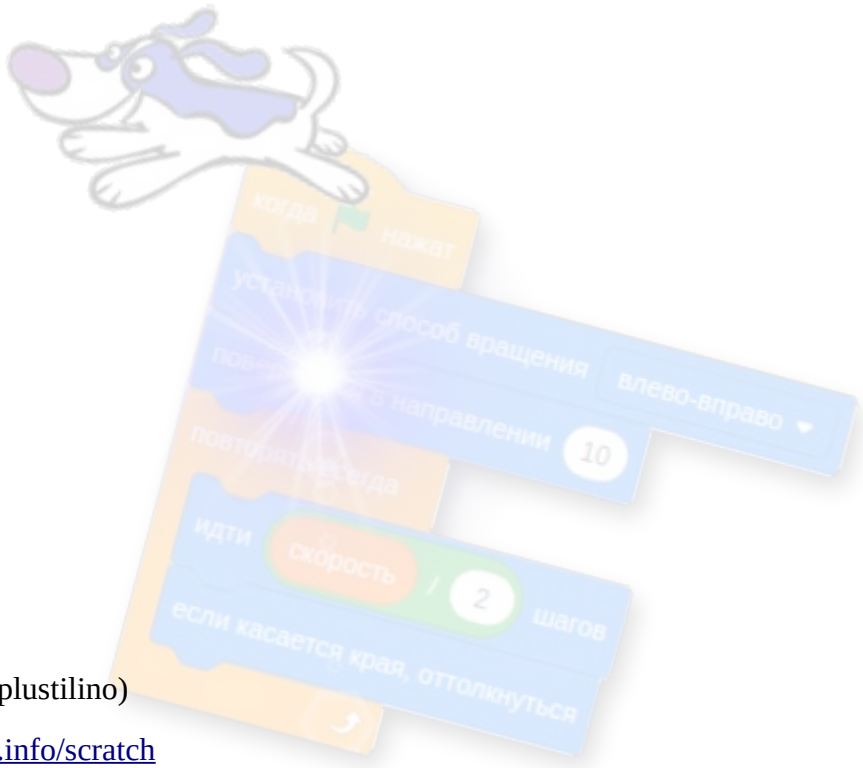
Программирование в Scratch. Курс

12 уроков
с ответами к заданиям



Автор: Светлана Шапошникова (plustilino)

Опубликовано: https://younglinux.info/scratch

Версия: март 2019

О Скретче

Scratch - это язык и среда программирования, ориентированные в первую очередь на обучение детей и подростков. Scratch-программист не пишет код, а собирает его из блоков как конструктор Lego.

В более ранних версиях Scratch был доступен в том числе оффлайн, т. е. распространялся установочный пакет, который можно было установить на компьютер. В начале 2019 года был выпущен Scratch 3.0, предполагающий работу онлайн на сайте проекта - https://scratch.mit.edu

О курсе

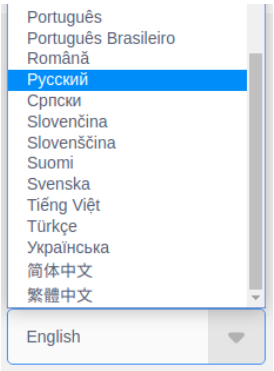
Курс "Программирование в Scratch" представляет собой цикл уроков по основам работы в среде Scratch. Также курс подспудно знакомит с базовыми концепциями основных парадигм программирования.

PDF-версия курса включает два дополнительных урока ("Списки" и "Функции"), а также ответы к заданиям.

Что такое Scratch

Scratch – это не только среда для программирования, это еще большое сообщество. Во многих странах мира дети и взрослые, учителя и школьники используют Scratch, чтобы учиться программированию, развивать творческие способности, создавать игры и интерактивные анимации, а также общаться между собой, изучать и использовать проекты друг друга.

Сайт сообщества и среды программирования находится по адресу https://scratch.mit.edu/ . Поскольку это общемировая социальная сеть, язык интерфейса сайта может оказаться английским. Чтобы поменять его на русский, надо прокрутить страницу вниз и в выпадающем списке выбрать русский язык.



Чтобы зарегистрироваться на сайте, надо нажать кнопку "Присоединиться", которая находится вверху сайта.

После этого на экране появится диалоговое окно, где будет предложено придумать себе псевдоним, пароль, заполнить другие поля. Также у вас должен быть адрес электронной почты. После регистрации на него придет письмо, кликнув по ссылке в котором, вы подтвердите свою учетную запись на сайте Scratch.

В заголовке сайта после логотипа Scratch есть четыре кнопки – "Создавай", "Исследуй", "Ideas" (Идеи), "О проекте".

Если кликнуть по "Создавай", откроется среда программирования Scratch. Как в ней работать, рассмотрим чуть позже. Если вы зашли сюда и не знаете как выйти, просто кликните по логотипу Scratch в верхнем левом углу. Это вернет вас на главную страницу сайта.

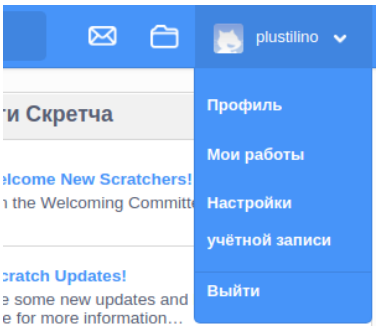
Если кликнуть по "Исследуй", вы перейдете на страницу, где можете посмотреть чужие проекты. Часть из них доступна также с главной страницы.

Чтобы посмотреть чужой проект, надо кликнуть по нему. Откроется страница, где слева будет сам проект, справа – его описание, внизу – комментарии. Проект будет в режиме исполнения, то есть вы смотрите готовую работу, а не то, как она запрограммирована и из каких блоков собрана.

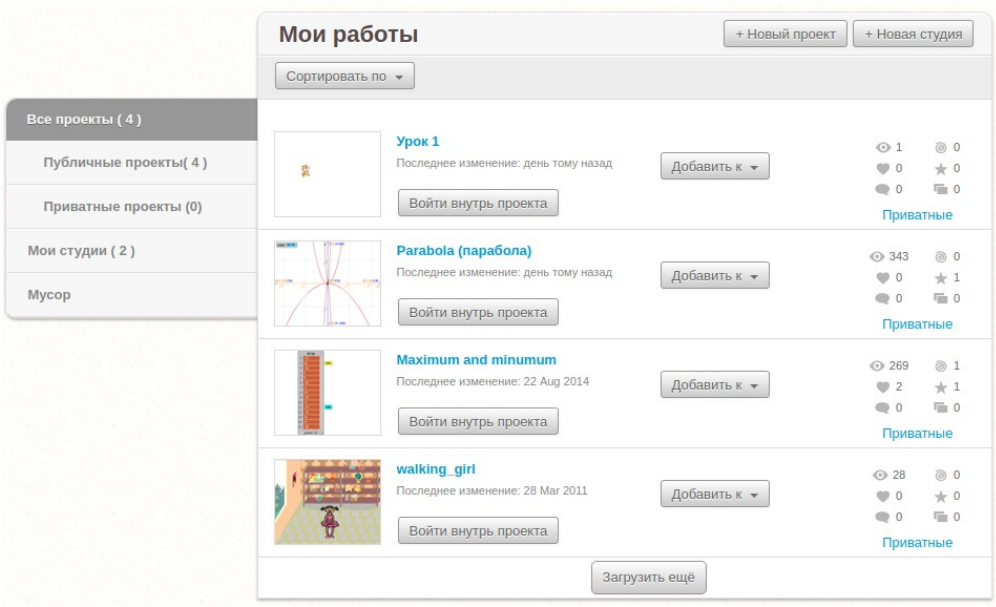
Чтобы запустить игру, надо нажать на зеленый флажок. Чтобы остановить – на красный кружок. Значок с четырьмя стрелками разворачивает игру на большую площадь экрана.

Если вы кликните по кнопке "Войти внутрь проекта", то окажетесь в среде программирования Scratch и сможете посмотреть внутреннее устройство проекта, его логику. На основе чужого проекта можно создать свой, то есть ремикс. Однако сначала надо научиться работать в среде программирования и создавать собственные простые проекты.

Вернемся к шапке сайта и рассмотрим меню справа:

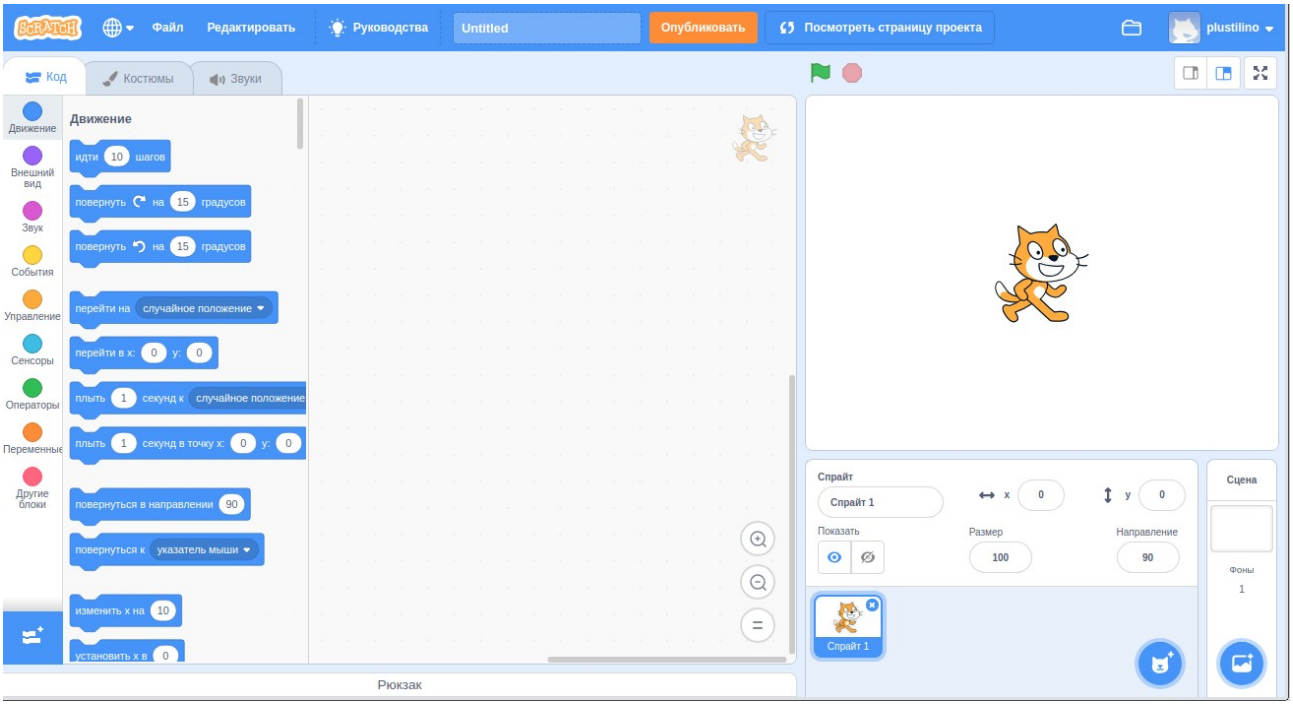


В первую очередь нас интересует ссылка "Мои работы". Она открывает страницу, с которой вам доступны собственные проекты. На эту страницу также можно перейти, кликнув по изображению папки в меню.



Чтобы начать новый проект, надо нажать кнопку "+ Новый проект". Откроется та же среда программирования, что при клике по ссылке "Создавайте" вверху. Если вы хотите править ранее созданный проект, нажимайте на кнопку "Войти внутрь проекта". А вот клик по названию проекта откроет его в режиме исполнения, то есть просмотра.

Что же, создадим новый проект и изучим интерфейс Скретча:



Если редактор открылся не на русском языке, нажмите глобус слева вверху и выберите русский язык.

По умолчанию проект назван Untitled (Неозаглавленный). В поле, где написано это слово, можно задать более осмысленное имя. Оно сохранится автоматически.

Если же нажать на кнопку "Опубликовать", ваш проект станет доступен всем для просмотра. Обычно сырые проекты не публикуют, и они остаются доступными только своим создателям.

В основной области экрана слева на вкладке "Код" расположены строительные блоки. Они задают поведение объектов, которые вы видите в области справа на игровом **холсте, или сцене**. Сейчас там только кот.

Большое пустое поле посередине – это самая важная область – **редактор кода**. Здесь вы "пишите" код, а точнее, конструируете его из блоков, которые перетаскиваете из левой части.

Строительные блоки слева разделены по секциям-разделам "Движение", "Внешний вид", "Звук" и так далее. Нажимая на соответствующий цветной кружок, вы быстро перейдете к блокам необходимой секции.

В этом уроке не будем рассматривать весь интерфейс среды программирования Scratch. Постепенно, изучая курс, мы познакомимся и с другими возможностями. А сейчас для создания первой программы достаточно уже полученных сведений.

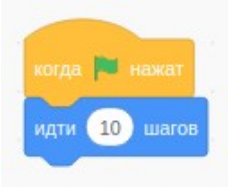
Возьмите блок "идти 10 шагов" и киньте его в редактор кода. Теперь пощелкайте по нему мышью. Вы увидите, как от каждого клика кот на сцене будет смещаться на небольшое расстояние.

Если кот на холсте далеко ушел от середины, его можно взять здесь мышью и перетащить в новое место холста.

По идее программа должна запускаться, когда кликают по расположенному надо холстом зеленому флажку, а не по блоку команды, как это делали мы. Если мы сейчас кликнем на флажок, то ничего не произойдет.

Чтобы исправить это, нам нужна команда "когда флажок нажат" (в команде вместо слова "флажок" будет изображение зеленого флажка). Эта команда находится в желтом разделе "События". Переключимся сюда и бросим в редактор кода "когда флажок нажат".

После этого два блока – "когда флажок нажат" и "идти 10 шагов" – надо соединить подобно элементам конструктора Лего.



Последовательность блоков важна. Команда, которая расположена выше, будет выполняться раньше, чем команда, которая расположена ниже. Часто конфигурация самих блоков подсказывает, как их надо соединять. Например, мы никогда не сможем сделать команду "когда флажок нажат" второй, третьей или какой угодно, кроме как первой. У этого блока нет выемки сверху, куда можно было бы вставить другой блок. Программа всегда начинается с команды "когда флажок нажат".

Теперь кот будет перемещаться при клике по зеленому флажку над холстом.

Однако от каждого клика кот по прежнему шагает всего где-то на 10 пикселей, и никакой нормальной анимации перемещения мы не видим.

Обратите внимание, что в команде "идти 10 шагов" поле с числом редактируемое. Туда можно записать любое число. То есть команда может выглядеть так: "идти 100 шагов". Или так: "идти -25 шагов". Поэтому правильно упоминать эту команду так: "иди … шагов", где вместо троеточия подразумевается произвольное число.

Мы могли бы предположить, что если вместо числа 10 в команде "идти 10 шагов" вписать число больше, то кот будет перемещаться на большее расстояние. Правильно. Код будет перемещаться. Однако анимации все-равно не будет. Кот будет делать гигантский прыжок от каждого клика по флажку.

Поэтому вернем наше заветное число 10, а может быть даже 5 или 1.

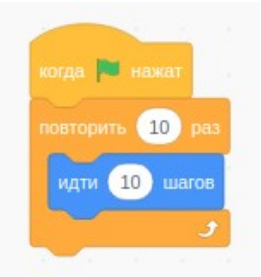
Давайте вернемся к реальности и подумаем, как мы на самом деле ходим. По-сути мы много раз делаем одно и тоже: шаг левой, шаг правой, шаг левой, шаг правой. Мы зацикливаем шаг, повторяем его много раз. Правильнее сказать: "мы шагаем в цикле".

