 20, 30, 30);
}
```

Если `a` равно 12, то рисуется эллипс. Но так как команда всего одна, то можно записать вот так, в одну строчку:

```
if (a == 12) ellipse(20, 20, 30, 30);
```

— Ого! — поразился Яша! — Так это же красивее и короче!

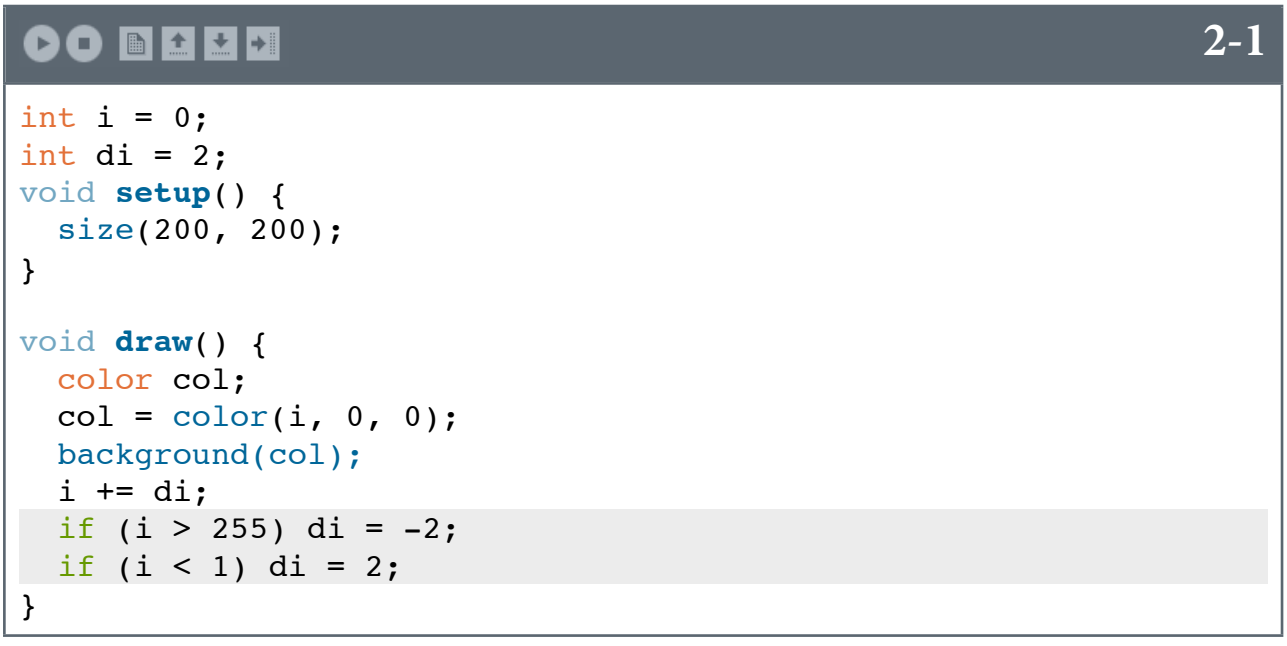
— Только аккуратнее применяй! Легко ошибку допустить!

Яша применяет секреты к своей программе

— Я тоже так хочу, — заявил Яша. — Сейчас свою последнюю программу поменяю! У меня там, в условиях, как раз всего одна команда, значит, можно ее записать без фигурных скобок и в одну строчку! Была такая программа:


```
int i = 0;
int di = 2;
void setup() {
  size(200, 200);
}

void draw() {
  color col;
  col = color(i, 0, 0);
  background(col);
  i += di;
  if (i > 255) {
    di = -2;
  }
  if (i < 1) {
    di = 2;
  }
}
```

А стала такая:




```
int i = 0;
int di = 2;
void setup() {
  size(200, 200);
}

void draw() {
  color col;
  col = color(i, 0, 0);
  background(col);
  i += di;
  if (i > 255) di = -2;
  if (i < 1) di = 2;
}
```

Функция fill() или кисточка

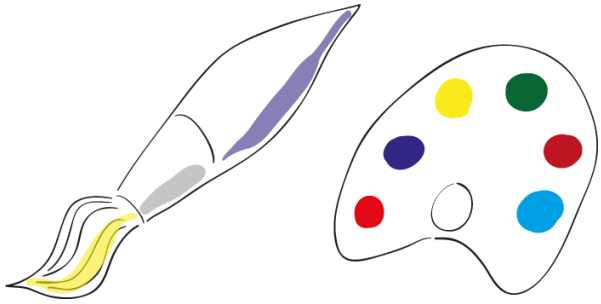
— Яша, перед тем, как начать рисовать, что нужно сделать? — спросил Интик.

— Сначала нужно кисточку окунуть в нужный цвет! — ответил Яша.

— Точно, и функция `fill()` как раз позволяет нам выбрать цвет для рисования.

— Но ведь в раньше мы рисовали, и не вызывали эту функцию! — удивился Яша.

— Так если мы ее не вызываем, то по-умолчанию рисуем белым цветом!



И вообще, это даже не одна функция, а целых несколько! Все зависит от того, сколько параметров в круглых скобках мы ей передадим.

Если передадим один параметр, например `fill(150)`, то число в скобках будет означать серый цвет, при этом, чем число меньше, тем серый будет темнее, а чем число ближе к 255 — тем светлее.

— Это мне очень напоминает функцию `background()`, которая устанавливает цвет фона у окна! — воскликнул Яша.

— Точно! Когда передаем только один параметр, то и там и там — это номер серого цвета! — подтвердил Интик.

— А два параметра нельзя! Это я уже знаю! Так как цвет задается тремя числами! — со знанием дела произнес Яша.

— Верно! Или же можем передать переменную из семейства `color`! — добавил Интик.

```
fill(число);
//читается «фил», переводится «заливка»
```

где:

`число` — это число от 0 до 255. Черный цвет это 0. Белый цвет это 255. Все остальные числа от 1 до 254 — это серый цвет от самого темного до самого светлого.

Или же

```
fill(переменнаяСемействаColor);
```

где:
переменнаяСемействаColor — это и есть переменная семейства color.

Например, выбираем синий цвет для рисования:

```
color col = color(0, 0, 255);
fill(col);
```

Или

```
fill(красный, зеленый, синий);
```

где:
красный — это число от 0 до 255. Сколько красного цвета взять для смешивания и получения итогового цвета;
зеленый — это число от 0 до 255. Сколько зеленого цвета взять для смешивания и получения итогового цвета;
синий — это число от 0 до 255. Сколько синего цвета взять для смешивания и получения итогового цвета;

Например, выбираем синий цвет для рисования:

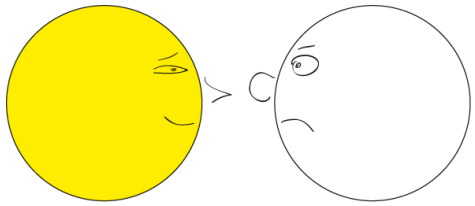
```
fill(0, 0, 255);
```

Функция noFill() или отключение кисточки

— А кстати, еще есть функция «noFill();», — добавил Интик, — она отключает все цвета, и никаким цветом фигуры не рисуются!

— Что же тогда остается? — удивился Яша.

— Остается одна обводка! Контур! Полностью прозрачный, и под ним все видно, что снизу творится!



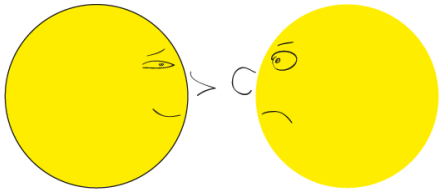
fill() и noFill()

Функция рисования обводки: stroke()

— А еще, — рассказывал Интик, — ты наверно заметил, что наши фигуры часто обведены тонкой линией, которую мы еще называем обводка, или контур.

— Конечно, заметил, — подтвердил Яша.

— Так вот цвет этой линии можно поменять! И делается это так же легко, как вызывается функция `fill();`



`stroke()` и `noStroke()`

Этому служит следующая функция:

```
stroke(число);
//читается «строук», переводится «обводка»
```

где:

`число` — это число от 0 до 255. Черный цвет это 0. Белый цвет это 255. Все остальные числа от 1 до 254 — это серый цвет от самого темного до самого светлого.

Или же

```
stroke(переменнаяСемействаColor);
```

где:

`переменнаяСемействаColor` — это и есть переменная семейства `color`.

Например, выбираем синий цвет для обводки:

```
color col = color(0, 0, 255);
stroke(col);
```

Или

```
stroke(красный, зеленый, синий);
```

где:

`красный` — это число от 0 до 255. Сколько красного цвета взять для смешивания и получения итогового цвета;

`зеленый` — это число от 0 до 255. Сколько зеленого цвета взять для смешивания и получения итогового цвета;

`синий` — это число от 0 до 255. Сколько синего цвета взять для смешивания и получения итогового цвета;

Например, выбираем синий цвет для обводки:

```
stroke(0, 0, 255);
```

Функция отключения обводки: noStroke();

```
noStroke();
//читается «ноуСтроук», переводится «нетОбводки»
```

Отключает обводку у фигур и текста

О прозрачности!

— А еще секретик есть, наши цвета можно сделать прозрачными! Как зеленое бутылочное стекло сквозь которое мы, балуясь, смотрим на улицу. И тогда под одним цветом будут просвечиваться другие.

— Как же это сделать? — спросил Яша.

— Для этого в функции fill(); или stroke(); нужно добавить еще один параметр!

— Ты же говорил, что два параметра нельзя?

— Ну, это для цвета нельзя! А для прозрачности как раз нужно!

Вот тебе примеры.

Серый цвет с номером 100, но полупрозрачный:

```
fill(100, 150);
```

А вот красный почти совсем прозрачный, еле видимый:

```
fill(255, 0, 0, 50);
```

Или вот синий, но практически непрозрачный:

```
stroke(0, 0, 255, 200);
```

— Интик, ты не сказал, но я, кажется, догадался, что прозрачность тоже меняется от 0 до 255?

— Верно! Если пишем 0 — значит совершенно прозрачный цвет, и его вообще не видно. А если пишем 255 — значит совершенно непрозрачный!

Яша пишет цветной графический редактор

— Применю-ка я новые знания в той программке, где разными кнопками мыши рисовались разные фигуры.

Пусть они теперь разными цветами рисуются!

Во-первых, пусть рисуются без обводки, поэтому в `setup()` я размещу

```
  noStroke();
```

Круги пусть рисуются оранжевым цветом, я не знаю, как это точно в числах выразить, но пусть будет много красного, и немного зеленого, а синего пусть вообще не будет:

```
    fill(255, 100, 0);
```

Квадраты пусть рисуются зеленым! То есть красного будет нуль, зеленого, например, 150, а синего, ну его капельку добавлю, скажем: 10.