Аналогично делается в Scratch. Следует поместить команду "идти … шагов" в какой-нибудь цикл. Циклы – это тоже строительные блоки и одно из важнейших понятий программирования. Циклам стоит посвятить отдельный урок. Однако сейчас не будем теряться, зайдем в оранжевый раздел "Управление" и перекинем в редактор кода блок "повторить 10 раз".

Однако куда его присоединять? Если мы подставим блок цикла снизу к команде "идти … шагов", это будет неправильно. В программировании логика важна. Получится, что кот сначала сделает шаги, а потом будет 10 раз повторять непонятно что. На самом деле ничего, так как тело цикла пусто, то есть блок "повторить … раз" ничего не обрамляет, ничего в себя не включает.

Программу надо пересобрать. Для этого сначала надо отсоединить блок "идти … шагов" от "когда флажок нажат". Чтобы разорвать связь между блоками, надо потянуть за нижний блок.

Теперь поместим команду "идти 10 шагов" внутрь цикла "повторить 10 раз", а цикл присоединим к блоку "когда флажок нажат".



В данном случае кот сделает в общей сложности 100 шагов, так как 10 раз умножить на 10 шагов будет 100 шагов. Однако эта программа отличается, если бы мы просто дали команду "идти 100 шагов". В случае цикла перед каждым его повторением будет небольшая невидимая на глаз задержка, в результате чего будет наблюдаться относительно плавная анимация.

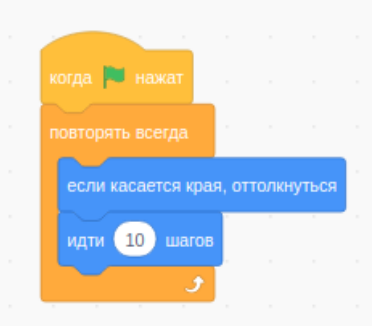
Анимацию можно сделать более плавной, если установить меньшее число шагов и большее число повторений. Например, 1 шаг и 200 повторений. Но в этом случае кот будет двигаться медленно. Таким образом можно регулировать скорость перемещения объекта: чем больше шагов он делает за один повтор цикла, тем больше его скорость.

Заменим в нашей программе цикл "повторить … раз" на "повторять всегда". Эта команда также находится в оранжевой секции "Управление". Запустим программу. Что произойдет, когда кот дойдет до края экрана? Он остановится. Программа продолжает работать, но кот уже никуда не идет.

Чтобы принудительно остановить программу, надо нажать на красный кружок над холстом.

Если мы хотим, чтобы кот действительно двигался всегда, как минимум до тех пор, пока мы сами не остановим программу, следует добавить еще одну команду – "если касается края, оттолкнуться". Этот блок находится в синем разделе "Движение".

Команду "если касается края, оттолкнуться" следует поместить в цикл. А вот до команды "идти 10 шагов" или после, не столь важно. Кот сначала может сделать 10 шагов, а потом, если коснулся края холста, то перевернуться. Или же он сначала проверит, не касается ли края холста, после чего перевернется.



Когда вы запустите такую программу, кот будет ходить от левого края к правому без конца, до тех пор, пока вы сами не остановите программу. Однако скорее всего ваш кот будет ходить немного странно: влево идет ногами, а как направо – становится на голову.

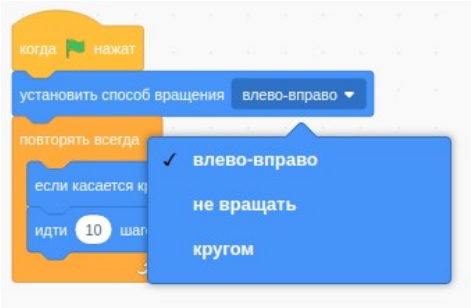
Дело все в том, что в Scratch по умолчанию объекты при перевороте вращаются по кругу. Что это значит? Вы должны знать, что полная окружность составляет 360 градусов. Когда же мы переворачиваемся в обратную сторону, то делаем разворот на половину окружности, то есть на 180 градусов.

А вот от того, как мы делаем разворот, то есть как осуществляем сам процесс поворота, результат может быть различным. Представьте, что вы смотрите влево. Потом резко поворачиваете вправо вокруг своей вертикальной оси, которая проходит от макушки до пяток. Это естественный разворот для человека, так что все нормально. Вы продолжаете стоять на ногах, просто смотрите в другую сторону.

Теперь представьте, что вы подтянулись на перекладине так, что ваши руки и перекладина находятся на уровне пояса. И вы решили перевернуться на 180 градусов вокруг этой перекладины. После этого ваша голова окажется внизу. Смотреть же вы будете туда, куда и хотели: если смотрели влево, будете смотреть вправо.

Примерно тоже самое происходит с нашим виртуальным котом. Однако мы можем поменять это поведение по умолчанию с помощью команды "установить способ вращения влево-вправо". Она находится в разделе "Движение". Команду достаточно выполнить один раз, так что место ей до цикла.

Эта команда многовариантная. Вместо влево-вправо можно выбрать "кругом" или "не вращать". Кругом мы уже видели, так ведут себя объекты по умолчанию. Если выбрать "не вращать", объект при отталкивании никак не будет вращаться. Он будет двигаться задом, когда натолкнется на край холста передом, и будет двигаться передом, когда натолкнется на край задом.



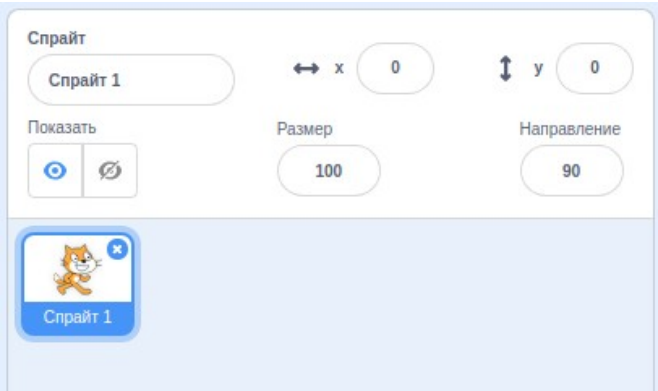
На этом закончим наш первый урок, целью которого было познакомиться со Скретч, понять основные принципы работы в среде программирования. В следующих уроках мы будем изучать другие команды языка и особенности программирования.

Один из главных моментов, которые вы должны понять на этом этапе, – команд существует много, а уж их комбинаций – бессчетное количество. Поэтому программирование – это не только наука, а также искусство и творчество.

Задания

Задание 1.

В уроке мы не обсудили часть интерфейса среды Scratch, которая находится под холстом и в которой задаются свойства объекта.



Изучите представленные здесь поля самостоятельно. Что определяют их значения? Подумайте, почему важно задавать объектам, которые в Scratch по умолчанию называются Спрайтами, осмысленные имена.

Задание 2.

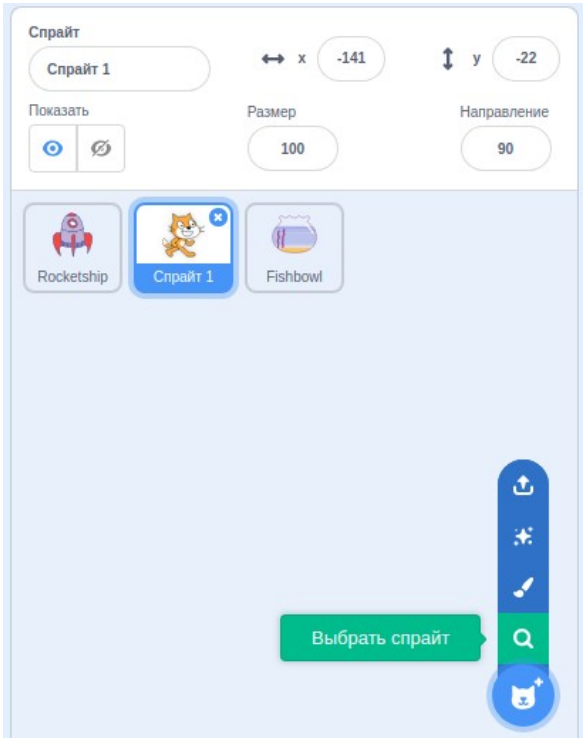
Составьте программу, согласно которой кот ходит туда-сюда, но через каждые 300 шагов останавливается на 1 секунду и говорит "Мяу". Подсказка: среди прочего вам понадобятся две команды, которые не рассматривались в этом уроке. Одна из них находится в разделе "Управление", а вторая – в разделе "Звук".

Спрайты

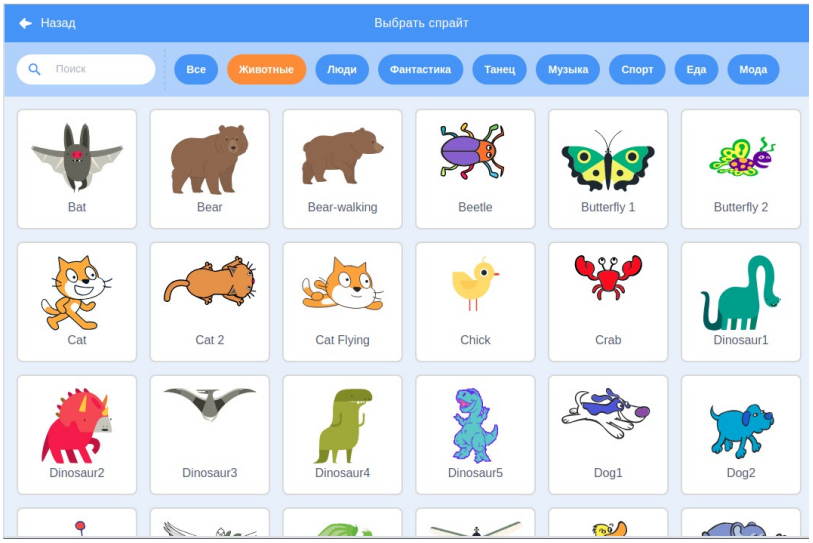
Слово "спрайт" часто используется в программировании игр для обозначения фигурок героев и предметов. Такие фигурки – это на самом деле рисунки или фотографии, то есть компьютерная графика.

Однако спрайт – не только картинка, это объект, наделенный программным кодом и как следствие свойствами и способностями. Со спрайтом можно взаимодействовать как с виртуальной сущностью, отдавать ей команды и получать от нее данные.

Кот в Scratch – не единственный объект-спрайт. Добавить на сцену другие можно несколькими способами: загрузить картинку с компьютера, нарисовать в самой среде программирования, выбрать из библиотеки. Для всего этого в Scratch предусмотрено специальное меню, которое находится внизу справа на панели спрайтов:



Пока ограничимся тем, что будем выбирать готовые спрайты из библиотеки. Для этого надо нажать лупу. Откроется библиотека, где можно найти подходящего героя.

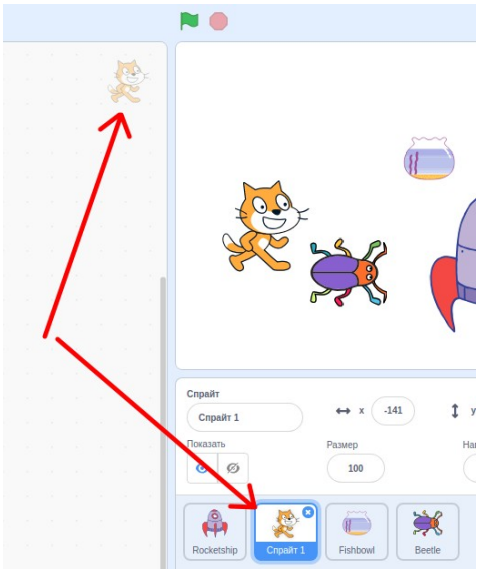


После клика по нему он появится на сцене. Также на панели спрайтов под свойствами появится его иконка. Рядом с иконкой кота. Свойства объекта, где вы задаете ему имя, положение, размер и поворот, отображаются только для того спрайта, который выделен.

Если вы захотите удалить один из спрайтов, кликните по его иконке под свойствами. Этот спрайт станет выделенным, выбранным, иконка обрамится синим цветом и у нее появится крестик, кликнув по которому можно убрать спрайт из игры.

Когда в Scratch мы собираем блоки в программный код, этот код не существует в прострации сам по себе. Чаще всего он относится к какому-то спрайту. Другими словами, у каждого спрайта свой программный код, своя программа, свое предназначение. Герои могут "общаться" между собой через сигналы, которые посылают друг к другу. Но это другая история.

Сейчас надо запомнить, что когда мы работаем с блоками, следует обращать внимание на то, какой объект выделен. В Scratch появляется подсказка в верхнем правом углу поля редактора кода. Там бледным цветом отображается фигурка выделенного на данный момент спрайта. А значит весь код, который вы видите в рабочей области, относится только к данному герою.



Когда выделяется другой спрайт, то код предыдущего становится невидимым.

Нередко код одного спрайта оказывается похожим на другой. Например, оба героя ходят по сцене, просто с разной скоростью и в разные направления. Собирать два раза почти одинаковую комбинацию блоков – не слишком удачное решение. Проще скопировать программу одного спрайта, добавить ее другому, а потом немного изменить.

В Scratch копировать собранные в программу блоки можно как в пределах одного проекта, так и переносить копию в "рюкзаке" из одного проекта в другой. Рюкзак находится внизу интерфейса среды разработки.

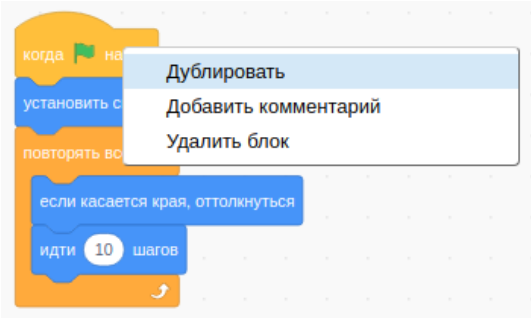
Допустим, у нас есть проект под названием "Проект 1", где мы составили программу движения кота туда-сюда. Мы хотим скопировать этот код в "Проект 2". Тогда мы заходим в первый проект, открываем рюкзак и перетаскиваем сюда конструкцию блоков.

При этом надо помнить правило: переносится участок кода от верхнего блока, за который мы держимся, и блоки, расположенные ниже него. Если вы взялись за блок, расположенный где-то в середине программы, то верхние блоки оторвутся и не будут захвачены.

Положив в рюкзак программу, мы возвращаемся во второй проект. Выделяем спрайт, которому хотим добавить код. Открываем рюкзак и достаем из него нашу программу.

Чтобы скопировать программу одного спрайта на другой в пределах одного проекта, достаточно захватить код и бросить его на иконку другого спрайта, которая расположена под свойствами на панели спрайтов.

Если надо скопировать код в пределах одного спрайта, надо кликнуть по конструкции правой кнопкой мыши и в контекстном меню выбрать "Дублировать".



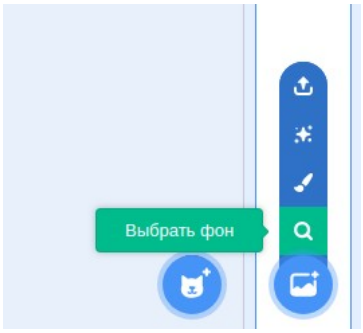
Также обратите внимание на пункт "Добавить комментарий". Программисты часто пользуются комментариями, чтобы пояснить себе и остальным, что делает та или иная часть программы. Ведь при взгляде на сложный код не всегда сразу можно понять, для чего он, и как все это работает.