```
    fill(0, 150, 10);
```

Треугольники сделаю просто синими! Красного значит опять ноль, зеленого совсем чуть-чуть, 30. А синего почти по-максимуму: 200.

```
    fill(0, 30, 200);
```

А вот и целиком программа:

4-2

```
void setup() {
  size(250, 250);
  background(0);
  noStroke();
}

void draw() {
  if (mousePressed == true && mouseButton == LEFT) {
    fill(255, 100, 0);
    ellipse(mouseX, mouseY, 20, 20);
  }
  if (mousePressed == true && mouseButton == RIGHT) {
    rectMode(CENTER);
    fill(0, 150, 10);
    rect(mouseX, mouseY, 20, 20);
  }
  if (mousePressed == true && mouseButton == CENTER) {
    fill(0, 30, 200);
    triangle(mouseX, mouseY, mouseX + 30, mouseY + 10, mouseX +
15, mouseY + 20);
  }
}
```

— И в ней уже можно цветную картинку нарисовать!

День восьмой. Точные числа и случайные, деление и умножение

О функциях-почтальонах

— Сегодня мы познакомимся с весьма необычной функцией, — начал рассказывать Интик. — А особенность ее в том, что если все прошлые функции были сами по себе, как коты в марте — каждая из них располагалась на своей строчке и занимала ее целиком. Например, так:

```
самостоятельнаяФункция();
```

```
- Тук-тук.
- Кто там?
- Это я, функция-почтальон.
Принес подарок для вашего монстрика
```

Эта же функция не мыслит своего существования без монстриков-переменных.

Этакая функция-почтальон. Чтобы она ни делала, а обязана доставить кому-то посылку, или письмо. Поэтому функция-почтальон никогда не располагается на строчке сама по себе. А ее значение обязательно должно присваиваться какому-то монстрику-переменной. Вот так:

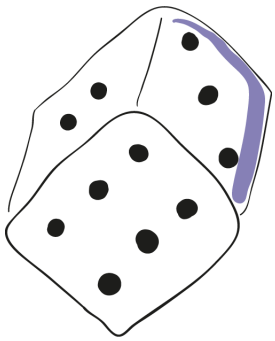
```
переменная = функцияПочтальон();
```

— А, кажется, понял! Функция-почтальон очень похожа на монстриков-переменных, они тоже сами по себе ничего не делают, а только хранят какое-то число, или строку, или еще что-нибудь.

— Да! И там, где Создатель Планет встречает имя монстрика-переменной, то он его заменяет на то, что тот помнит! И с функцией-почтальоном такая же история. Где бы он ее ни встретил, он получает от нее посылку, то есть значение, которое функция возвращает, и использует это значение вместо самой функции.

Функция-почтальон random()

Читается, как «рэндом», переводится «случайноеЧисло».



random(1,6);

— Функция random() — это что-то вроде подбрасывания монетки, когда случайно выпадает орел или решка, или игральной кости — кубика с точками на гранях, когда выпадает случайно число от 1 до 6.

А эта функция возвращает случайное число от 0 до того числа, которое ты ей укажешь в скобках.

И если, предположим, у нас есть переменная r, которой мы хотим присвоить случайное число от 0 до 10, то вызов функции random() выглядит вот так:

```
r = random(10);
```

— Удобно! Что-то вроде многогранного кубика получается, — заинтересовался Яша.

— Ага, вот только возвращает она не целые числа, а дробные.

— Это как?

— Сейчас расскажу, смотри.

```
float random(число);
//читается «рэндом», переводится «случайноеЧисло»
```

Возвращает случайное число от 0 до параметра число

где:

число — число (или переменная из семейства чисел).

Например:

```
float r = random(10);
print(r);
```

В результате может быть напечатано:

```
1.42
```

Или еще можно два параметра передать:

```
float random(числоОт, числоДо);
```

Возвращает случайное число от `числоОт` до `числоДо`

где:
`числоОт` — число (или переменная из семейства чисел).
`числоДо` — число (или переменная из семейства чисел).

Например:

```
float r = random(10, 40);
print(r);
```

В результате может быть напечатано:

```
13.62
```

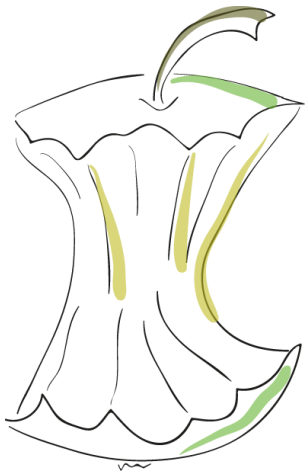
Монстрики-переменные из семейства float

— Интик, ты сейчас рассказывал про `random();` и в ее описании встретилось мне совсем не понятное слово `float`, я даже его прочитать не могу!

— Сейчас объясню, все очень просто. Помнишь, мы уже создавали монстриков-переменных из семейства `int`, то есть целых чисел? — спросил Интик.

— Да, они, например, могли запомнить, сколько у меня яблок в корзинке, — ответил Яша.

— Правильно, но ведь яблок не всегда бывает целое количество?



— Ну, бывает и огрызок какой-нибудь останется, или недоеденное, — ответил Яша.

— Я имею в виду, — уточнил Интик, — если нам нужно разделить яблоко на десять человек, сколько каждому достанется?

— По одной дольке, то есть по одной десятой от яблока.

— Правильно! Но вот наши монстрики из семейства `int` могут помнить только количество целых яблок, как же нам записать еще и дольки? — спросил Интик.

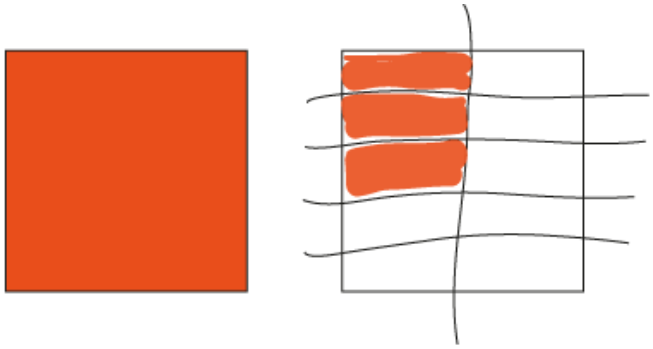
— Наверное, для это существует еще одно семейство монстриков? — спросил Яша.

— Ага, и называются они семейством `float` (читается «фл**о**ут», переводится «плавающие»), и могут помнить не только количество целых яблок, но и количество долек. Для этого в написании числа ставится точка «.» — слева от нее пишется количество целых яблок, а справа, сколько долек от еще одного уже не целого яблока.

Причем число справа показывает, сколько осталось долек, если умозрительно разделить яблоко на десять частей.

Например, если у меня 2 целых яблока и еще 4 дольки от еще одного обгрызенного, которое разрезано на 10 частей, то записывается это как «2.4».

Или вот если у нас есть один квадрат, и еще три кусочка от другого квадрата, разделенного на десять кусочков, то записывается это как «1.3».



1.3

1 целый квадрат и еще три десятых квадрата

— Примерно понятно, — сказал Яша. — А что, если у нас яблоко или квадрат на еще более мелкие части разделены?

— Ну, например, у нас 1 целый квадрат плюс три десятых квадрата, и еще 5 сотых, тогда это записывается так: «1.35».

— Ну-ка я повторю. Целый квадрат записывается так: «1». Одна десятая квадрата, как «0.1». Одна сотая от квадрата: «0.01». А дальше?

— А дальше есть тысячные части, десятитысячные части, стотысячные части, миллионные...

— Эмм... сложновато немного.

— Главное понять следующее. Функция `random()` возвращает число из семейства `float`, поэтому нам его нужно дать запомнить монстрику тоже из семейства `float`.

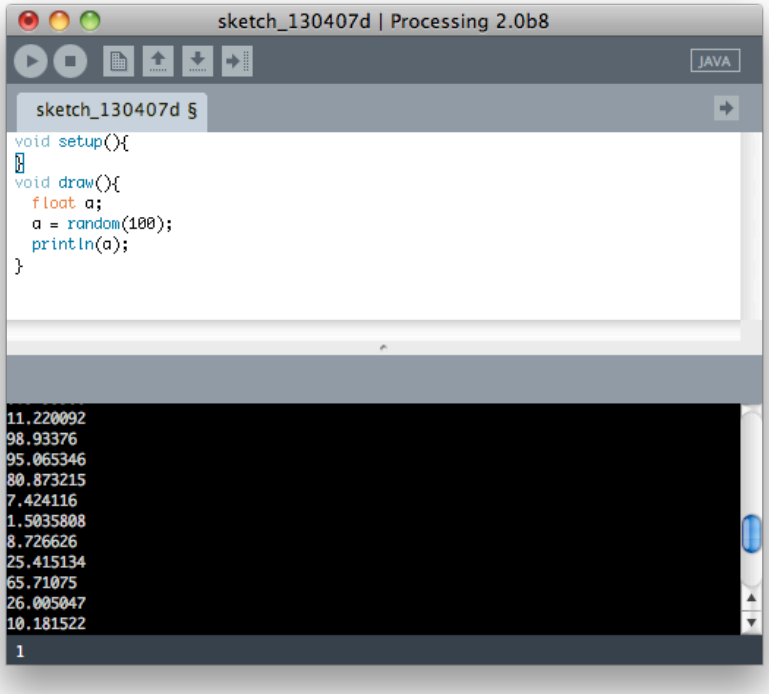
— И как это будет выглядеть?

— Например получение случайного числа от 0 до 100 выглядит так:

4-2

```
void setup(){
}
void draw(){
  float a;
  a = random(100);
  println(a);
}
```

Вот смотри, что получится, если ее запустить:



— Я вижу, что функция `print()` вывела в место для шпаргалки кучу чисел! Вот такие: «`11.220092`», «`98.93376`», «`95.065346`», «`80.873215`», — заметил Яша.

— Вот такие точные числа может хранить монстрик `float`.

— То есть слева от точки запятой число ничем не отличается от `int`, так как показывает, сколько целых у нас яблок, а число справа от точки уточняет сколько к этим целым яблокам нужно еще добавить кусочков от еще одного яблока, но уже не целого.

— А если мне нужно присвоить переменной `myColor` из семейства `int` то, что помнит `float`? — спросил Яша. — Они же родственные друг другу! И то и то — числа.

— Если ты просто попробуешь присвоить переменной из семейства `int` то, что вернет функция `random();` то возникнет ошибка! Так как `int` не могут запоминать дробные числа!

Вот позволь еще по-другому объяснить.

Представь, что у семейства `int` есть корзина только для целых яблок. А у семейства `float` одна корзина для целых и еще одна малюсенькая корзиночка для обрезков. И поэтому, когда к переменной из семейства `int` попадает, например, одно целое яблоко и еще пять десятых, то она теряется и не знает куда девать эти «десятые» — возникает ошибка. Чтобы этого не случилось,

нужно ей явно говорить, что-то вроде: «Ну, не переживай так, просто выбрасывай то, что лишнее», я покажу дальше, как это делается.

— Понял, понял, — подхватил Яша. — В то же время если переменной из семейства `float` отдать целое яблоко, то она просто положит его в корзину для целых яблок! И ошибки никакой не будет!

— Верно! А теперь про то, как семейству `int` запомнить число от семейства `float`. В этом случае используется родственное преобразование `(int)`, оно вот так записывается:

```
int myColor;
myColor = (int)random(255);
```

— А если присвоить наоборот. Переменной `float`, то, что помнит `int`?

— А они присваиваются без проблем! Ведь `float` помнит более точные числа, поэтому она без проблем запомнит менее точное число, которое помнит `int`!

Яша пишет программу — серпантин из рандома!

— Хочу написать новогодний серпантин! — заявил Яша. — И для этого как раз подойдет эта функция случайных чисел, ведь я заранее не знаю, где мои серпантинки окажутся, и каких они будут цветов!

Чтобы цвет был случайным, я передам функции `fill();` не какие-то точные числа, а пусть `random(255);` вернет число от 0 до 255. Напишу так:

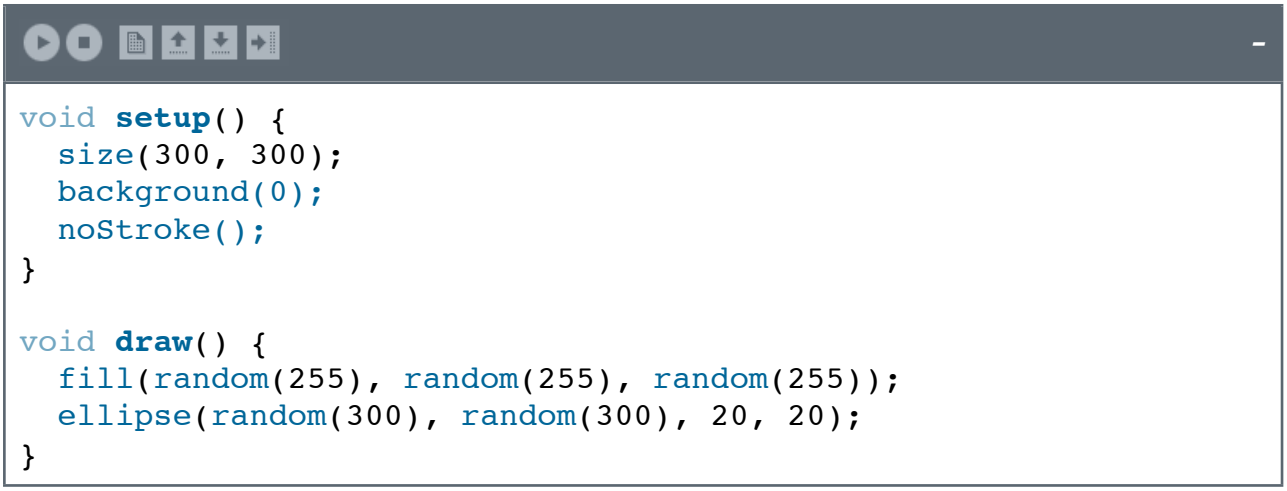
```
fill(random(255), random(255), random(255));
```

— Кстати, Яша, — вмешался Интик, — ты все правильно сделал. Не знаю, как тебе удалось, ведь я не рассказывал еще, что функции `fill();` действительно можно передавать не только целые числа, но и дробные! Она их просто округляет незаметно.

— Спасибо, Интик, не знал. Это у меня случайно так получилось. Так вот, координаты кружков тоже сделаю случайными:

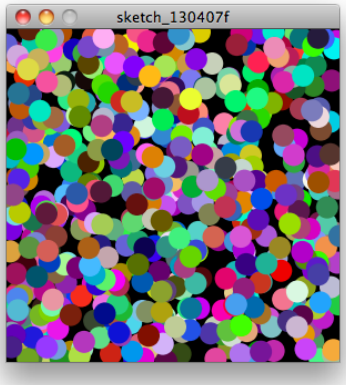
```
ellipse(random(300), random(300), 20, 20);
```

Вот такая коротенькая программка получилась!



```
void setup() {
  size(300, 300);
  background(0);
  noStroke();
}

void draw() {
  fill(random(255), random(255), random(255));
  ellipse(random(300), random(300), 20, 20);
}
```

— И вот какой красивый серпантин она рисует!


sketch_130407f

Деление и умножение

О том, как делить

Гена: "Нам на двоих дали 10 апельсинов: каждому по 8!"
Чебурашка: "Но ведь не получается каждому по 8!"
Гена: "Не знаю, что у тебя получается, а я свои 8 уже съел!"

Интик продолжал рассказывать:

— Раньше мы уже научились складывать и вычитать с помощью монстриков-переменных.

— Пора бы уже научиться делить и умножать! — добавил Яша.

— Ну, тогда давай изучать! — согласился Интик.

Правила у нас простые.

Умножение на нашем программистском языке записывается знаком звездочка, вот таким «*».

— А деление?

— А деление наклонной палочкой, вот так «/».

Тут главное, не перепутать, потому как на нашей клавиатуре есть палочки, наклоненные в разные стороны: «/» и «\».

— А нам нужна только «/» — я запомнил!

— Тогда попробуем поделить что-нибудь на что-нибудь!

— Сначала создадим монстрика `matepus`, и пусть он запомнит число 200:

```
int matepus = 200;
```

— А теперь создадим монстрика `matepom`:

```
int matepom;
```

— И пусть `matepom` теперь запомнит число, которое получится, если то, что помнит `matepus` разделить на 10:

```
matepom = matepus / 10;
```

— Сколько будет помнить `matepom`?

— МАТЕПОМ будет помнить: 200 разделить на 10, а это значит 20!

— Верно!

Деление дробных чисел

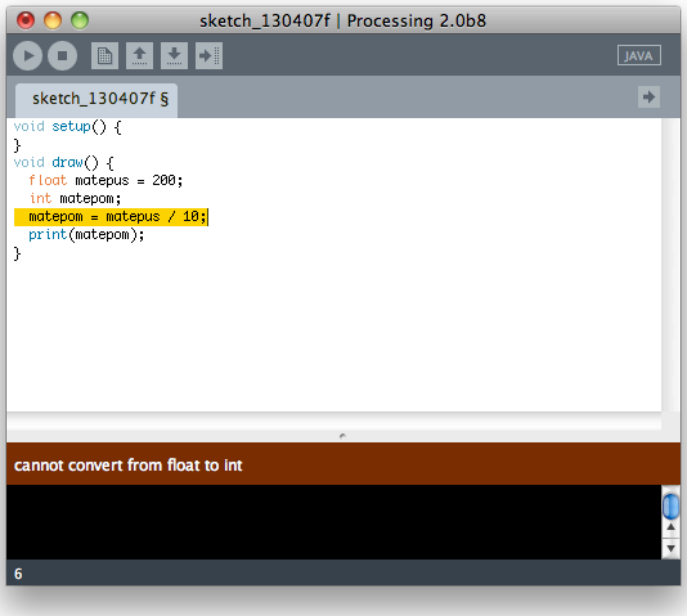
— А теперь задачка посложнее, — сказал Интик. — Вот в предыдущем примере, что случится, если мы переменную `matepus` создадим не в семействе целых чисел `int`, а в семействе дробных чисел `float`?

То есть вот так:

```
float matepus = 200;
int matepom;
matepom = matepus / 10;
print(matepom);
```

— Наверно, ничего не изменится... — предположил Яша.

— А вот и нет! — не согласился Интик. Вот такую ошибку выдаст:



«кэннот конверт фром флоат ту инт», «не могу превратить из float в int», «cannot convert from float to int»

— И что же это значит? — спросил Яша.

— А вот про это я и говорил только что! Переменная `matepom` у нас из семейства `int`, а присвоить мы ей пытаемся результат деления `matepus`, которая принадлежит `float`, на 10!

— То есть даже если мы делим монстрика `float` на целое число, в результате все равно получаем число дробное? Как-то это запутанно!

— Ага.

— Ну, тогда ясно, что `matepom` не хочет запоминать дробное число! Ему нужно преобразование написать такое:

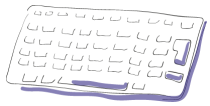
```
matepom = (int) matepus / 10;
```

— Верно!

Подсказка

Как набрать на клавиатуре знак деления «/».

На клавиатурах Windows обычно это клавиша где-то слева от правой клавиши «Shift», главное, чтобы язык набора был установлен в «русский».



Внимание! Постарайся не перепутать знак «/» со знаком «\»! Они так похожи!

Как набрать на клавиатуре знак умножения «*».

Необходимо нажать и удерживать клавишу «Shift», после чего нажать клавишу с цифрой восемь «8».

Шарик бежит за мышкой!

— Интик, у меня идея! — заявил Яша.

— Какая?

— Хочу сделать шарик-кошку, который бы гонялся за мышкой!

А раз он гоняется за мышкой, значит, у него должны быть свои координаты, вот я их и сделаю, и заодно сразу присвою им 0:

```
int x = 0, y = 0;
```

А дальше я буду просто рисовать круг размером 20 на 20 с этими координатами:

```
  ellipse(x, y, 30, 30);
```

Теперь осталось малое, придумать, как их менять так, чтобы они становились все ближе и к ближе к текущим координатам мышки.

Если я их просто буду увеличивать на какое-то число вот так:

```
  x += 10;
  y += 10;
```

То круг, конечно, движется, но совсем не к мышке!

— А чтобы двигаться к мышке, — подсказал Интик, — тебе нужно каждый раз увеличивать координату твоего круга на разность координаты мышки и координаты круга!

Тогда если твой круг будет в точке 100, а мышка в точке 150 (для примера, это координата x), то разница 150-100 будет 50.

— А если я к 100 прибавлю 50, то как раз и получу 150, то есть координату мышки! Если же мышка будет в точке 50, то 100 минус 50 будет равно -50. А если сложить 100 и -50, то снова получу 50, как и хочу!

Тогда я напишу вот так:

```
  x += mouseX - x;
  y += mouseY - y;
```

— Но только в этом случае, твой круг будет двигаться слишком быстро! За один ход сразу передвинется к мышке, а ведь ты не этого хочешь?

— Да, я хочу, чтобы он, как кошка, к ней постепенно подкрадывался! — сказал Яша.

— Тогда тебе не нужно прибавлять сразу всю разность `mouseX-x`, а прибавить только часть ее, например одну десятую.

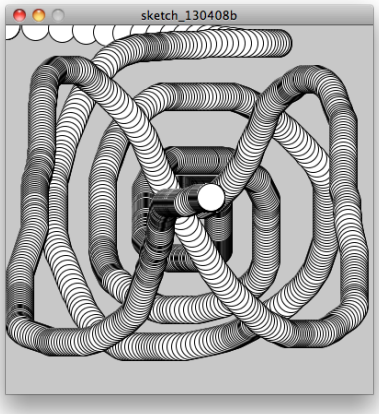
— Одну десятую? Это значит нужно разделить на десять:

```
  x += (mouseX - x) / 10;
  y += (mouseY - y) / 10;
```

И вся программа будет у меня вот такой:

```
int x = 0, y = 0;

void setup() {
  size(400, 400);
}

void draw() {
  ellipse(x, y, 30, 30);
  x += (mouseX - x) / 10;
  y += (mouseY - y) / 10;
}
```

Вот такой классный след остается!



— Вот только почему-то центр кружка-кошки никогда не ловит саму мышку, а болтается немного рядом. Почему так, Интик?

— А это потому, что ты координаты хранил, как числа из семейства `int`. А они же не очень точные, вот ошибка постепенно и накапливается при делении!

— Попробую заменить тогда их на семейство `float`, — сказал Яша, — вот так:

```
float x = 0, y = 0;
```

— О, знаешь, а теперь все заработало, как надо! Центр кружка идеально совместился с курсором мыши! Магия!

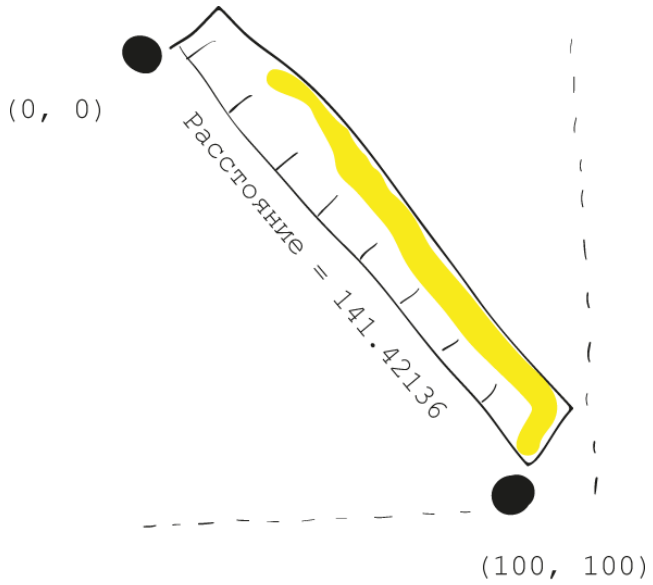
Функция почтальон-линейка: dist()

— Интик, вот у меня есть круг, как мне узнать, что мышка находится внутри него?

— Для этого есть классная функция, которая рассчитывает расстояние между двумя точками на экране!

— А зачем мне расстояние между точками? — с недоумением спросил Яша.

— Очень даже зачем, ведь если расстояние от центра твоего круга до текущего положения мышки будет меньше, чем размер круга (радиус круга), то значит, как раз мышка в нем и находится!



функция-линейка dist()
прилежно вычисляет расстояние

— Здорово, тогда я смогу делать кнопки!

Скажем, если мышка нажата И одновременно находится в моем круге — значит моя кнопка-круг нажата! Расскажи скорее про эту функцию! — попросил Яша.

— Да запросто! Главное помнить, что это тоже функция-почтальон. И ее бессмысленно вызывать саму по себе.

— Кстати, когда я тебе сейчас напишу карточку-подсказку про функцию-почтальон, то перед ней укажу то семейство переменных, которое готово получить результат выполнения функции, — заметил Интик. — Вот так:

```
float dist(x1, y1, x2, y2);
//читается «дист», сокращение от «дистэнс», переводится как «дистанция» или «расстояние»
```

Функция-линейка, рассчитывает расстояние между двумя точками, и возвращает его в формате семейства `float`

где:

`x1` — координата x первой точки.

`y1` — координата y первой точки.

`x2` — координата x второй точки.

`y2` — координата y второй точки.

Например, рассчитаем расстояние от точки (0, 0) до точки (100, 100) и присвоим результат рассчета переменной `r` из семейства `float`:

```
float r = dist(0, 0, 100, 100);
print(r);
```

В результате будет напечатано:

```
141.42136
```

Яша тренируется пользоваться линейкой

— Напишу-ка я маленькую программку, чтобы рисовала круг синего цвета, а если я на него мышкой нажимаю, то он бы становился красным! Как настоящая кнопка!

Размер экрана я сделаю 400 на 400 пикселей.

А круг нарисую ровно в центре, с шириной и высотой в 100 пикселей:

```
ellipse(200, 200, 100, 100);
```

Затем создам переменную из семейства `float`, и сразу скажу ей запомнить расстояние от того, где мышка находится, до центра моего круга:

```
float rad = dist(mouseX, mouseY, 200, 200);
```

А дальше напишу контроллера, который пусть сработает, если одновременно и мышка нажата, и эта переменная `rad` будет меньше размера круга 100!

— Яша, только не 100!

— Почему?

— А потому, что 100 это у тебя ширина круга от его правого края до левого, а тебе же нужно до центра! А до центра будет в два раза меньше.

— То есть 50. Хорошо. Ну и когда условие выполнится, то пусть контроллер установит цвет рисования в красный:

```
if (mousePressed == true &&  rad < 50) {
   fill(255, 0, 0);
}
```

Вот вроде и все...

— А по-моему и не все, — заметил Интик.

— Как это не все? — удивился Яша.

— Ну ты круг красным цветом рисуешь? Рисуешь. А синим, как хотел?

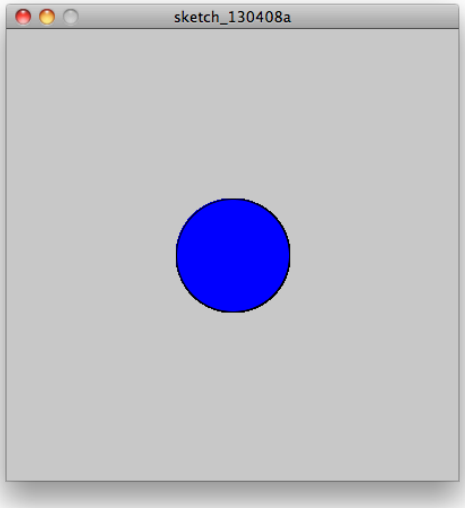
— А, точно. Тогда для этого еще одного контроллера сделаю, который если мышка не нажата, то устанавливает цвет рисования в синий:

```
if (mousePressed != true) {
  fill(0, 0, 255);
}
```

И вся программка вот такая получается:

9-1-1

```
void setup() {
  size(400, 400);
}

void draw() {

  ellipse(200, 200, 100, 100);
  float rad = dist(mouseX, mouseY, 200, 200);
  if (mousePressed == true &&  rad < 50) {
    fill(255, 0, 0);
  }
  if (mousePressed != true) {
    fill(0, 0, 255);
  }
}
```



— И при нажатии на круг он становится красным! Ураа!

Секрет про контроллеров

РЕЖИМ ЧТЕНИЯ:
повышенная сложность

— А хочешь, я тебе секрет про контроллеров расскажу?

— Спрашиваешь!

— Помнишь же, что настоящие маги-программисты обожают делать свои программки как можно изящнее, и, по-возможности, короче?

— Помню-помню, а секрет-то про что?

— Секрет заключается в том, что у обычного контроллера есть помощник, которого зовут `else` («элс», переводится «иначе»). И если у контроллера условие выполняется, то он пускает Создателя Планет к себе. А вот если не выполняется, то к своему помощнику `else`!