Когда на сцене обитают несколько спрайтов, становится важным их положение относительно друг друга, их место на сцене. Как вы уже должны были понять, изучая панель свойств спрайта, положение героя задается координатами, то есть точкой на плоскости.

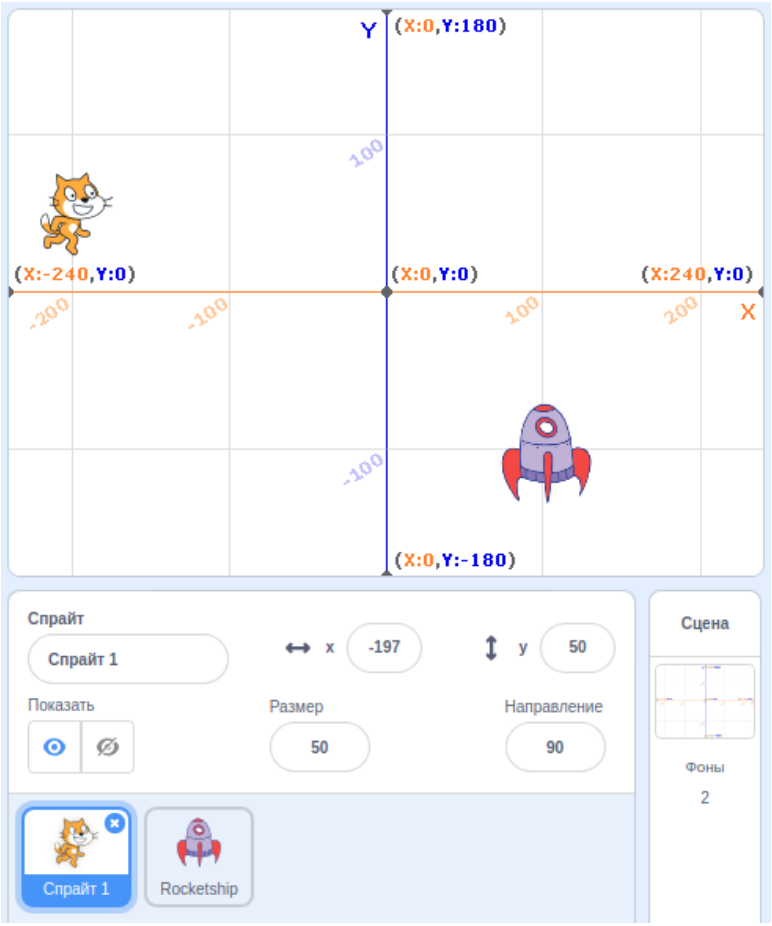
В Scratch начало отсчета, то есть точка с координатами (0; 0), находится в центре сцены. Общая ширина сцены составляет 480 точек. Это значит, что координата *x* может принимать значения от -240 до 240. Общая высота сцены составляет 360 точек. Это значит, что координата *y* может принимать значения от -180 до 180.

Хотя работу с фонами мы будем изучать позже, добавим на сцену координатную сетку. Это поможет нам сориентироваться в системе координат Скретча. Чтобы выбрать фон из

библиотеки, надо нажать на лупу в меню сцены, которое находится в нижнем правом углу окна, на панели "Сцена".



Откроется библиотека фонов, прокрутите ее в самый низ и выберите фон "Xy-grid". На сцене появится координатная сетка.



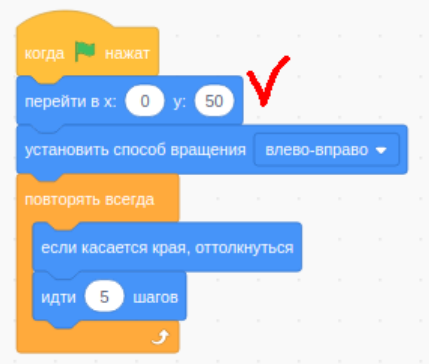
Давайте уменьшим размер наших спрайтов раза в два, то есть примерно до 50%, чтобы они не занимали большую площадь сцены, а больше походили на точку. Теперь поиграем с их координатами. Для этого можно менять значения *x* и *y* на панели свойств и смотреть, где после этого окажется герой. Или просто перетаскивать мышью спрайт на сцене и смотреть как меняются *x* и *y* в свойствах.

Например, если x = 0, y = 150, спрайт окажется в центре по горизонтали и вверху по вертикали. Если x = 200, y = 0, то спрайт окажется справа по горизонтали и в центре по вертикали.

На самом деле не обязательно до запуска программы устанавливать спрайты в нужное место с помощью мыши или панели свойств. Обычно это делают программно, с помощью

специальных блоков кода, находящихся в разделе "Движение". Одним из управляющих местоположением блоков является команда "перейти в x: … y: …". Вместо точек указываются желаемые координаты.

Эта команда быстро перебрасывает спрайт в заданную точку, поэтому ее часто используют в начале программы:

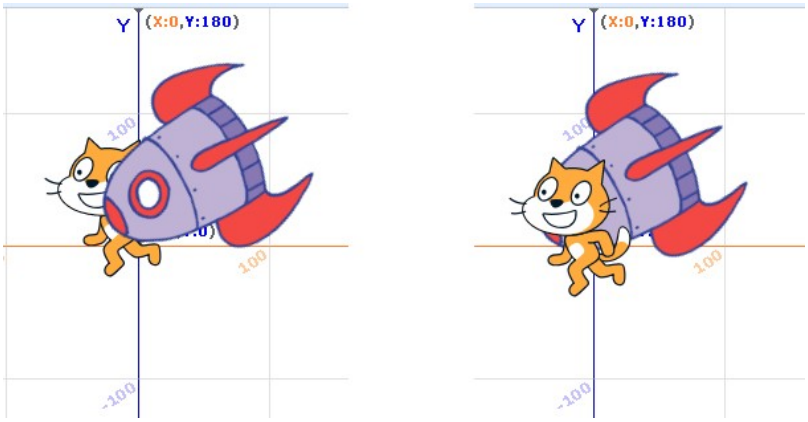


В приведенном примере, где бы ни стоял на сцене спрайт до запуска программы, как только будет нажат зеленый флажок, герой тут же окажется в точке (0; 50).

Отметим напоследок про слои. Наше пространство не двумерное, как плоскость, а трехмерное. У нас есть высота, ширина и длина. Хотя Scratch позволяет создавать только двумерные анимации, на самом деле даже в 2D-анимации есть своего рода третье измерение. Это слои.

Пусть по сцене перемещаются два спрайта. Что будет когда их координаты совпадут? Ничего. Они не столкнуться и не затормозят друг друга. Один спрайт пройдет как бы позади другого, а другой – поверх первого. Каждый живет на своем слое плоскости и ему нет дела до другого.

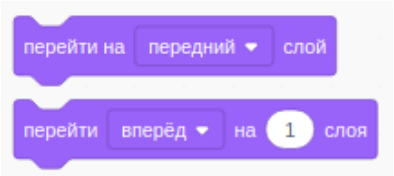
Однако бывает важно, слой какого спрайта расположен выше, то есть впереди. Посмотрите на рисунок:



Слева слой ракеты расположен выше, слоя кота. Справа, наоборот, кот находится над ракетой.

В Scratch изменить очередность слоев можно как мышкой на сцене, так и с помощью блоков кода. Чтобы это сделать первым способом, надо взять тот спрайт, который должен быть над, и кинуть его на спрайт, чей слой должен быть под.

Блоки изменения очередности слоев находятся в фиолетовой секции "Внешний вид". Это команды "перейти на передний/задний фон" и "перейти вперед/назад на … слоя".



Задание

Составьте программу, в которой один спрайт постоянно ходит за указателем мыши, а второй – находится в центре сцены и, стоя на одном месте, медленно крутится по часовой стрелке. Слой первого объекта должен быть над слоем второго.

Для выполнения задания потребуются две дополнительные команды – "повернуться к указатель мыши", "повернуть по часовой стрелке на … градусов" (вместо слов "по часовой стрелке" на блоке изображена дуговая стрелка).

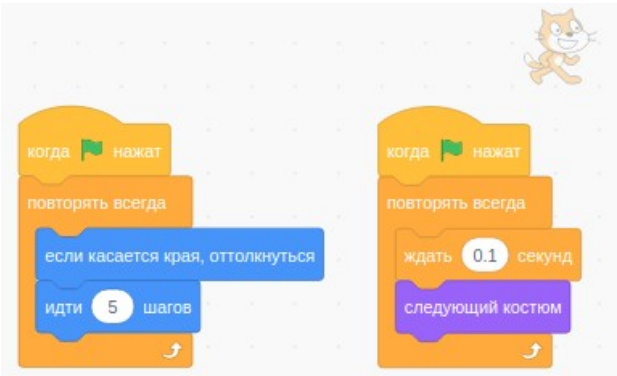
Последовательное и одновременное выполнение скриптов

В прошлых уроках мы соединяли блоки команд в конструкции, которые можно называть маленькими программками. Также их можно назвать **скриптами**. В одном проекте может быть множество скриптов. Например, если есть два героя-спрайта, вероятно у каждого будет свой скрипт, своя маленькая программа. А весь проект – это большая общая программа, состоящая из множества мелких.

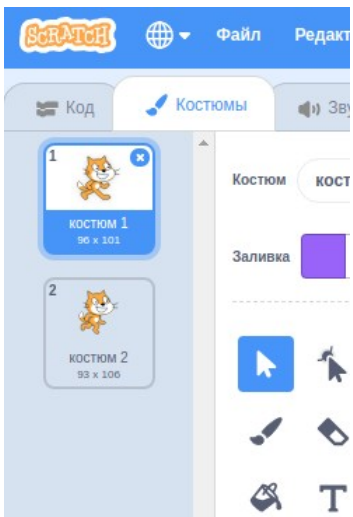
В программировании отдельные части большой программы могут выполняться либо по очереди друг за другом, то есть последовательно, либо одновременно друг с другом, то есть параллельно. Во втором случае говорят о многопоточности, то есть каждый скрипт работает в своем потоке, а разные потоки текут во времени одновременно.

Когда для каждого спрайта одного проекта вы добавляете свою конструкцию блоков, которая начинается с команды "когда флажок нажат", то как только игра запускается, оба героя начинают параллельно независимо друг от друга выполнять свои команды. Это пример одновременного выполнения скриптов.

Более того, один и тот же спрайт может содержать два независимых скрипта, каждый из которых будет выполняться в своем потоке. Рассмотрим пример. Пусть кот ходит по сцене и при этом меняет свой костюм.



В Scratch многие спрайты на самом деле состоят не из одного изображения, а из нескольких. Это можно увидеть на вкладке "Костюмы".



Поэтому когда выполняется команда "следующий костюм", происходит переключение на следующее изображение спрайта. Если костюмов всего два, то они будут чередоваться.

В приведенном выше примере блоков кода оба скрипта будут выполняться одновременно. Мы увидим, что кот не только перемещается, но еще двигает ногами. Этот эффект возникает от того, что изображения-костюмы быстро меняются.

Если отсоединить блок "когда флажок нажат" от первого скрипта, то при запуске игры кот будет стоять на месте, будут двигаться только его ноги.

Если же мы отсоединим блок "когда флажок нажат" только от второго скрипта, то кот будет перемещаться, но его ноги не будут двигаться.

Таким образом, каждый скрипт вносит свой вклад и работает независимо от другого, но одновременно с ним.

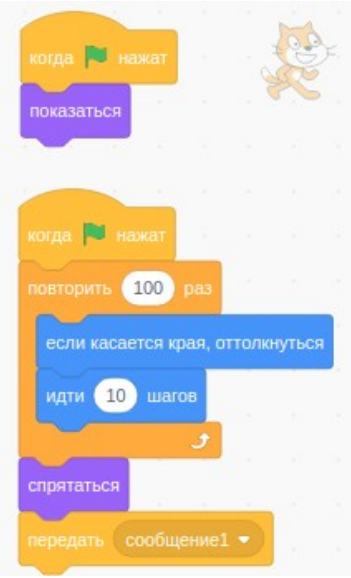
Вы спросите, почему нельзя было вставить команду "следующий костюм" в цикл первого скрипта и вовсе отказаться от второго? На самом деле можно. Однако в этом случае кот начнет менять костюмы так часто, что у вас зарябит в глазах. Тогда почему бы сюда же не добавить команду "ждать … секунд"? Потому что в этом случае, кот будет останавливаться, и плавного движения не получиться. Движение будет урывистым, перед каждой сменой костюма будет возникать задержка.

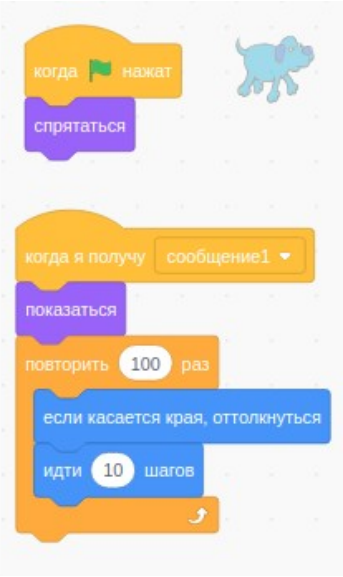
Теперь поговорим о последовательном выполнении скриптов. На самом деле такое поведение встречается чаще хотя бы потому, что команды в одном скрипте выполняются последовательно. А чтобы друг за другом выполнялись разные скрипты, один из них должен вызывать другой, то есть передавать своего рода сообщение.

Также бывает, что скрипт начинает выполняться только при определенных условиях. Например, когда нажимается одна из клавиш на клавиатуре. При этом работа других скриптов может как останавливаться, так и не останавливаться. Во втором случае это уже не может считаться последовательным исполнением.

Давайте рассмотрим вариант передачи сообщения, когда один скрипт вызывает другой. Пусть у нас будет два спрайта – кот и собака. Сначала кот ходит по сцене. И только когда он останавливается и исчезает, на сцене появляется собака и начинает ходить.

Поскольку при запуске игры один из спрайтов должен быть видимым, а другой невидимым, воспользуемся командами "показаться" и "спрятаться". Вынесем их в отдельные скрипты, чтобы они не мешались в основной логике игры.





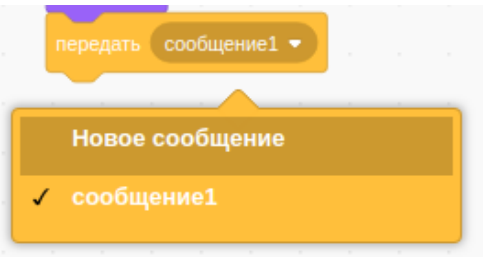
Теперь когда кот выполнит свою порцию шагов, с помощью блока "передать сообщение1" он пошлет сообщение, которое может перехватить любой объект программы. В данном случае сообщение будет ждать только собака, потому что только ей мы добавили команду "когда я получу сообщение1".

И как только она получит сообщение, она покажется и начнет ходить.

В итоге получится, что скрипты, отвечающие за ходьбу кота и собаки, будут выполняться последовательно, один за другим.

Команда "передать сообщение" и "когда я получу сообщение" могут передавать любое сообщение, а не только "сообщение1". В проекте могут циркулировать десятки различных сообщений, которые одни объекты посылают, а другие ждут.

Чтобы добавить новое сообщение, надо кликнуть по маленькому треугольнику в этих командах и в раскрывающемся списке выбрать "Новое сообщение".



После этого на экране появится диалоговое окно, куда вводится сообщение. Например, "кот вызывает собаку".

Когда сообщение создано, оно становится доступным через раскрывающийся список команд "передать …" и "когда я получу …", где вместо точек может стоять любое сообщение.

При этом надо понимать, что если, например, кот передает сообщение "кот вызывает собаку", то собака должна получать именно это сообщение, а не какое-либо другое. Иначе она не будет реагировать.

Задание

Составьте программу по следующему описанию:

1. Кот недолго ходит по сцене, потом останавливается, но не исчезает.
2. Появляется второй спрайт и тоже недолго ходит по сцене.
3. Когда второй спрайт останавливается, кот снова начинает недолго ходить по сцене.
4. Когда кот останавливается во второй раз, на сцене появляется третий недолго бегающий спрайт.

При выполнении задания также воспользуйтесь блоком "передать сообщение и ждать до конца". Эта команда останавливает текущий скрипт до тех пор, пока не выполнится скрипт, который получает сообщение.

Условный оператор "если …"

В программировании важное место занимает **условный оператор** "если". С его помощью в программах организуется ветвление, поэтому его также называют **оператором ветвления**. Это значит, что при определенных условиях программа выполняется одним способом, а при отсутствии этих условий или наличии других условий программа выполняется другим способом, то есть идет по другому пути, другой ветке.

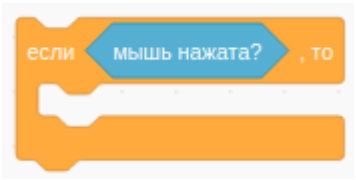
Допустим, в Scratch кот будет идти только в том случае, если нажата клавиша мыши. Если условие с мышкой не выполняется, кот никуда не пойдет. Чтобы реализовать такой сценарий, понадобиться блок "если … , то", который находится в оранжевом разделе "Управление".

В этот заголовок этого блока вставляется другой блок особого типа – **логическое выражение**. Такие блоки как бы что-либо спрашивают. И ответом на их вопрос может быть либо "да", либо "нет". Никаких "не знаю" или "надо подумать". Только да или нет.

Также можно представить, что логические выражения не спрашивают, а просто что-то утверждают. При этом то, что они утверждают, либо правда, либо ложь.

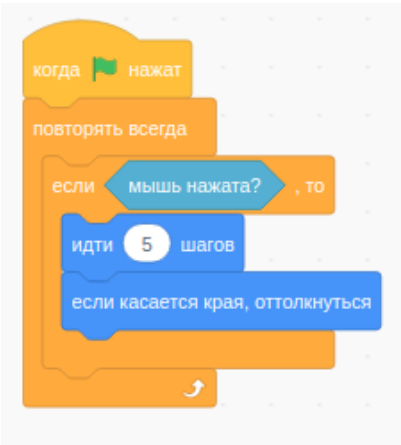
Блоки логических выражений не находятся в одном разделе. Их можно найти среди блоков как бирюзового раздела "Сенсоры", так и зеленого раздела "Операторы". В Scratch логические блоки имеют форму с острыми углами справа и слева.

В данном случае нам нужен блок "мышь нажата?", который мы вставляем в поле после слова "если" в оператор ветвления "если … , то".



Теперь, если клавиша на мышке будет зажата, будут выполняться команды, которые обрамляет блок "если мышка нажата, то". Но если клавиша на мышке не будет зажата, то команды внутри блока условного оператора выполняться не будут.

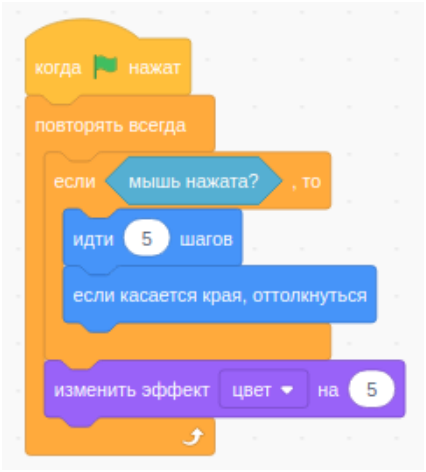
Сейчас внутри оператора ветвления, то есть в его теле, ничего нет. Поместим сюда команды движения, а сам блок "если" вставим в цикл "повторять всегда", чтобы программа работала всегда, пока мы ее сами не остановим.



В результате выполнения данного скрипта всегда будет проверяться, нажата ли какая-нибудь клавиша мышки. И если она нажата, кот будет перемещаться. Когда клавиша мыши будет отпускаться, спрайт будет останавливаться, потому что условие "мышка нажата?" станет неправдой.

А что будет, если мы поместим команды до или после всего блока "если". Влияет ли на них выполнение условия нажатия мышки? Абсолютно нет. Условие влияет только на те команды, которые находятся внутри "если". Команды за пределами "если" будут выполняться всегда.

Разместим команду "изменить эффект цвет на … " после блока "если".

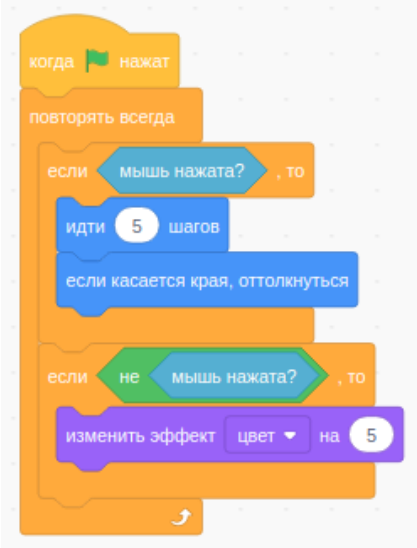


Теперь кот будет мигать разными цветами независимо от того, стоит он или двигается. Блок изменения цвета находится за пределами условного оператора, а значит выполняется всегда.

Но допустим, мы хотим, чтобы кот изменял свой цвет только тогда, когда стоит. Когда же он двигается, цвет не должен меняться.

Эту задачу можно решить двумя способами. Добавить второй блок "если", в котором проверять обратное условие, то есть то, что мышка не нажата. Или же, что более правильно, воспользоваться условным оператором с двумя ветками, то есть блоком "если … иначе".

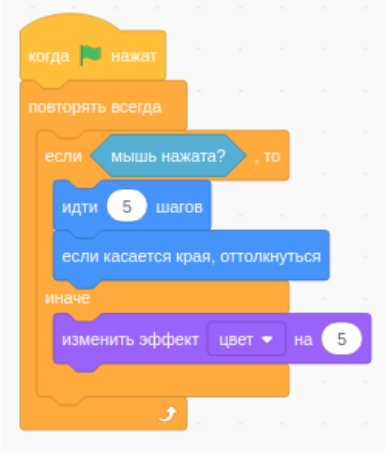
Рассмотрим сначала первый, не совсем верный, вариант:



Здесь мы используем новый для нас зеленый блок "не …", в который вставляем "мышка нажата?". **Сложное логическое выражение** "не мышка нажата?" помещаем в заголовок "если".

Таким образом, второй "если" проверяет, что мышка не нажата. И если она не нажата, то будут выполняться команды тела второго "если". В данном случае это одна команда изменения цвета.

Рассмотрим второй вариант решения задачи, который делает то же самое что первый, но использует блок "если … иначе":



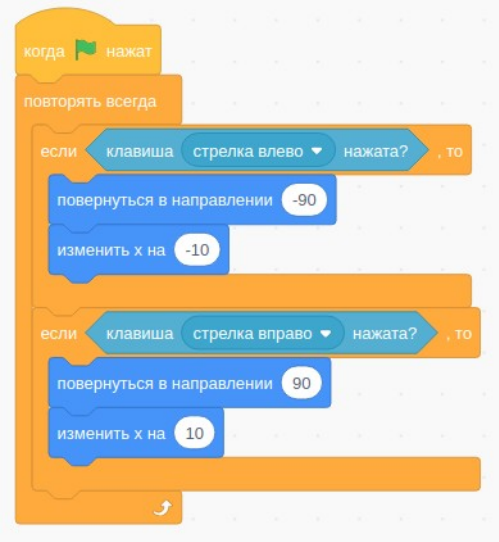
Этот вариант лучше первого не только потому, что содержит немного меньше команд. Важна логика. И здесь она более правильная.

Команды, вложенные в ветку "иначе" выполняются только тогда, когда условие при ветке "если" оказывается ложным. В данном случае если выражение "мышь нажата?" неправда, то будут выполняться команды внутри ветки "иначе".

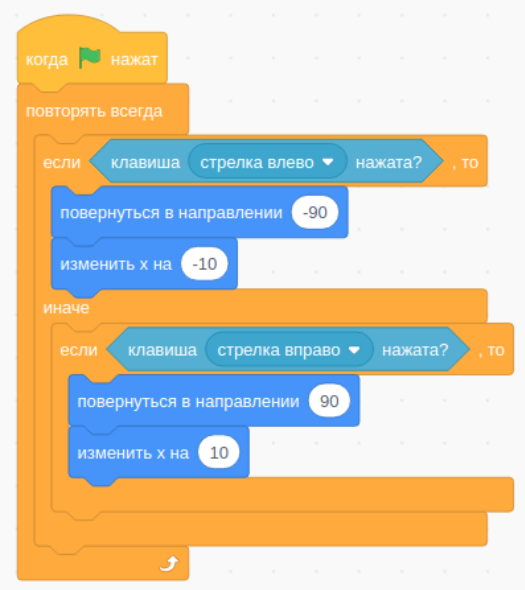
Программе не надо два раза проверять условие нажатия мыши, как в первом варианте. За один оборот цикла она проверяет его единожды, в заголовке "если … иначе". И если условие правдиво, будут выполняться команды внутри ветки "если". Если условие оказывается ложным, то программа сразу без дополнительных проверок пойдет выполнять команды в ветке "иначе".

Выполняется либо то, либо другое. Никогда и то и другое. Не может быть, чтобы мышка одновременно была нажата и отпущена.

А как быть, если мы хотим, чтобы кот двигался налево, когда мы нажимаем клавишу налево. И кот двигался бы направо, когда мы нажимаем клавишу направо. Можно решить задачу, используя два отдельных "если":



Однако мы также можем вложить второй "если" в ветку "иначе":



В данном случае в Scratch лучше работает первый вариант. Здесь если вы зажмете сразу две клавиши, кот никуда не будет идти, потому что шаг влево гасится шагом вправо. Во втором варианте, кот всегда будет предпочитать левую строну, так как в случае зажатия двух клавиш до ветки "иначе" поток выполнения просто не доходит.

Задание

Составьте программу с говорящим котом:

1. Если зажимается клавиша 1 на клавиатуре, кот говорит "Один".
2. Иначе, если нажимается клавиша 2 на клавиатуре, кот говорит "Два".
3. Иначе, если нажимается клавиша 3 на клавиатуре, кот говорит "Три".
4. Когда не нажимается ни одна клавиша клавиатуры, кот должен молчать.

Чтобы кот говорил, используйте команду "сказать Привет!" из фиолетового раздела "Внешний вид".

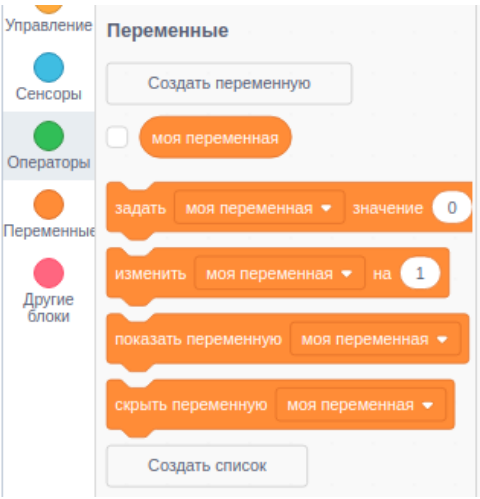
Переменные

Переменные настолько важны в программировании, что оно без них немыслимо. Ведь несмотря на то, что программа – это в первую очередь алгоритмы и логика, они не могут обрабатывать пустоту. Алгоритмы работают с данными, которые хранятся в памяти компьютера и к ним надо как-то обращаться, связываться с ними.

Как раз эту задачу решают переменные. Переменная состоит из имени и значения. Значение – это и есть данные. Через имя переменной мы обращаемся к ячейкам памяти, чтобы получить оттуда данные или изменить их.

Данными может быть любая информация, любой объект. Например, число, слово, спрайт, список. В этом уроке мы будем связывать имена переменных только с числами.

В Scratch ярко-оранжевый раздел "Переменные" содержит несколько блоков:



В программе может быть множество переменных. Поэтому есть кнопка "Создать переменную", при нажатии на которую открывается диалоговое окно, куда надо ввести имя переменной и выбрать, будет ли она доступна всем спрайтам или только текущему. Во втором случае созданная переменная не будет видна остальным спрайтам. Ей сможет использовать только один спрайт.

После создания переменную можно выбирать в выпадающих списках блоков раздела "Переменные".

Имя переменной может быть любым. Однако желательно, чтобы оно было описательное. Например, если переменная предназначена для хранения количества мячей, ее лучше так и назвать: "количество мячей" или просто "мячи".

На изображении выше мы видим, что сначала в Scratch есть только одна переменная под именем "моя переменная". Если около нее установить флажок, то ее имя и значение отобразятся на сцене.

Если в процессе выполнения программы требуется то показывать на сцене переменную, то скрывать ее, следует использовать блоки "показать переменную …" и "скрыть переменную …".

Команда "задать <переменной> значение …" записывает в указанную переменную те или иные данные. Например, число 10.

Команда "изменить <переменную> на …" в случае чисел изменяет значение переменной на указанную величину. То есть, если переменная содержала число 10, а мы дали команду изменить ее на 2, переменная будет хранить значение 12, а не 2.

Если же переменная содержала 10, а мы хотим присвоить ей значение 2, надо воспользоваться командой "задать <переменной> значение …".

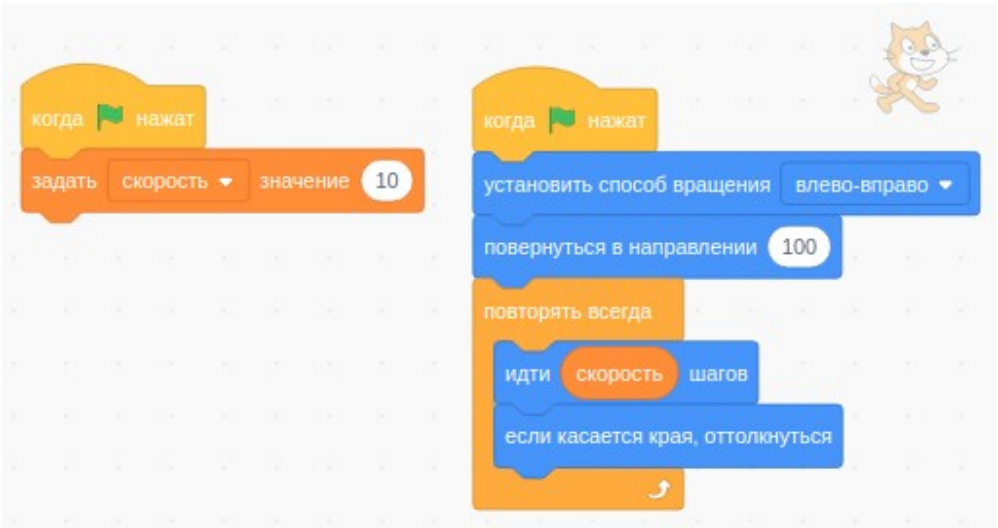
Рассмотрим как работать с переменными на примере. Пусть в нашем проекте по сцене ходят два спрайта – кот и краб. Ходить они в разных направлениях, но с одинаковой скоростью.

Мы пока не знаем точно, какую общую для спрайтов скорость хотим задать. Нам надо протестировать программу при разных скоростях. Если мы будем просто вписывать число в команду "идти … шагов", придется потом изменять ее два раза, у каждого спрайта. Это не совсем удобно, особенно если бы спрайтов было много. Для кого-то можно забыть изменить величину.