```
if (какоетоУсловие) {
   // если условие верно, то выполняются команды здесь
} else {
      // иначе, то есть, если условие не верно, то выполняются команды здесь
}
```

— Я вижу у него своя территория, у этого `else`, — заметил Яша.

— Ага, но видишь, как хитро она прилеплена к окончанию территории самого контроллера?

— Вижу! Я тогда в своей программе так сделаю:

9-1-1

```
void setup() {
  size(400, 400);
}

void draw() {

  ellipse(200, 200, 100, 100);
  float rad = dist(mouseX, mouseY, 200, 200);
  if (mousePressed == true &&  rad < 50) {
    fill(255, 0, 0);
  } else {
    fill(0, 0, 255);
  }
}
```

Яша пишет программу: межзвездные червяки

— А улучшу-ка я свою программу. Сделаю так, чтобы прозрачность обводки менялась в зависимости от расстояния между мышкой и кругом:

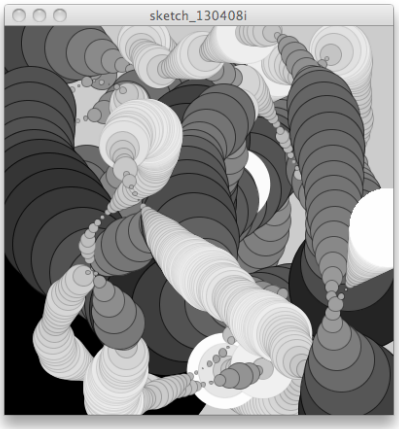
```
  float di = dist(x, y, mouseX, mouseY);
  stroke(0, di);
```

И цвет заливки круга, и сам радиус круга тоже сделаю, чтобы менялся в зависимости от дистанции между кругом и мышкой:

```
  fill(255 - di);
  ellipse(x, y, 80 - di, 80 - di);
```

7-2

```
float x = 100, y = 50;

void setup() {
  size(400, 400);
}

void draw() {
  float dx = (mouseX - x) / 10;
  float dy = (mouseY - y) / 10;
  float di = dist(x, y, mouseX, mouseY);
  stroke(0, di);
  fill(255 - di);
  ellipse(x, y, 80 - di, 80 - di);
  x += dx;
  y += dy;
}
```



День девятый. Продвинутая математика

Функция рисования линии, line()

— О полезнейшей функции хочу тебе поведать, о достойнейший Яша! — произнес Интик.

— Интик, а попроще можно? А то я с таким высоким слогом, наверное, и не пойму ничего, — попросил Яша.

— Да, пожжалте, — согласился Интик. — А функция, о которой хочу рассказать, рисует линию! И работает очень просто, ей нужно указать координаты первой точки и координаты второй.

```
line(x1, y1, x2, y2);
//читается «лайн», переводится «линия»
```

Рисует линию от точки (`x1`, `y1`) до точки (`x2`, `y2`).

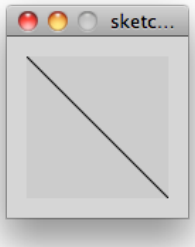
где:

`x1` — координата x первой точки.

`y1` — координата y первой точки.

`x2` — координата x второй точки.

`y2` — координата y второй точки.



Например, нарисуем линию от точки (0, 0) до точки (100, 100):

```
line(0, 0, 100, 100);
```

Функция изменения ширины линии, strokeWeight()

— А эта функция меняет ширину линии. А так как линия это все равно, что и обводка, то меняет и ширину обводки у кругов и квадратов!

```
strokeWeight(число);
//читается «строукВэйт», переводится «весОбводки»
```

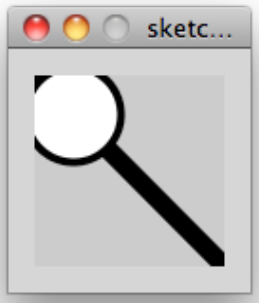
Устанавливает ширину линии, равную значению параметра `число`

где:

`число` — ширина линии.

Например:

```
strokeWeight(10);
line(0, 0, 100, 100);
strokeWeight(4);
ellipse(20, 20, 50, 50);
```



Аборигены pmouseX и pmouseY

— Яша, помнишь про аборигенов `mouseX` и `mouseY`?

— Спрашиваешь! Конечно, а что?

— У них есть приятели, тоже близнецы! Только если `mouseX` и `mouseY` помнят текущее положение мышки на экране, то их приятели помнят предыдущее!

```
pmouseX
//читается «пиМаусИКС»
```

Хранит предыдущее положение мышки, а точнее ее координату X

```
pmouseY
//читается «пиМаусИГРЕК»
```

Хранит предыдущее положение мышки, а точнее ее координату Y

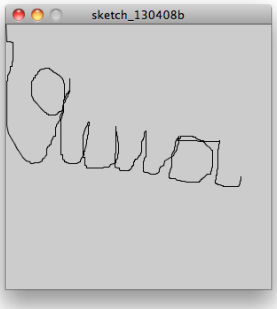
— Интик, а зачем мне это может понадобиться?

— Ну, например, ты можешь нарисовать линию от текущего положения мышки, до предыдущего. И получится так, словно ты рисуешь ручкой!

— О, так я сейчас и попробую:

9-1-1

```
void setup() {
  size(300, 300);
}
void draw() {
  line(mouseX, mouseY, pmouseX, pmouseY);
}
```



Функция-лупа map()

— А вот еще тебе может пригодиться полезная функция. Это функция лупа, она позволяет масштабировать числа.

— Машта... что? — не понял Яша.

— Ну, например, ты хочешь, чтобы в твоей предыдущей программе, чем ниже ты рисовал, тем толще становилась линия, а чем выше — тем тоньше.

Чтобы вверху ширина линии была 1, а внизу 20!

— Ой, обычными способами я даже не знаю, как это сделать! — сказал Яша.

— Вот-вот, а эта функция может вот что сделать.

Ты ей говоришь, что у тебя есть переменная `mouseY`, которая вообще-то меняется от 0 до высоты экрана, скажем, до 255, но ты хочешь, чтобы она менялась от 1 до 20.

И на те, пожалуйста, эта функция-почтальон преобразует твое число в соответствии с твоим заданием.

— То есть, если у меня координата будет 0, то она вернет число 1. А если координата будет 255, то вернет число 20?

— Да! А если координата будет меняться от 0 до 255, то на выходе она будет меняться от 1 до 20, правда, удобно!

— Очень удобно!

```
float map(число, начало1, конец1, начало2, конец2);
//читается «мап», переводится «карта»
```

Преобразует `число` из диапазона (`начало1, конец1`) в диапазон (`начало2, конец2`)

где:

`число` — число (или переменная-число), которое нужно преобразовать.

`начало1` — число, начало первого диапазона.

`конец1` — число, конец первого диапазона.

`начало2` — число, начало второго диапазона.

`конец2` — число, начало второго диапазона.

Например:

```
float a;
a = 0;
println(map(a, 0, 255, 1, 20));
a = 255;
println(map(a, 0, 255, 1, 20));
```

```
a = 125;
println(map(a, 0, 255, 1, 20));
```

Выведет в место для подсказки:

```
1.0
20.0
10.313725
```

Яша пишет программу

— Тогда я добавлю эту функцию `map()` в свою программу и посмотрю, что получится, — сказал Яша.

Пусть у меня ширина линии, как раз через `map()` рассчитывается! Как мы только что и говорили. За основу первым параметром укажу `mouseY`. Вторым и третьим параметром укажу, что `mouseY` может меняться от 0 до 300. А вот на выходе, пусть я получу число от 0 до 20, что укажу четвертым и пятым параметром:

```
strokeWeight(map(mouseY,0,300,0,20));
```

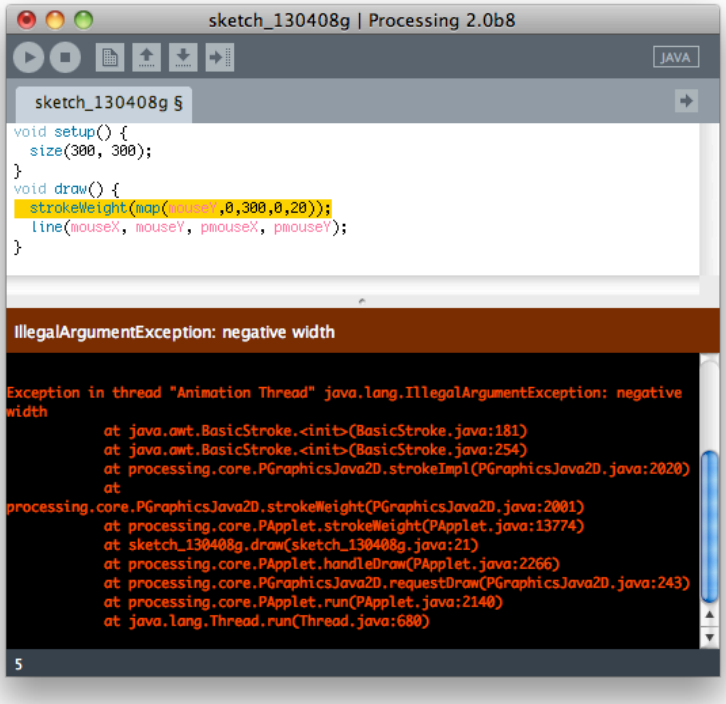
9-1-1

```
void setup() {
  size(300, 300);
}
void draw() {
  strokeWeight(map(mouseY, 0, 300, 0, 20));
  line(mouseX, mouseY, pmouseX, pmouseY);
}
```

— Вон, какие колючки классные получаются! — воскликнул Яша.

Поиск страшного бага

— Интик, вот только если я нажимаю кнопку мыши и такой нажатой увожу ее вверх, за пределы окна рисования, и там отпускаю, у меня какой-то страшный баг вылазит, вот посмотри:



— Он пишет «IllegalArgumentException: negative width», «илли*и*галАгумэнтЭксепшн: нэгатив видз», «недопустимыйАргумент: отрицательная ширина».

А дело здесь в том, что ты же сказал, что `mouseY` у тебя будет меняться от 0 до 300, а когда ты уводишь мышь вверх и отпускаешь, то `mouseY` становится отрицательной, меньше нуля, например -10 или -100. Вот он и выдает ошибку!

— Но это настоящий и очень крутой баг! Я не хочу, чтобы на моей планете такой жил! — заявил Яша. — Что же мне делать?

— А тут тебе просто условие дополнительное поможет. Если ты перед вызовом `map()` сравнишь `mouseY` с нулем, и если оно меньше 0, то присвоишь 0. Так ты избежишь отрицательных чисел! И `mouseY` будет меняться в том диапазоне, который ты указал: от 0 до 300!

— Хорошо, тогда я сделаю временную числовую переменную `m` и сразу присвою ей 0, а вот только в том случае, если `mouseY` будет больше 0, я ей присвою `mouseY`:

```
int m = 0;
if (mouseY > 0) m = mouseY;
```

Ну и саму установку толщины линии поменяю, вместо `mouseY` укажу эту новую `m`:

```
  strokeWeight(map(m, 0, 300, 0, 20));
```

8-1

```
void setup() {
  size(300, 300);
}
void draw() {
  int m = 0;
  if (mouseY > 0) m = mouseY;
  strokeWeight(map(m, 0, 300, 0, 20));
  line(mouseX, mouseY, pmouseX, pmouseY);
}
```

— Класс! Теперь никакой ошибки нет!

Функция-клетка, constrain()

— Вот тебе еще одну функцию полезную подкину, — сказал Интик. — В подобных случаях, когда ты не хочешь, чтобы какое-нибудь число выходило за свои пределы, то посади его в клетку!

— Это как?

— Есть функция-почтальон которой ты передаешь первым параметром число (ну, или имя монстрика числового), а вторым указываешь меньше какого числа, оно не должно становится. А третьим указываешь больше какого числа, оно не должно быть.

В итоге, если оно вдруг станет меньше, чем ты указал, то функция-почтальон, как клетка его ограничит и вернет то, что ты вторым параметром написал.

Вот например, первым параметром ты указал `mouseY`, вторым указал 0, и третьим указал 300. То если `mouseY` станет меньше 0, то эта функция-почтальон все равно вернет 0. А если `mouseY` будет больше 300, например 400, или даже 500, то эта функция-почтальон все равно вернет 300!

— Да, это даже удобнее, чем контроллера писать, — согласился Яша.

```
float constrain(число, минимальноеЗначение,
максимальноеЗначение);
//читается «констрэйн», переводится «ограничение»
```

Не дает параметру `число` стать меньше чем `минимальноеЗначение`, или больше чем `максимальноеЗначение`.

где:

`число` — число (или переменная из семейства чисел), которое нужно преобразовать.

`минимальноеЗначение` — число (или переменная из семейства чисел).

`максимальноеЗначение` — число (или переменная из семейства чисел).

Если `число` больше `минимальноеЗначение` и меньше `максимальноеЗначение`, то возвращает `число` без изменений.

Если `число` меньше `минимальноеЗначение`, то возвращает `минимальноеЗначение`.

Если `число` больше `максимальноеЗначение`, то возвращает `максимальноеЗначение`.

Например:

```
float a;
a = 5;
println(constrain(a, 10, 255));
a = 100;
println(constrain(a, 10, 255));
a = 400;
println(constrain(a, 10, 255));
```

Выведет в место для подсказки:

```
10.0
100.0
255.0
```

— Я тогда в своей программке `constrain()` попробую! — сказал Яша. — Только хитрым способом! Сделаю аж три вложенных функции!

`constrain(mouseY, 0, 300)` вернет мне число от 0 до 300 в соответствии с координатой мышки по вертикали.

Его я вставлю как первый параметр в `map(_вот_сюда_, 0, 300, 0, 20)`

Получится так: `map(constrain(mouseY, 0, 300), 0, 300, 0, 20)`

А все это вместе передам как параметр в `strokeWeight(_вот_сюда_);`

И получится так:

```
strokeWeight(map(constrain(mouseY, 0, 300), 0, 300, 0, 20));
```

8-1

```
void setup() {
  size(300, 300);
}
void draw() {
  strokeWeight(map(constrain(mouseY, 0, 300), 0, 300, 0, 20));
  line(mouseX, mouseY, pmouseX, pmouseY);
}
```

— Ужасно сложно! Но зато сработало!

Монстрики-аборигены: ширина и высота экрана, displayWidth, displayHeight

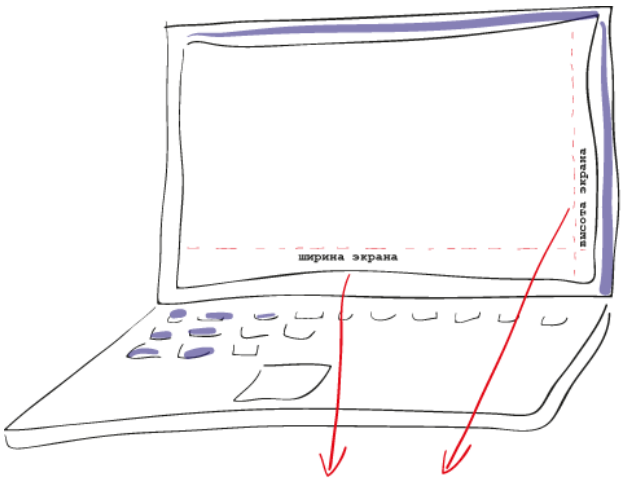
— А вот кратенько про еще двух аборигенчиков.

Монстрик-абориген `displayWidth`, читается, как «дисплэйВидз», а переводится «экранаШирина», всего-навсего хранит в себе ширину нашего экрана.

Хотя еще не все, он очень любит, когда у него спрашивают: «эй, дисплэйВидз, какова ширина нашего экрана?». Он в этом случае с готовностью, но медленно и очень важно отвечает, например, так: 1024.

— Что-то крайне лаконичный ответ, — заметил Яша.

— Да, словоохотливостью он не славится. Зато славится постоянством. Всегда отвечает одно и то же!



size(displayWidth, displayHeight);

А кроме него есть еще брат-близнец: монстрик-абориген `displayHeight`, читается, как «дисплэйХайт», а переводится «экранаВысота», а хранит в себе высоту нашего экрана.

— Интик, а у меня идея! Что будет, если я напишу так:

```
size(displayWidth, displayHeight);
```

— О, правильно мыслишь! Тогда у тебя окно откроется с самым большим размером, каким возможно!

Монстрики-аборигены: ширина и высота окна, width и height

— Знаешь, Яша, когда ты вызываешь функцию `size()`, то автоматически рождаются пара переменных-аборигенчиков.

— Первый раз про такое слышу!

— Ага, это редкость. Они запоминают те числа, которые ты в `size()` указал.

Вот ты например написал:

```
size(500, 200);
```

И в это самое время рождаются монстрики-переменные:

`width` (читается «*видз*», переводится «*ширина*») — запоминает текущую заданную ширину окна

`height` (читается «*хайт*», переводится «*высота*») — запоминает текущую заданную высоту окна

— То есть в нашем примере `width` помнит теперь 500, а `height` помнит 200, — подвел итог Яша. — Но вот только вопрос. А зачем это нужно?

— Ну, это удобно, чтобы, если в процессе написания программы тебе вдруг вздумалось другой размер окна установить, то ты просто в `size()` поменял числа и все! Но для этого тебе в самой программе вместо 500 и 200 нужно использовать `width` и `height`.

— Все равно еще не понимаю, — не понимал Яша.

— Как же, вот у тебя, скажем, длиннющая программа, и в ней эти 500 и 200 мильон раз используются. А теперь ты вдруг захотел сделать окно не в 500, а 600 шириной, ну, что-то у тебя не поместилось по-задумке. И ты в миллионах строчках кода меняешь, как мартышка 500 на 600!

— Ууу, грустная ситуация, — заметил Яша. — Вот теперь понимаю.

День десятый. Работаем с клавиатурой

Переменная-абориген, хранящяя нажатые клавиши на клавиатуре, keyCode

— Яша, а хочешь научиться работать с клавиатурой? — спросил Интик.

— Конечно, хочу.

— Тогда тебе пригодится знание еще одного монстрика-аборигена, который хранит код последней нажатой клавиши на клавиатуре.

```
keyCode
//читается «кИКоуд», переводится «кодКлавиши»
```

Хранит код последней нажатой клавиши в виде числа.

— А кроме того, его можно сравнивать с несколькими особыми словами, — сказал Интик.

— То есть можно его с числами сравнивать, а можно с особыми словами? — переспросил Яша.

— Ага, для удобства так сделано.

— Вот эти слова:

`UP` (читается «ап», переводится «вверх») — означает клавишу «стрелка вверх».

`DOWN` (читается «даун», переводится «вниз») — означает клавишу «стрелка вниз».

`LEFT` (читается «лэфт», переводится «левая») — означает клавишу «стрелка влево».

`RIGHT` (читается «райт», переводится «правая») — означает клавишу «стрелка вправо».

`ALT` (читается «элт», от «alternate», переводится «смена», у нас обычно принято произность так: «альт») — означает клавишу «Alt».

`CONTROL` (читается «кэнтрол», переводится «управляющая», у нас обычно принято произность так: «контрол») — означает клавишу «Control».

`SHIFT` (читается «шифт», переводится «сдвиг») — означает клавишу «Shift».

Ну и стоит помнить, что как любому не нами созданному монстрику-аборигену эти значения, конечно, нельзя присваивать. Но зато можно у него узнавать, какая последняя из кнопок мыши была нажата!

Яша использует keyCode

— Интик, а функция `println();` может использовать `keyCode`? — спросил Яша.

— Ну конечно! Ведь `keyCode` такая же переменная, как и все остальные, только абориген, поэтому ее менять нельзя.

— Тогда я сделаю маленькую тестовую программку, в которой просто буду выводить `keyCode`, заодно узнаю, какие клавиши какой код имеют.

8-1

```
void setup(){
}
void draw(){
  println(keyCode);
}
```

— Во, я запустил, — поделился Яша, — и узнал, что у клавиши вверх цифровой код равен 38, а у клавиши вниз равен 40! Влево код 37, а вправо 39!

— Молодец!

— Вот только я странность заметил, что клавиша пробела и буквенные клавиши при нажатии 0 всегда выдают! И только при отпускании показывают свой код. Например, у пробела он 32!

— Да, есть такая особенность, здорово, что ты ее заметил! Сможешь учесть в своих программах!

— Ага, теперь я знаю, что клавиши стрелок он узнает сразу при нажатии, а вот некоторые другие клавиши только при отпускании. Полезная программка!

Яша пишет программу

— Как бы мне теперь применить умения работать с клавиатурой... — задумался Яша. — О, сделаю-ка я так, чтобы кругом можно было управлять с клавиатуры!

Раз у меня будет двигаться круг, значит, для хранения его координат я заведу монстриков-аборигенов вне территорий `setup()` и `draw()`:

```
int x = 100, y = 100;
```

А дальше напишу условия.

Если код клавиши 38, то есть «вверх», то вычту из `y` единичку:

```
if (keyCode == 38) y--;
```

Если код клавиши 40, то есть «вниз», то добавлю к `y` единичку:

```
if (keyCode == 40) y++;
```

Если код клавиши 37, то есть «влево», то вычту из `x` единичку:

```
if (keyCode == 37) x--;
```

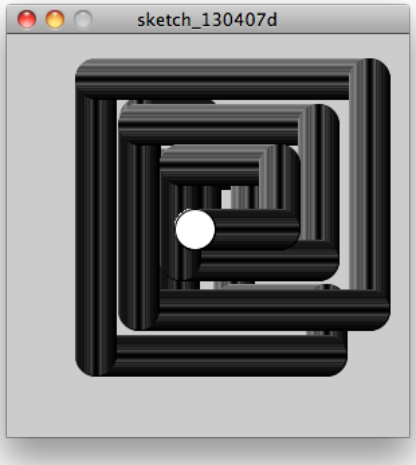
Если код клавиши 39, то есть «вправо», то добавлю к `x` единичку:

```
if (keyCode == 39) x++;
```

Ну и в конце концов напишу сам вызов эллипса: `ellipse(x, y, 30, 30);`

```
int x = 100, y = 100;

void setup(){
  size(300, 300);
}

void draw(){
 if (keyCode == 38) y--;
 if (keyCode == 40) y++;
 if (keyCode == 37) x--;
 if (keyCode == 39) x++;
 ellipse(x, y, 30, 30);
}
```

— Вот, замечательно круг ездит! Смотри! — восхитился Яша.



— Здорово, что ты научился сам определять коды клавиш и их использовать! — похвалил Интик. — Вот только, например, вместо числа 38 для клавиши вверх можно было бы использовать особое слово `UP`:

```
if (keyCode == UP) y--;
```

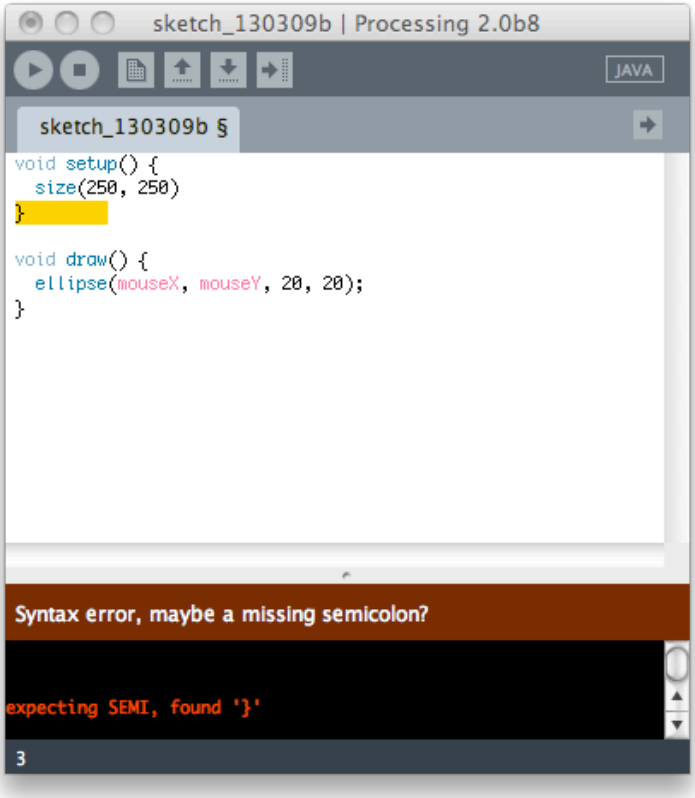
— Сработало бы точно так же! — уверил Интик.
— Ну а раз точно также, то я и не буду переделывать! — ответил на это Яша.
— Ну да, это я так, на будущее...

День одиннадцатый. Работа над ошибками — продолжаем ловить жучков

Если у тебя вдруг не запускается программа, значит, в ней есть какая-то ошибка или опечатка, давай попробуем разобраться, какие ошибки могут встретиться и когда.

Ошибка первая: пропущена точка с запятой

Если мы после вызова функции забыли поставить точку с запятой, то у нас может появиться такая ошибка, вот смотри, что будет, если мы пропустим точку с запятой после вызова функции `size(250, 250)`:

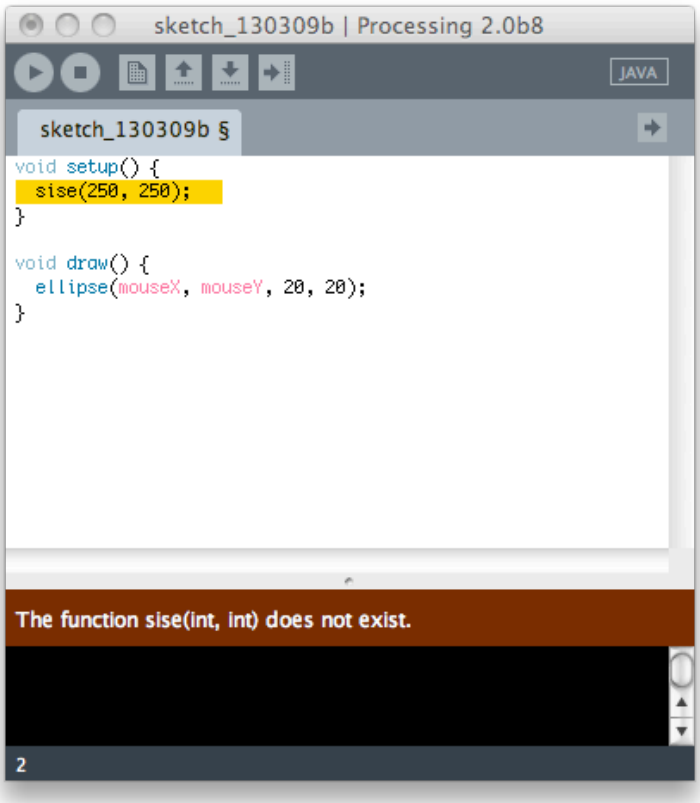


Создатель Планет пытается даже угадать, где ты допустил ошибку! Правда, заметим, что ему это не всегда удается. Но в данном случае он все верно угадал.

Вот посмотри на надпись белым цветом: «Syntax error, maybe a missing semicolon?». Что в переводе означает: «Ошибка синтаксиса, может быть пропущена точка с запятой?».

Ошибка вторая: неправильное имя функции

Если имя функции будет написано с ошибкой, то Создатель Планет будет не знать, что ему делать и может выдать следующую ошибку выполнения программы:



Здесь мы вместо `size(250, 250);` написали `sise(250, 250);` — то есть вместо буквы «z» написали букву «s».

Надпись белым цветом гласит: «The function sise(int, int) does not exist». Что в переводе будет: «Функции sise(int, int) не существует!».

— Интик, а что за (`int`, `int`) после названия функции в сообщении об ошибке? — поинтересовался Яша.

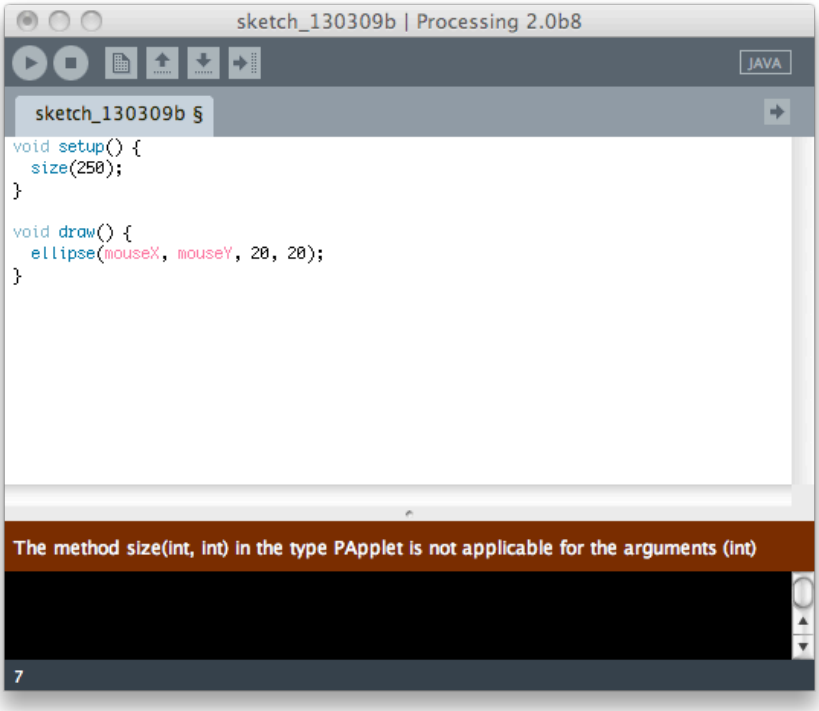
— А это Создатель Планет увидел, что ты пытался передать функции числа 250 и еще раз 250, а эти числа могут хранить только монстрики-переменные семейства `int`! Вот он нам и сказал, что функции под названием `sise`, которой передаются два параметра семейства `int` — не существует!

— А он, кстати, еще и подсвечивает ту функцию, которая ему не понравилась! — воскликнул Яша.

— Ага, это удобно. Только не забывай, пожалуйста, что он может тоже ошибаться, если ошибка какая-нибудь не тривиальная. Он только по простым ошибкам хорошо подсказывает.

Ошибка третья: неправильные параметры

Еще может произойти ошибка, если функции будут переданы неправильные параметры, с которыми она просто не знает что делать. Вот смотри, здесь мы нашей злосчастной функции `size()` недодали один параметр:



Надпись гласит: «The method size(int, int) in the type PApplet is not applicable for the arguments (int)». Что примерно в переводе: «Функция size(int, int) там-то там-то не поддерживает аргумент (int)».

Это намекает нам на то, что от нас ожидалось два параметра (`int`, `int`), а мы передали только один (`int`).

— Здесь уже сложнее понять, в чем ошибка, — задумался Яша.

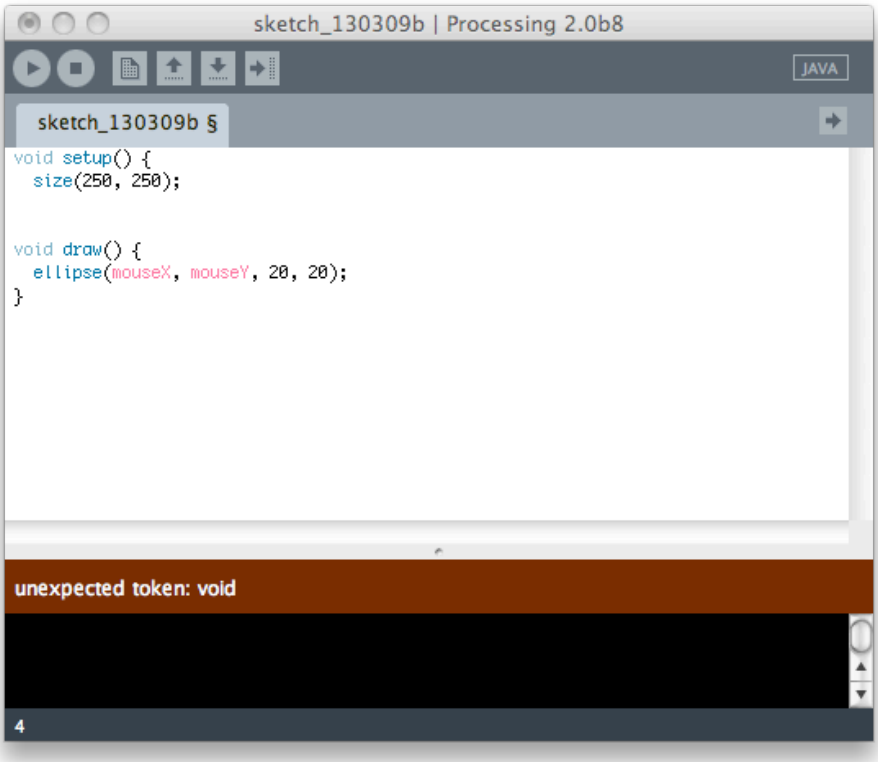
— Да, чем сложнее ошибки, тем сложнее их ловить! Мы ошибки, кстати, называем багами, то есть «жучками».

— Ну что, продолжим ловить жучков? — поинтересовался Яша.

— Давай еще одну рассмотрим, и на сегодня достаточно, — согласился Интик.

Ошибка четвертая: пропущена фигурная скобка

Вот давай пропустим одну из фигурных скобок на территории `setup()` и посмотрим, что произойдет:



— Видишь, отсутствие фигурных скобок сложнее всего отловить! Поэтому за ними нужно часто и пристально наблюдать! Чтобы на каждую открывающую была своя закрывающая.

— А здесь у нас нет закрывающей скобки у территории `setup`, — заметил Яша.

— Верно, поэтому нам пишут: «unexpected token: void». То есть «неожиданное слово `void`».

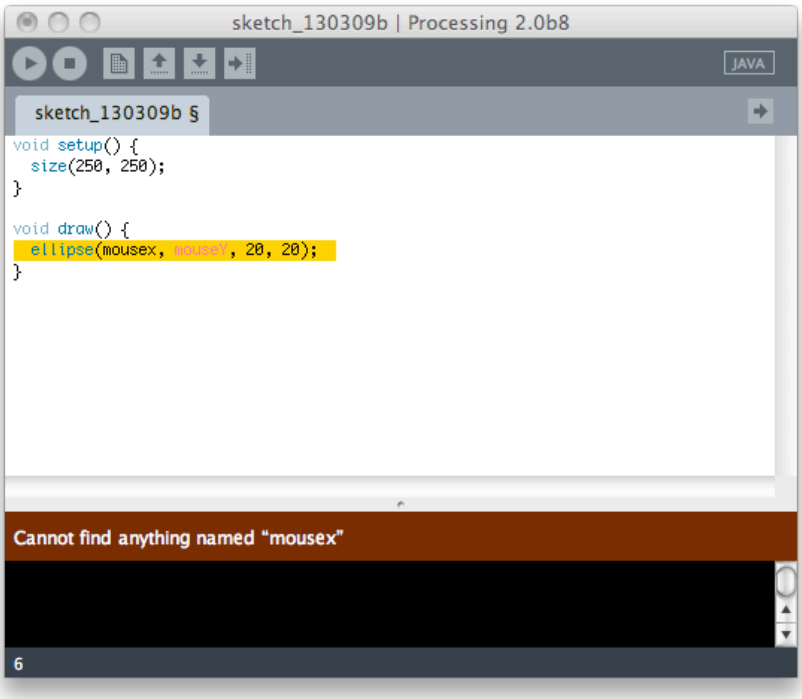
— Так это видно Создатель Планет ожидал закрывающую фигурную скобку! А вместо нее увидел слово `void`! — догадался Яша, — вот и написал про неожиданный `void`.

— Точно!

— Интик, а еще про какую-нибудь ошибку расскажи? — попросил Яша.

Ошибка пятая: перепутанные большие и маленькие буквы

— Нам нужно все время следить, чтобы большие буквы оставались большими, а маленькие — маленькими, вот смотри, что получится, если мы вместо `mouseX` напишем `mousex`:

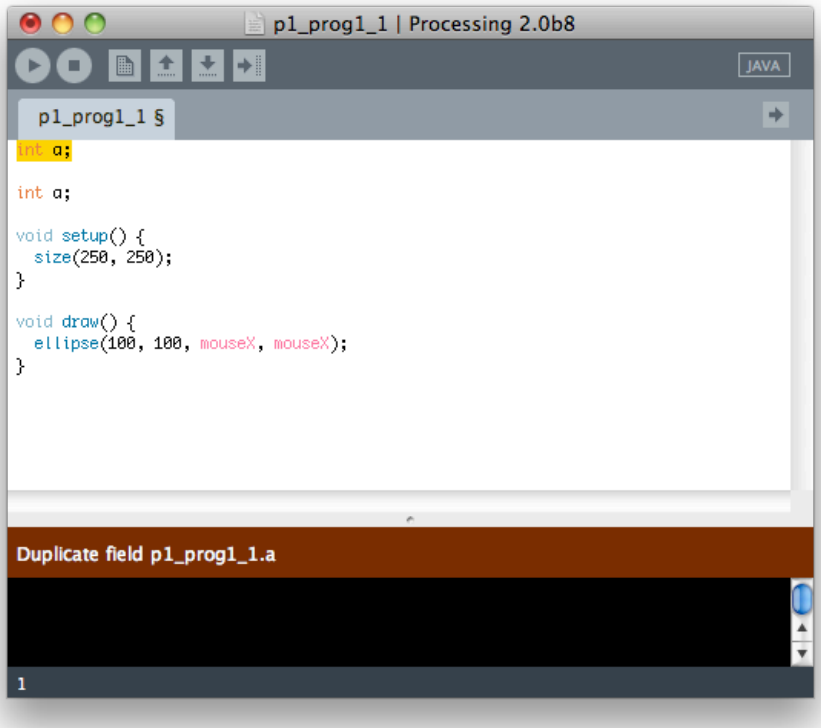


— Нам пишут: «Cannot find anything named «mousex». В переводе: «Не могу ничего найти с именем mousex». Видишь как! Вместо большой «X» написали маленькую «x» и уже такого монстрика-переменной не существует!

Ошибка шестая: попытка создания переменных с одинаковым именем

— Переменные не могут быть с одинаковым именем. Вот давай попробуем два раза подряд создать переменную с одним и тем же именем.

Тогда увидим такую ошибку:



— Пишет, что «дабликэйт фиэлд», что означает «дублирование поля»!

— Под «полем» он наверное имеет ввиду переменную. А что означает «p1_prog1_1.a», которое после этого дабликэйт фиэлд написано?

— Это название нашей программы, вон видишь, оно еще в самом верху на сером фоне повторяется!

Послесловие

Что осталось за кадром?

— Эх, Интик, замечательно мне было здесь с тобой в созвездии Большая Гигабайтица (в той, что рядом с Маленькой Гигабайтицей) в системе Процессинг! Столько нового я узнал, что даже не могу в это поверить! Но что-то я соскучился по дому.

— Так мы тебя мигом обратно доставим. Ты ведь еще прилетишь?

— Обязательно! И еще я подозреваю, что ты мне не все рассказал?

— О да! Самое интересное еще впереди! Это целый огромный подход к разработке наших программ, который называется Объектно-ориентированное программирование, если кратко, то ООП.

— ООП, ООП, ООП-а, процессинг-стайл! — вдруг пропел Яша.

— Чего-чего? — недопонял Интик.

— Это я так...

— А кроме ООП, еще я тебя научу, как сохранять очки в игре, даже когда она закрыта. Создадим таблицу рекордов! Для этого мы научимся работать с файлами! Узнаем, как самим создавать циклы, как хранить много-много переменных в одном маленьком-маленьком месте, что позволит нам создавать салюты, фейерверки и много еще чего красочного! А также научимся использовать в программе не просто фигуры, а самые настоящие картинки и фотографии! А может даже попробуем создавать 3D!

История изменений

Номер	Изменение	Дата
1.2	Мелкие правки текста. История изменений	19.09.2013
1.3	Стр.73 Изображение	01.11.2013
1.4	Рецензия, реквизиты	13.11.2013
1.5	Обложка, испр. нумерация страниц	21.11.2013
1.5.1	Испр. опечатка на 23 и 22 странице. Спасибо kostya2908, thorr	22.11.2013
1.5.2	Испр. ошибки на стр. 27 - спасибо Елена Андреева, Испр. ошибки на стр. 148 Испр. произношение: 39, 55, 145, 164, 202, 86, 91 - спасибо DenisK Стил. исправления: 145, 151, 162, 164, 173, 174, 175, 177, 184, 186 + во всем тексте окружение арифм. знаков - спасибо DenisK	25.11.2013
1.6	Испр. ошибки на стр. 143, 145 - спасибо Дмитрий Турченко Испр. ошибка на стр. 148	27.11.2013

ProgrammingForKids.ru

Игорь Грессус

История о том, как Яша учился программировать

или
программирование для школьников
на современном, общедоступном, простом для освоения, мощном языке
Processing (основан на языке Java)

версия 1.6

Первая в России книга по обучению детей младшего и среднего школьного возраста профессиональному языку программирования Процессинг, который основан на Java.

Это не обычная книга для начинающих, пестрящая всевозможными терминами и сложная для понимания детьми. А самая настоящая детская сказка со множеством иллюстраций, историями, примерами и метафорами.

Книга, которая полностью перевернет сложившееся представление об обучении детей программированию.

http://www.programmingforkids.ru
email: programmingforkids@yandex.ru

Основы программирования
для детей младшего и среднего
школьного возраста