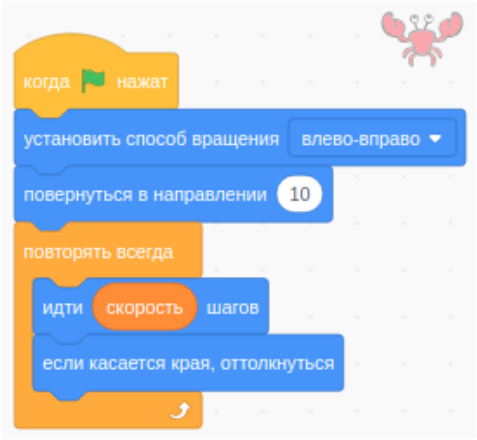
Использование переменной решает эту проблему. Достаточно задать ей значение в одном месте, а потом вставить имя переменной во все места, где она используется.

Создадим переменную под именем "скорость", доступную для всех спрайтов. Для этого надо нажать кнопку "Создать переменную" или переименовать существующую переменную, кликнув по ней правой кнопкой мыши и выбрав "Переименовать переменную".

Задать переменной значение можно у любого спрайта, второму значение будет доступно автоматически. Скрипты кота:



Скрипт краба:

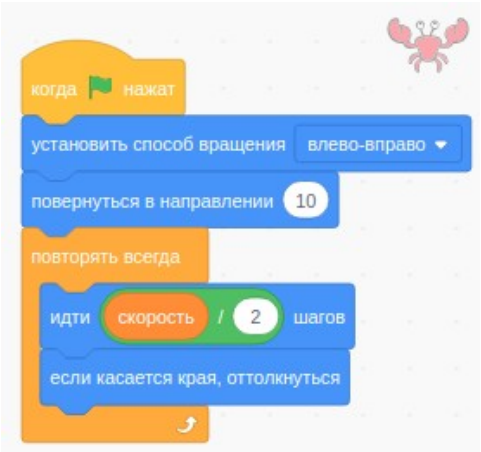


Теперь если мы захотим поменять количество шагов за один оборот цикла, достаточно сделать это в одном месте. Там, где задается значение переменной.

Когда выполняются команды "идти <скорость> шагов", то на место переменной "скорость" на самом деле подставляется ее значение. Например, число 10. В результате получается команда "идти 10 шагов", хотя мы этого не видим.

Более того, с помощью арифметических операций можно видоизменить нашу программу. Пусть краб ходит в два раза медленнее кота. То есть, если у кота скорость 10, у краба она должна быть 5. Если же у кота скорость 8, у краба она должна стать 4.

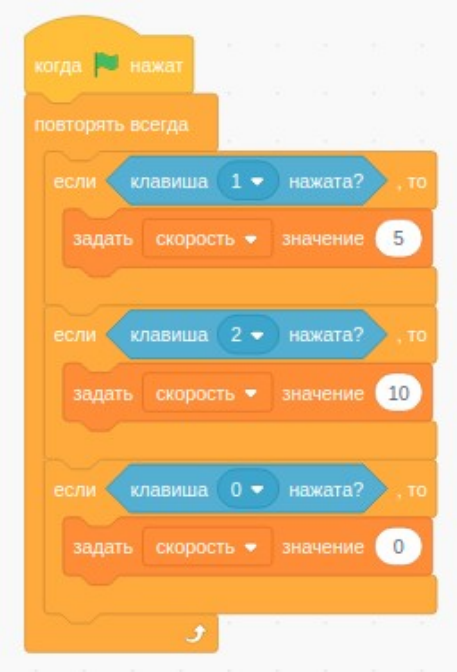
Для этого воспользуемся блоком с делением из зеленого раздела "Операции". Слева от знака деления поставим переменную "скорость", а справа – число 2.



Получается, что хотя скорость одного спрайта отличается от другого, скорости обоих взаимосвязаны через общую переменную.

Усложним нашу программу. Пусть значение переменной "скорость" задает пользователь. Если он нажмет на клавиатуре число 1, то скорость будет иметь значение 5, а если нажмет на клавиатуре число 2, скорость будет иметь значение 10. Если же человек нажмет 0, скорость должна установиться также в 0, то есть спрайты попросту остановятся.

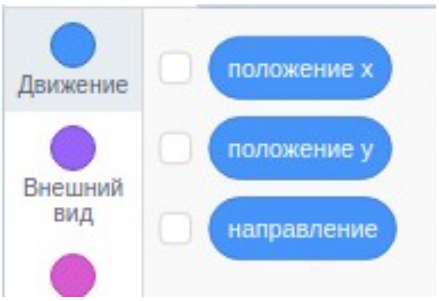
Тут нам потребуются условные операторы, а также цикл "повторять всегда", чтобы программа всегда ожидала ввода с клавиатуры.



Этот скрипт можно добавить любому спрайту.

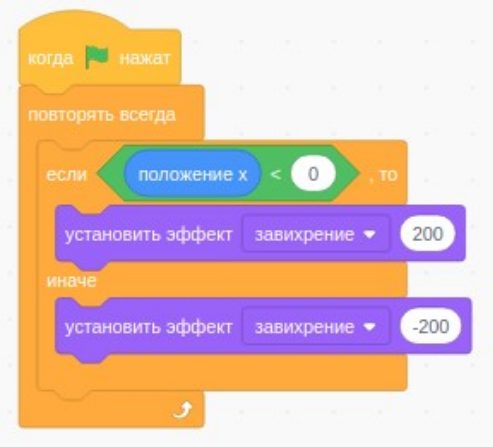
Таким образом значение переменной может меняться в процессе выполнения программы в зависимости от тех или иных условий.

Обратите внимание на форму блока, который содержит только имя переменной. Это прямоугольник со скругленными углами. Блоки подобной формы есть во многих разделах вкладки "Код". Например, в секции "Движение" это блоки "положение x", "положение y", "направление":



Подобные блоки можно назвать встроенными в Scratch переменными. В них записываются значения, которые извлекаются из каких-нибудь объектов. Так в переменную "положение x" записывается значение текущей координаты *x* спрайта. Мы можем извлекать это значение и использовать его в программе.

Допустим, наш кот, находясь на левой половине сцены, будет скручиваться в одну сторону, а когда находится на правой половине – в другую.



Здесь мы используем переменную "положение x", чтобы проверить, находится кот справа или слева. Вспомним, что 0 – это середина холста. Налево идут значения меньше нуля, направо – больше.

Задание

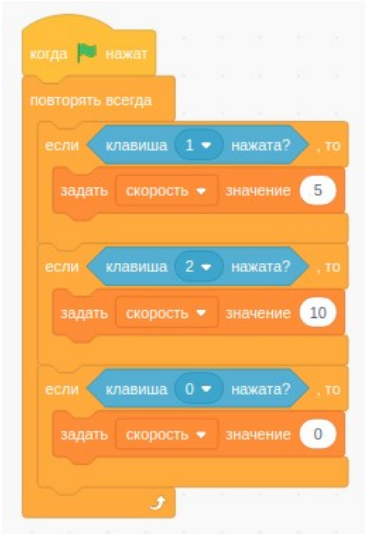
Составьте программу, в которой спрайт оказывается в том месте сцены, над которым была нажата клавиша мышки. При этом на сцене должны отображаться координаты клика (вместо клика также может быть зажатие клавиши мыши).

Ввод данных

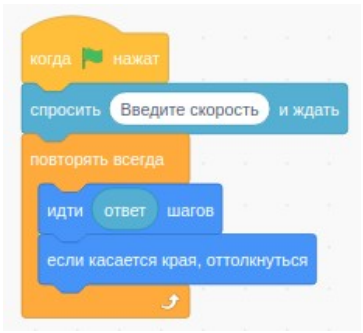
Часто в программах требуется, чтобы человек ввел какое-нибудь число или строку или выполнил любое другое действие. Это можно назвать вводом данных в программу. Мы уже с этим сталкивались в скриптах, ожидающих от пользователя нажатия клавиш клавиатуры или мыши.

Также программы могут просить человека ввести какое-нибудь число или строку. После этого программа связывает полученные данные с именем какой-нибудь переменной, чтобы сохранить данные и потом их использовать.

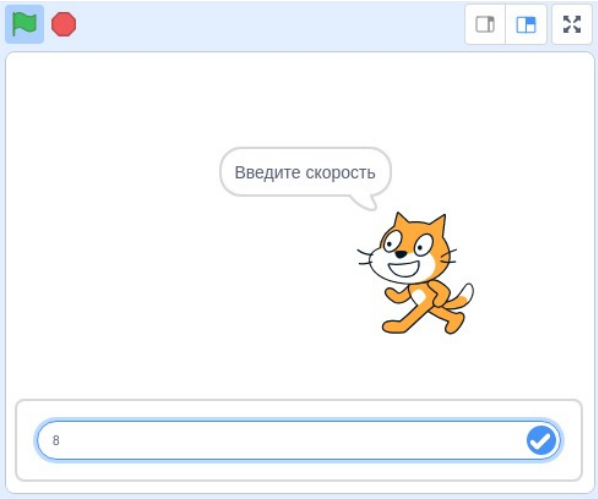
Вспомним один из скриптов прошлого занятия:



Если мы хотим предоставить человеку самому определять скорость, а не ограничивать ее только тремя вариантами, код станет проще:



Когда выполняется команда "спросить … и ждать", на сцене появляется поле, куда пользователь должен что-то ввести и нажать Enter на клавиатуре. Также он может ничего не вводить, а просто нажать Enter.



В Scratch команда "спросить … и ждать" блокирует свой скрипт, но не другие. Блокирует она свой скрипт до тех пор, пока не будет нажат Enter в поле ввода. После этого то, что было введено в поле, присваивается встроенной переменной "ответ", которая далее используется в скрипте.

Что будет, если человек введет не число, а слова? Переменная "ответ" будет содержать введенную строку текста. Но поскольку в команду "идти … шагов" бессмысленно подставлять строку, скорость кота не изменится.

Допустим, мы хотим, чтобы в случае ввода текста, скрипт не пыталась его подставлять в команду "идти … шагов". Вместо этого на сцене появлялось бы сообщение о неправильном вводе.

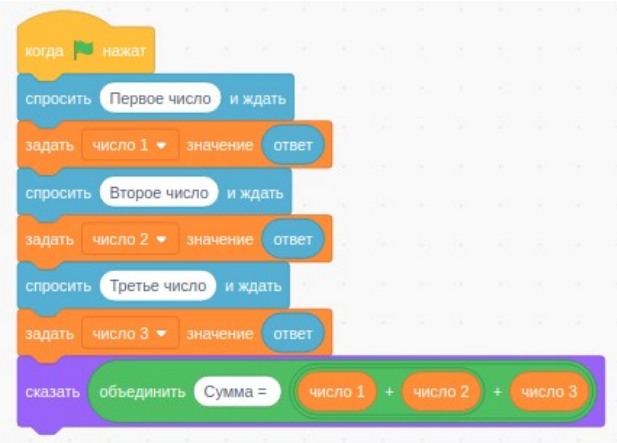
В Scratch нет команды проверки, введено число или строка. Однако можно придумать способ это узнать. В Scratch если выполнить над строкой арифметическую операцию, то результатом будет число 0. Например, если взять из положительного числа модуль, то вернется само число. Если же попытаться взять модуль из строки, что бессмысленно, будет получен 0.

Таким образом, если модуль ответа не совпадает с самим ответом, значит было введено что-то не то: либо строка, либо отрицательное число. Если же модуль ответа совпадает с самим ответом, значит было введено положительное число.

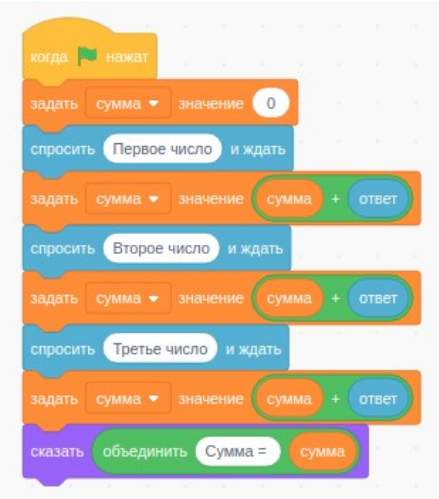
Рассмотрим второй пример использования команды "спросить … и ждать". Пусть наша программа складывает три числа, которые вводит пользователь. Здесь уже не обойтись одной

встроенной переменной "ответ". Придется создавать свои. При этом задачу можно решить разными способами.

Во первых, каждое число можно присваивать отдельной переменной, а потом выводить их сумму.



Второй вариант – создать одну переменную, в которой постепенно накапливать сумму.



Когда выполняется команда "задать <сумма> значение <сумма + ответ>", сначала выполняется подвыражение <сумма + ответ>. И только после этого результат этого подвыражения записывается в переменную "сумма".

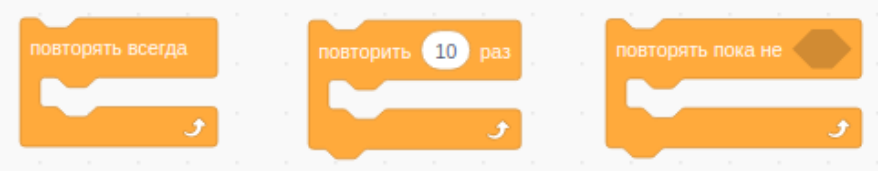
Например, переменная "сумма" содержит число 3. Пользователь вводит число 5. Когда выполняется выражение <сумма + ответ>, из переменной "сумма" извлекается число 3 и к нему добавляется 5. После этого результат 8 записывается в переменную "сумма". Старое значение 3 переменной при этом затирается, то есть теряется.

Задания

1. Составьте программу, в которой пользователь вводит координаты точки. После этого спрайт переходит в указанную точку, ждет секунду и опять запрашивает новые координаты.
2. В системе зарегистрированы три пользователя – Вася, Петя и Варя. Если в систему пытается войти один из них, кот его приветствует, говоря его имя, к которому добавляет слово "привет". Если в систему пытается проникнуть посторонний, кот включает аварийную сигнализацию, то есть мяукает.

Циклы

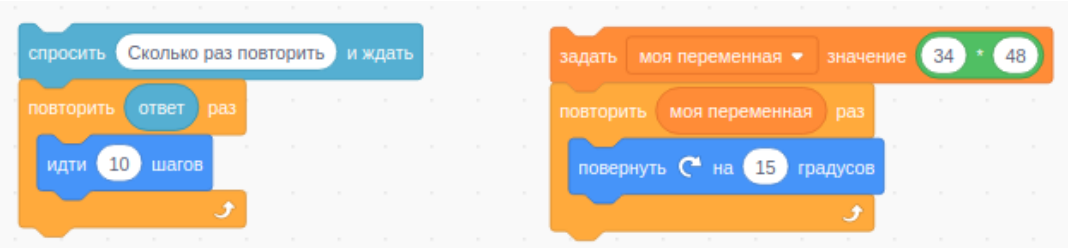
Мы уже не раз использовали циклы. Обычно это был цикл "повторять всегда". Всего в Scratch три разновидности цикла. Это блоки "повторять всегда", "повторить … раз", "повторять пока не …".



Цикл "повторять всегда" не дает скрипту самому закончить свою работу. В результате он работает бесконечно, повторяя и повторяя выполнение вложенных в тело цикла блоков. Прервать программу с таким зацикливанием можно только сигналом извне. Например, нажать на красный кружок над сценой.

С циклом "повторить … раз" мы тоже знакомы. Он повторяет вложенные в него команды количество раз, которое указано в его заголовке. Особенностью этого цикла является то, что заранее известно количество повторов.

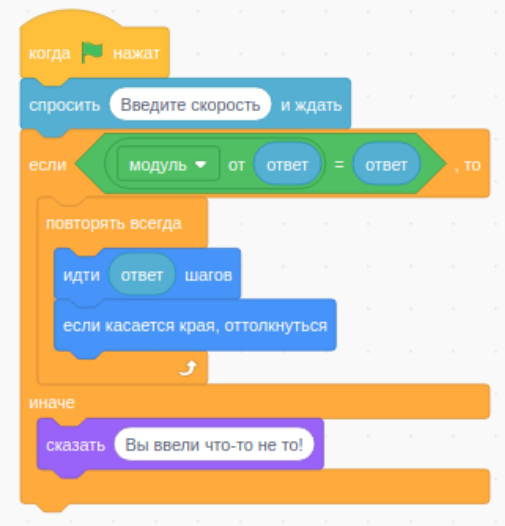
Конечно, мы можем запрашивать у пользователя количество повторов, или оно может определяться в результате арифметической операции, результат которой мы не знаем. Но все-равно, когда поток выполнения программы доходит до цикла "повторить … раз", программе уже известно количество повторов.



Новым для нас и самым сложным из всех циклов является цикл "повторять пока не … ". Тело этого цикла выполняется до тех пор, пока условие в его заголовке не вернет правду. Как только условное выражение станет истинным, цикл прекратит свою работу. Но когда условие станет правдой, мы не знаем. Поэтому неизвестно количество повторов цикла.

Тело цикла "повторять пока не …" может вообще ни разу не выполниться, если условие сразу вернет правду. Или же цикл так и не сможет завершиться, если условие всегда будет оставаться ложью.

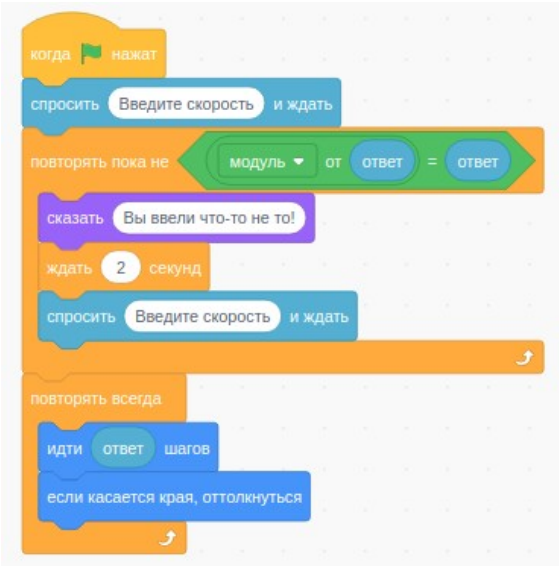
Вспомним этот пример из прошлого урока:



Здесь, если человек вводит положительное число, то программа зацикливается в цикле "повторять всегда" и спрайт ходит по сцене.

Однако, если человек вводит строку или отрицательное число, программа выполняет ветку "иначе" и завершается.

Что если мы хотим, чтобы программа продолжала запрашивать у пользователя число, пока он его не введет. То есть, когда человек вводит строку, программа должна сообщать ему о неправильном вводе и снова запрашивать число. В таком случае не обойтись без цикла "повторять пока не …".



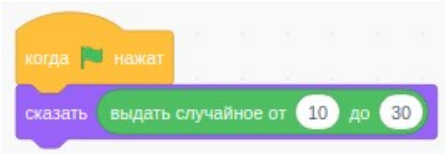
Логика здесь следующая. Пользователя просят ввести скорость. Если он вводит положительное число, то выражение "модуль от ответ = ответ" возвращает правду. Поэтому тело цикла "повторять пока не …" не будет выполнено ни разу. Поток выполнения программы сразу перейдет к циклу "повторять всегда", внутри которого реализовано ходьба спрайта.

Если человек вводит строку, выражение "модуль от ответ = ответ" возвращает ложь. Тело цикла "повторять пока не …" будет выполняться. Пользователю покажут предупреждение и

снова попросят ввести скорость. Если он опять введет строку, ему снова покажут предупреждение и снова попросят ввести скорость.

И так будет до тех пор, пока человек не введет положительное число. Когда это произойдет, мы не знаем. Но только после этого программа сможет перейти к циклу "выполнять всегда".

Рассмотрим второй пример использования цикла "повторять пока не …". Составим программу, в которой человеку предлагается угадать загаданное компьютером число. Однако перед этим познакомимся с блоком, генерирующим случайные цифры.



Команда "выдать случайное от … до …" генерирует случайное число в диапазоне от первого указанного числа до второго включительно. Например, если дана команда "выдать случайное от 10 до 30", то программа выдаст любое число, которое не меньше 10 и не больше 30. Это, например, может быть число 10, а может быть 17, или 25. Также возможно получить число 30.

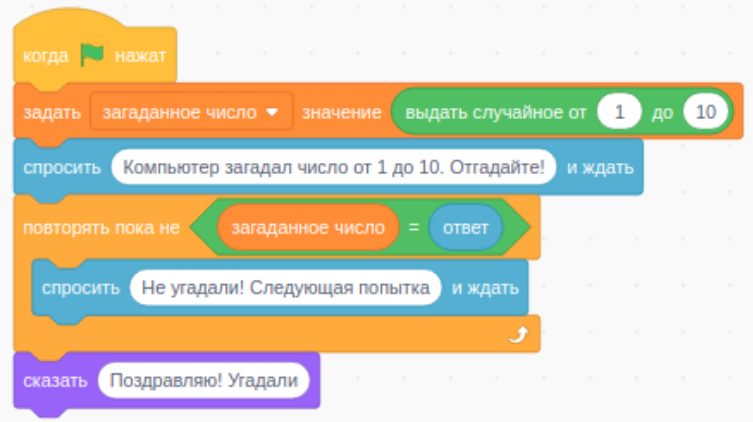
В Scratch допустимо сначала указать большее число, а затем меньшее. Например, "выдать случайное от 100 до 50". Команда и в этом случае будет правильно работать и генерировать любое число от 50 до 100.

Кроме того, можно указывать отрицательные и дробные числа. В последнем случае целая и дробная часть разделяются точкой, а не запятой.

Итак, чтобы компьютер загадал случайное число, нам нужна команда "выдать случайное от … до …". Это число надо будет где-то сохранить, чтобы потом сравнивать с тем, что вводит человек. Значит, нам понадобиться переменная.

Человек должен отгадывать число до тех пор, пока не отгадает. Значит, нам понадобиться цикл "повторять пока не …". Что значит, пока не угадает? Это значит, что пока то, что ввел человек не совпадет с тем, что загадал компьютер, цикл должен продолжать крутиться и все просить и просить ввести очередное число.

В случае совпадения чисел, цикл должен прекращать свою работу, а на сцене должно появляться поздравление.



Задание

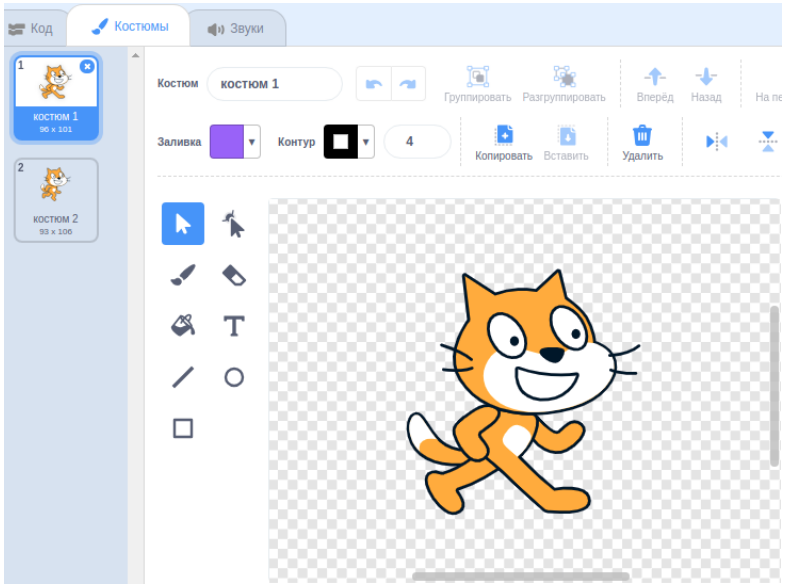
Переделайте скрипт про угадывание числа так, чтобы

1. компьютер загадывал число от 1 до 100.
2. Если человек вводит число, которое больше загаданного, то программа должна говорить ему "слишком много".
3. Если человек вводит число, которое меньше загаданного, то программа должна говорить ему "слишком мало".

Подумайте, когда программа выполняется, какой способ отгадывания числа самый оптимальный. То есть как нужно отгадывать число, чтобы угадать его за меньшее количество попыток?

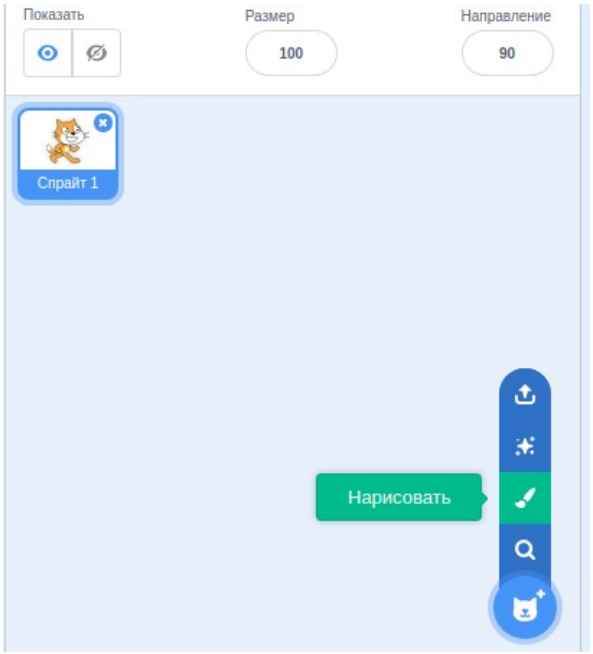
Создание спрайтов и костюмов

В Scratch на вкладке "Костюмы" есть встроенный графический редактор, в котором можно создавать собственные спрайты и их костюмы, а также изменять те, что были добавлены из библиотеки.



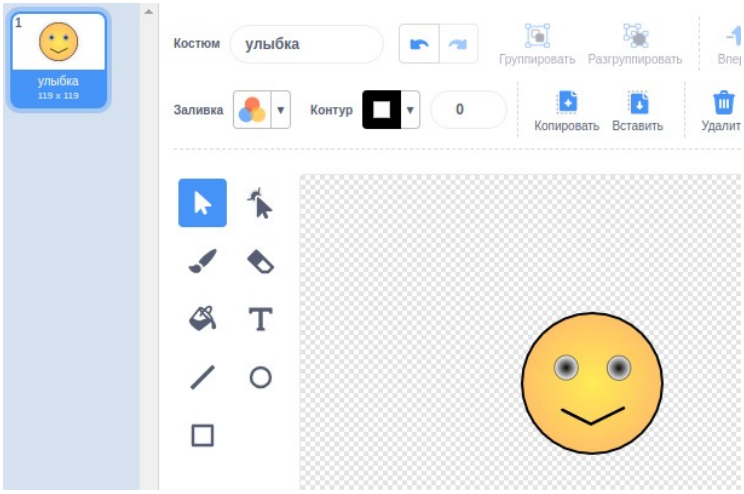
В этом уроке создадим собственный спрайт, а также несколько его костюмов.

Первым делом надо создать нового героя. Для этого на панели спрайтов, которая находится под сценой, в меню выбираем пункт "Нарисовать".



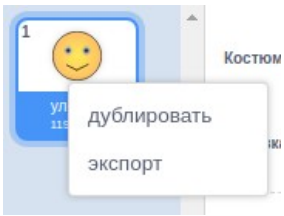
После этого здесь появится пустой спрайт, а на холсте вкладки "Костюмы" мы увидим, что ничего не нарисовано.

Пусть нашим героем будет смайлик. Нарисуем его с помощью инструментов "Круг" и "Линия". Также воспользуемся заливкой, чтобы раскрасить.

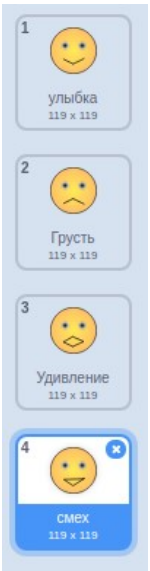


Костюмы можно переименовывать, как и спрайты.

Чтобы не рисовать второй костюм, продублируем первый. Для этого надо кликнуть по нему правой кнопкой мыши и выбрать в контекстном меню "дублировать".

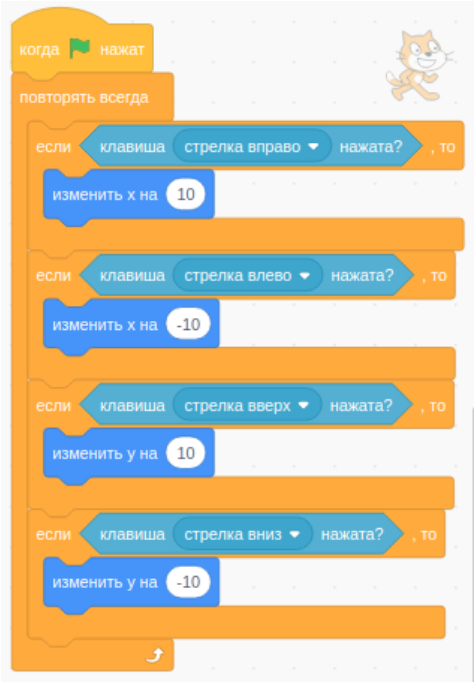


Таким образом, создадим смайлу несколько костюмов, изменяя только его улыбку.

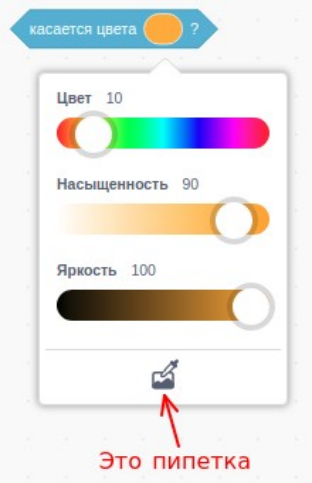


Теперь придумаем сценарий, в котором смайлу придется менять свои костюмы. Пусть кот ходит по сцене с помощью стрелок клавиатуры, когда касается смайла, тот случайным образом меняет свой костюм.

Для начала запрограммируем перемещение кота:



Теперь надо как-то узнавать, что кот касается смайла? В разделе "Сенсоры" есть команда "касается цвета …". Если кликнуть по полю с цветом, то появится панелька, где можно выбрать цвет. Чтобы точно определить тот или иной цвет, лучше воспользоваться пипеткой. Ей надо кликнуть по объекту на сцене, касание с которым будет обрабатываться программой.

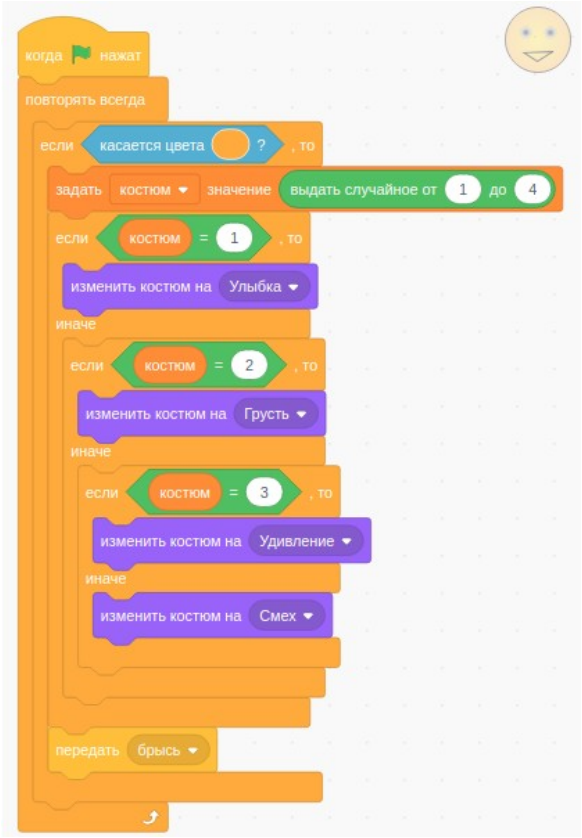


Однако кто кого касается? Кот смайла или смайл кота? В данном случае разница есть, так как если кот касается смайла, то блок "касается цвета …" должен добавляться коту. Если же смайл касается кота, то блок добавляется смайлу.

Результат взаимодействия спрайтов больше всего отражается на смайле. Ведь это именно он меняет свои костюмы. Значит, команду "касается цвета …" лучше всего добавить ему.

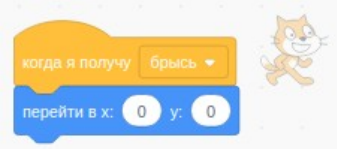
Бросим эту команду в редактор кода смайла, выберем пипетку и наведем ей на кота. Причем так, чтобы выбрать его оранжевый цвет. Ни в коем случае не белый или черный. А только какой-то уникальный цвет кота, характерный только для него.

Поскольку костюм смайла должен меняться случайно, понадобиться команда "выдать случайное от … до …". Итоговый скрипт смайла может выглядеть так:



Команда "передать брысь" внутри блока "если касается цвета" нужна для того, чтобы как только кот коснется смайла, он сразу отскакивал от него. Иначе цикл, который крутится очень быстро, снова зафиксирует касание. Получится, что смайлик от одного касания кота начнет очень быстро менять свои костюмы.

Чтобы этого не произошло, можно либо вставить команду "ждать … секунд", либо быстро убирать кота от смайла. В данном случае используется второй вариант. Посылается сигнал, а когда кот его принимает, то убегает туда, где нет смайла.



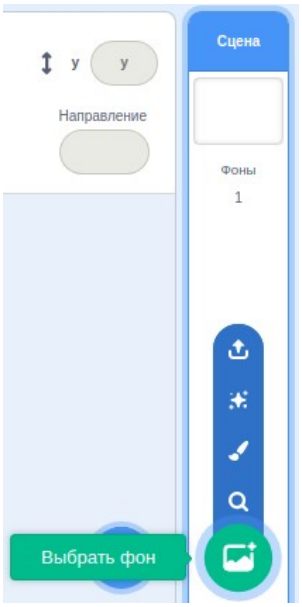
Задание

Реализуйте сценарий, в котором при каждом нажатии пробела два спрайта попеременно превращаются друг в друга. Например, при каждом нажатии пробела яблоко превращается в бананы, а бананы – в яблоко.

Сцена

В Scratch сцена – это такой же полноценный объект как спрайт, но с некоторыми особенностями. Если у спрайта может быть несколько разных костюмов, то у сцены может быть несколько разных фонов. Сцена может иметь собственные скрипты. Однако не все команды спрайтов доступны для сцены. Например, у сцены нет всех блоков раздела "Движение".

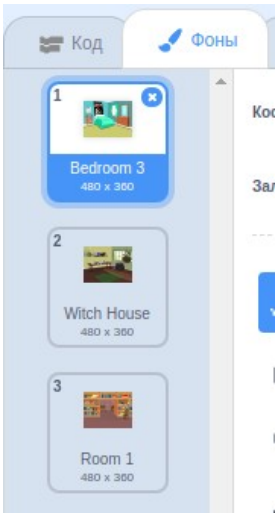
В Scratch, когда создается новый проект, у сцены имеется только один фон – белый лист. Для добавления новых фонов предназначена панель сцены, которая находится в нижней правой части интерфейса среды программирования:



Обратите внимание, когда вы кликаете по этой панели, сцена выделяется, а на вкладке "Код" будут команды для сцены. Также вкладка "Костюмы" теперь будет называться "Фоны". Здесь можно нарисовать фон, добавить новый, поменять фоны местами и т. д.

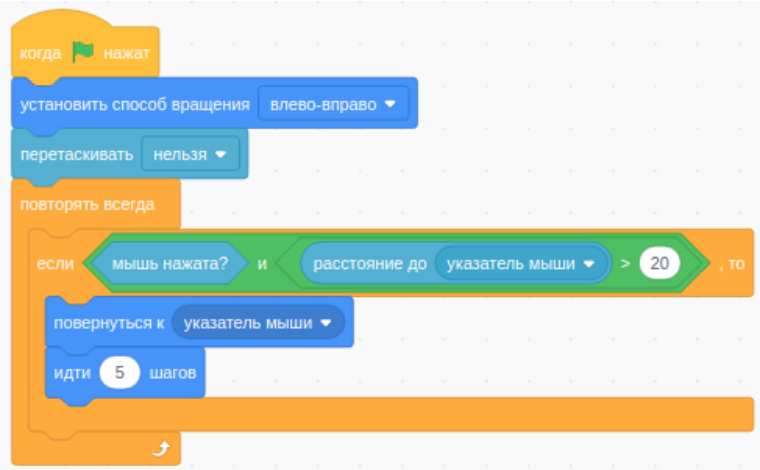
Если у сцены есть разные фоны, очевидно, что в процессе игры или анимации они будут меняться. Пусть по сценарию кот перемещается по трем разным комнатам. Одна из них играет роль центральной. Когда кот доходит до ее левого края, то переходит во вторую комнату. А когда кот доходит до правого края центральной, переходит в третью комнату.

Сначала надо добавить три фона для сцены:

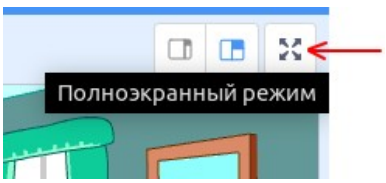


Исходный белый фон удалим.

Теперь составим скрипт перемещения спрайта в направлении к зажатому курсору мыши:



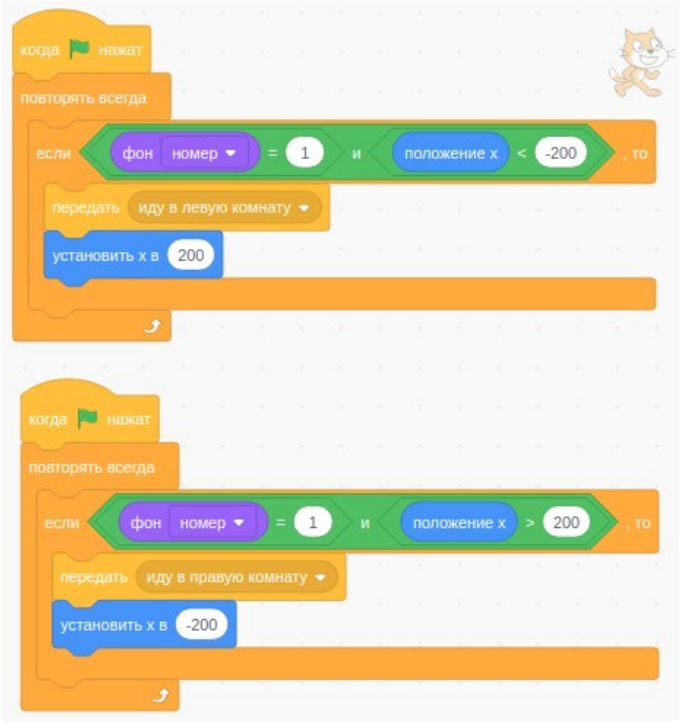
Данный скрипт может не совсем правильно работать, когда вы находитесь в режиме кодирования. Несмотря на команду "перетаскивать нельзя", спрайт все равно можно будет перетаскивать мышью. Поэтому для тестирования игры лучше переключаться в полноэкранный режим.



Вспомним, что размер холста в Scratch равен 480x360 пикселей, а начальная точка системы координат находится в центре. Поэтому у правого края значение x = 240, у левого x = -240. Пусть фон меняется, когда координата x спрайта равна 200 или -200.

Когда координата кота станет больше 200, он будет посылать одно сообщение, а когда меньше -200 – другое. Сцена, в зависимости от того, какое сообщение она получит, будет делать активным тот или иной фон.

Скрипты спрайта:

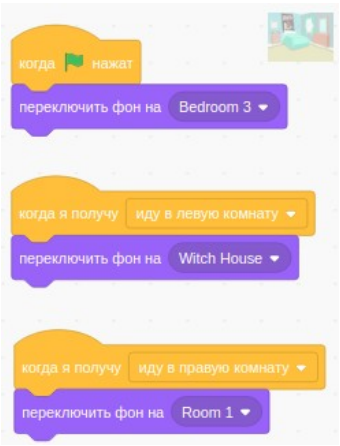


Обратите внимание, мы отслеживаем не только координату *x*, но и то, какой фон активен на данный момент. Ведь если кот переходит в другие комнаты, там эти блоки "если" не должны работать. В других комнатах все по-другому.

Во вторых, внутри "если" команда изменения координаты *x* используется для того, чтобы кот на другом фоне появлялся с верной стороны. Ведь если он входит в левую комнату, то на ней должен появляться с правой стороны.

Номер фона определяется его положением на вкладке "Фоны". Вместо номеров можно использовать имена фонов.

Скрипты сцены:

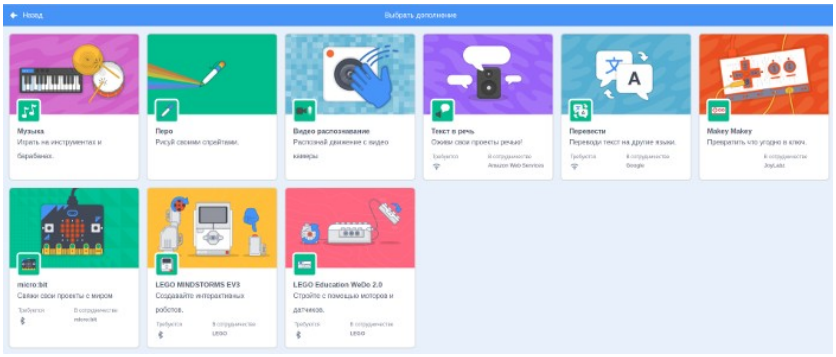


Задание

Доделайте приведенный в уроке сценарий так, чтобы спрайт мог возвращаться из левой и правой комнат в центральную.

Расширение "Перо"

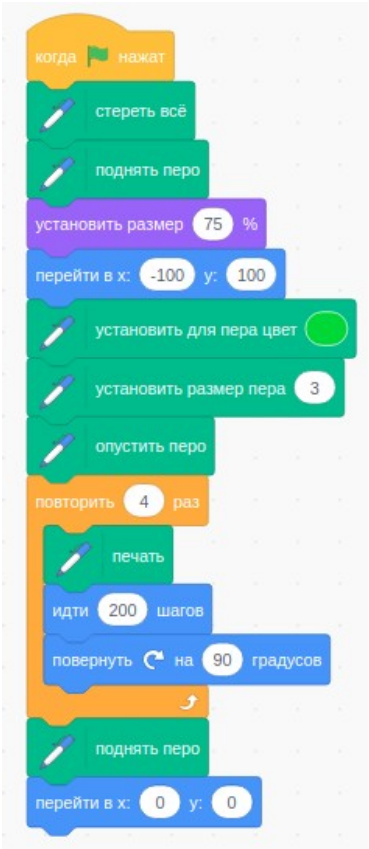
В Scratch кроме стандартных категорий блоков таких как "Движение", "Внешний вид" и так далее, которые сразу доступны во всех проектах, имеются другие. Их называют расширениями или дополнениями. Чтобы увидеть все доступные расширения, надо нажать на кнопку "Добавить расширение", которая находится на вкладке "Код" в самом низу. После этого откроется окно, где можно выбрать дополнение.



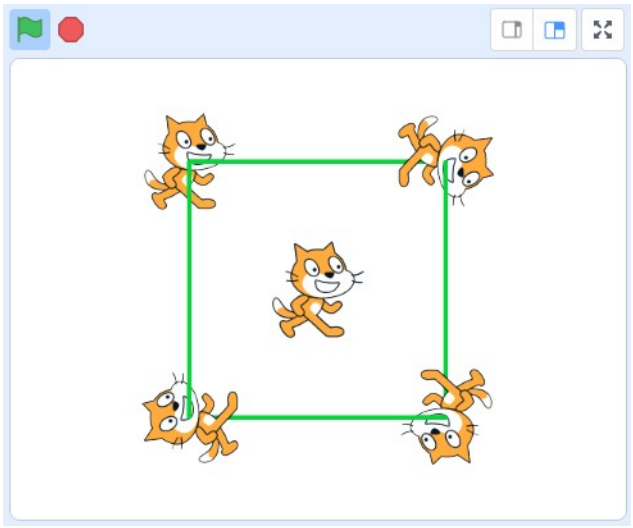
Большинство дополнений предполагают использование внешних устройств, таких как микрофон, камера, робот и др. Однако это не касается первых двух – "Музыки" и "Пера".

После выбора расширения "Перо" во вкладке "Код" среды программирования Scratch появляется новый раздел с блоками-командами, которые позволяют рисовать на сцене. Это значит, что спрайты при перемещении смогут оставлять после себя видимую линию.

Вот пример сценария для кота:



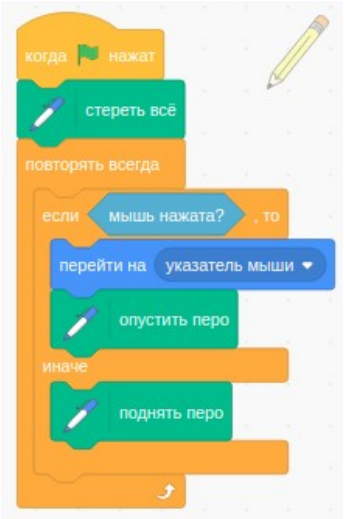
Проиграв его, получим такую картину:



Когда у спрайта поднято перо, то он не оставляет линии при перемещении. Когда перо опущено, как бы вы не перемещали спрайт из точки А в точку Б, от А до Б будет рисоваться линия установленного цвета и толщины.

Команда "печать" отпечатывает изображение спрайта в том месте, где он находится.

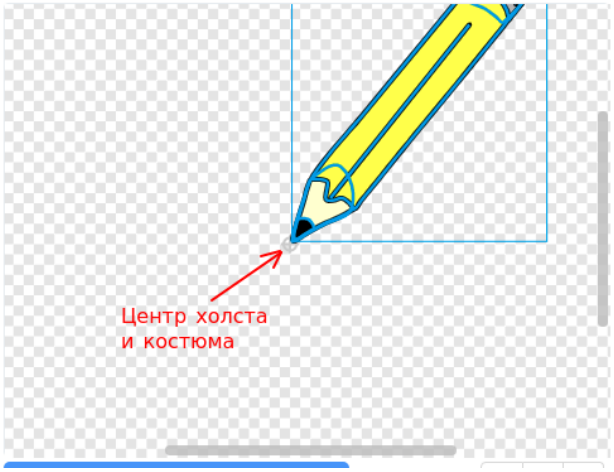
В Scratch можно создать сценарий, когда при его запуске пользователь сам будет рисовать, перемещая героя. Добавим на сцену спрайт-карандаш и запрограммируем его следующим образом:



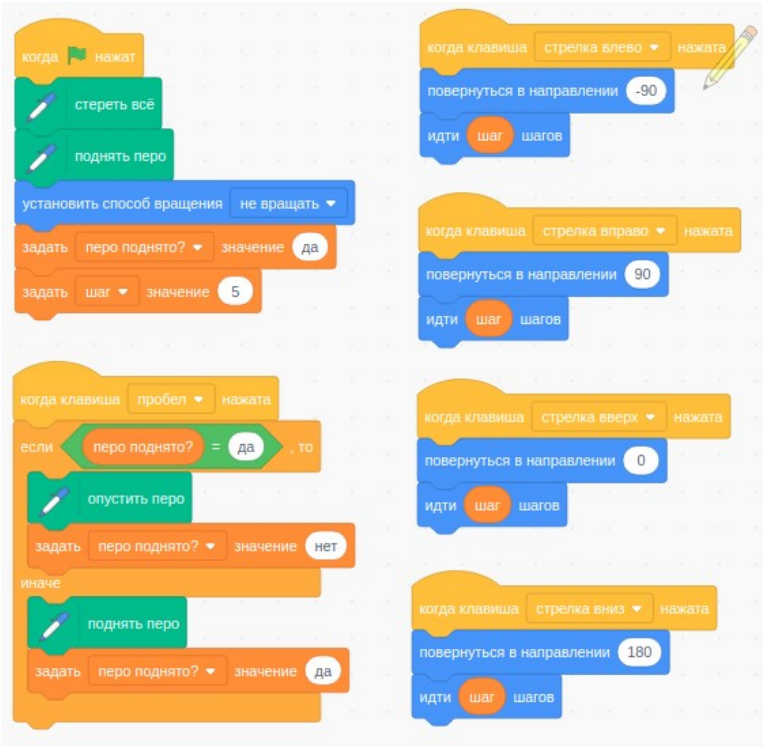
Игру следует запускать в полноэкранном режиме. В режиме программирования скрипт не сможет правильно работать.

После запуска карандаш можно перемещать мышью, и он будет оставлять след. Можно намалевать на сцене что-угодно. Однако линия будет выходить не из кончика карандаша, а его середины, что неправильно. Исправляется это редактированием костюма.

В центре холста есть чуть видимая точка, которая обозначает центр спрайта. Именно из этого центра рисуется линия. Надо переместить к центру холста кончик карандаша.

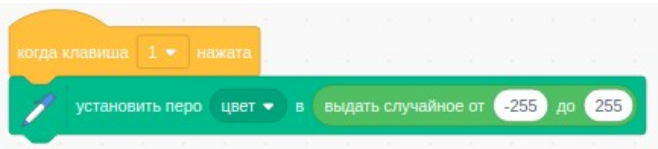


Рисовать мышью не совсем удобно. Поменяем управление на клавиатуру:



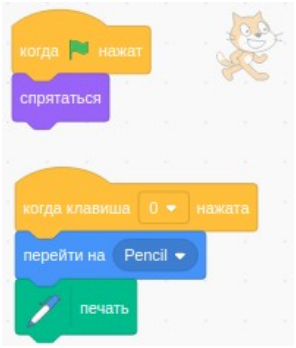
Теперь карандаш управляется стрелками клавиатуры. Когда мы нажимаем пробел, то попеременно меняем состояние пера. Поднимая перо, мы можем перемещать карандаш по сцене без рисования линии.

Сценарий спрайта можно усложнить, добавив к нему скрипты смены толщины пера и цвета. Пусть цвет, например, задается случайным образом:



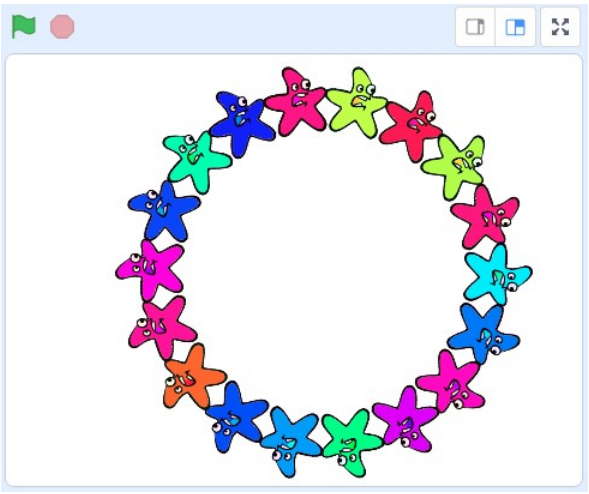
А что если мы хотим, чтобы карандаш печатал не самого себя, а какой-то другой спрайт? Этот другой спрайт должен появляться из кончика карандаша при нажатии, скажем, клавиши 0.

Тогда этому спрайту надо как-то передать координаты карандаша. Однако проще воспользоваться командой "перейти на …", в которой вместо "случайное положение" следует выбрать имя спрайта-карандаша.



Задание

Составьте программу, которая при запуске выводит на сцене окружность, состоящую из печатей спрайта.



Списки

В Scratch в категории блоков кода "Переменные" есть кнопка "Создать список". В отличие от обычной переменной, список может хранить не одно значение, а много.

Например, мы создаем обычную переменную "число" и связываем ее с хранящимся в памяти компьютера числом 5. Но если мы создаем список "числа", то можем связать его с несколькими числами, расположенными в памяти друг за другом. Пусть это будут числа 3, 8, 1, 5.

У элементов принадлежащих одному списку есть порядковые номера. Так в нашем примере число 3 имеет номер 1, число 8 – номер 2, число 1 – номер 3, число 5 – номер 4. Таким образом, используя номера элементов, мы можем извлекать значения элементов. Например, команда "элемент 2 в числа" вернет нам 8.

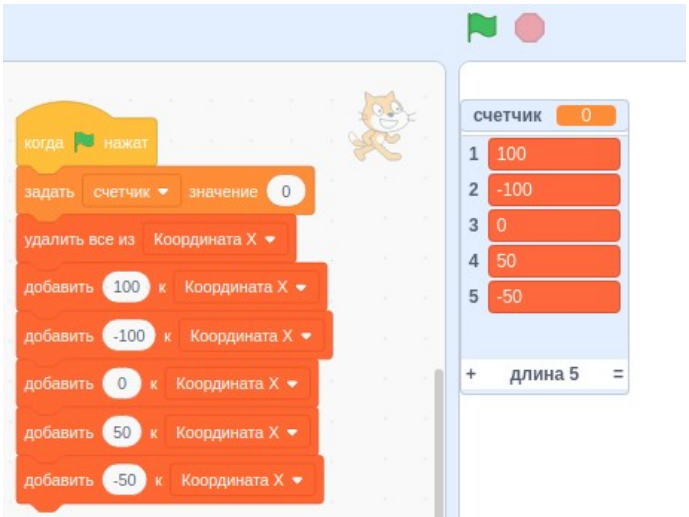
В программировании часто используется понятие массива. Списки и массивы – близкие понятия. Однако обычно их различают следующим образом. В массиве могут содержаться только данные, принадлежащие одинаковому типу. Например, только слова или только числа. Элементами одного и того же массива не могут быть и числа и слова.

А вот в случае со списком такое может быть. Например, список [3, А, 4, 5.2, Кот] содержит целые числа, дробное число, букву и слово.

Зачем вообще нужны списки и массивы? В первую очередь для удобства обработки данных. Представьте, что вам надо изменить нескольких чисел. Если каждое число будет храниться в своей переменной, то придется помнить имена всех переменных и обращаться к каждой индивидуально.

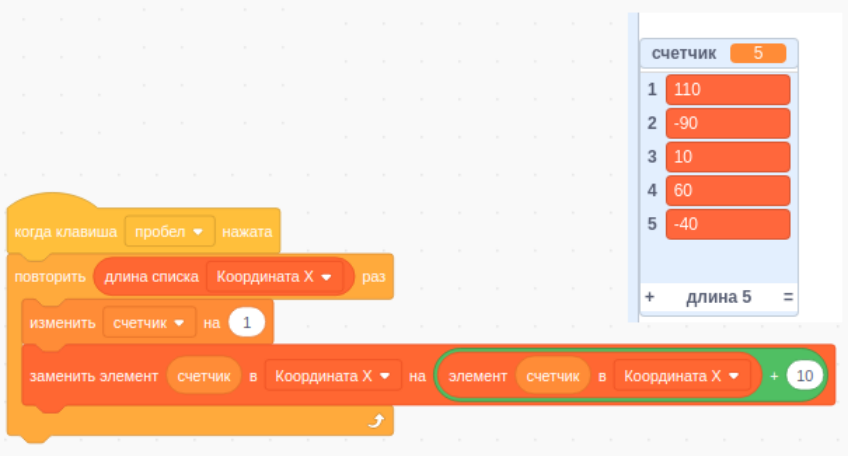
Если же ряд чисел хранится в одном списке, у нас имеется только одна переменная – имя списка. Извлекать же отдельные числа мы можем по их порядковым номерам, то есть индексам.

Рассмотрим пример. Заполним список "Координата X" числами и отобразим список на сцене, установив флажок у переменной.



Также у нас здесь есть обычная переменная "счетчик".

Теперь представим, что при нажатии пробела все координаты должны увеличиться на 10. Это значит, надо перебрать все элементы списка и изменить их. Такое обычно выполняют в цикле.



Справа на изображении показан список после нажатия пробела.

Сколько оборотов делает цикл "повторить … раз"? Столько, какова длина списка. А чему равна длина списка? В списке 5 элементов, значит, длина равна пяти.

Что происходит с переменной "счетчик" на каждом обороте цикла? Она увеличивается на единицу. Если исходное значение счетчика 0, то на первом обороте цикла он получит значение 1, на втором 2, на третьем 3, на четвертом 4, на пятом 5.

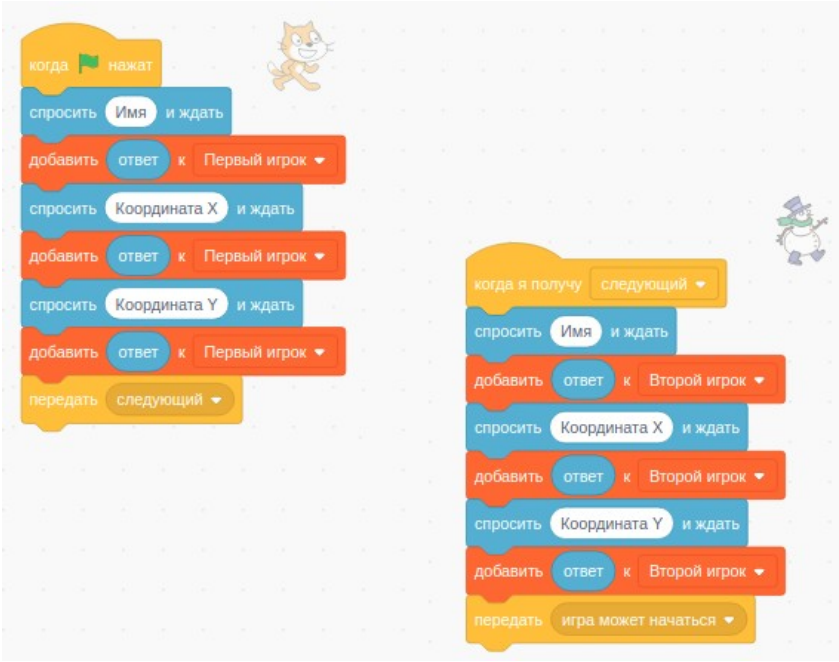
Что происходит внутри зеленого блока, который суммирует два числа? Сначала из списка "Координата X" извлекается значение элемента, номер которого равен счетчику. Например, при первом обороте цикла счетчик равен 1, значит из списка будет извлечено число 100.

К этому числу добавляется 10. Получиться 110. После этого будет выполняться команда "заменить элемент … в … на …", которая заменит число 100 в позиции, на которую указывает счетчик, на число 110.

Рассмотрим следующий пример. Есть два спрайта-игрока, каждый из которых должен получить имя и начальные координаты. Их задает пользователь. В данном случае имеет смысл хранить данные о каждом игроке в отдельных списках – "Первый игрок" и "Второй игрок".

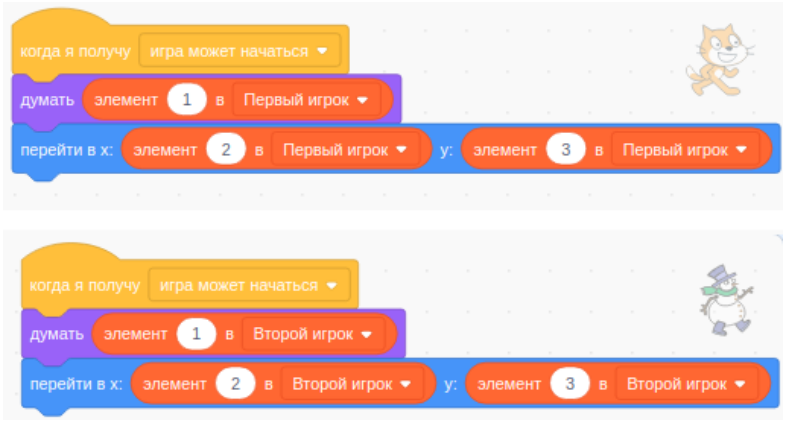
Договоримся, что первым элементом каждого списка будет имя игрока, вторым – координата X, а третьим – координата Y.

Заполняющие списки скрипты спрайтов.



Обратите внимание, что добавление элементов происходит в конец списка. Поэтому имя будет иметь порядковый номер 1, а координата Y – номер 3.

В процессе игры у спрайтов могут выполняться сценарии, извлекающие данные из списков.



Может заподозрить, что мы сделали много бессмысленных вещей. Зачем добавлять данные в список, если можно сразу использовать "ответ" для команд "думать …" или перемещения? Однако если данные будут требоваться неоднократно, то надо сохранять их. Причем мы обошлись только двумя переменными типа список – "Первый игрок" и "Второй игрок, а не шестью, как если бы пользовались обычными переменными.

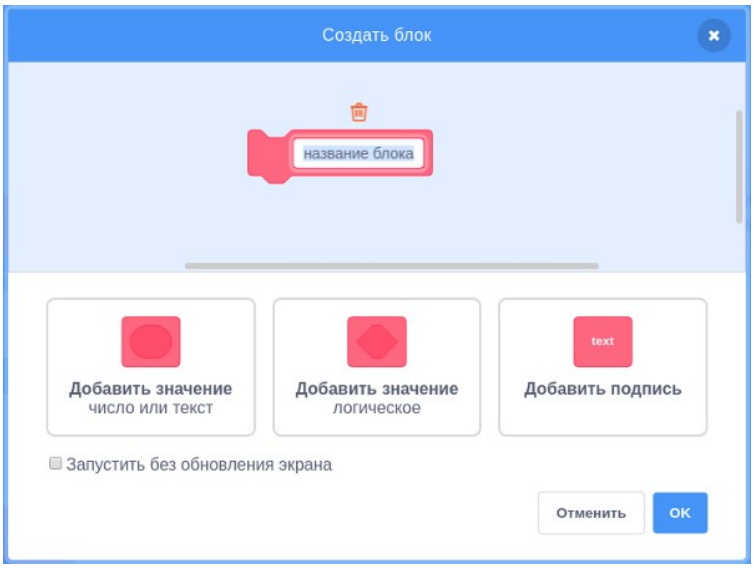
Задание

1. Заполните список десятью случайными числами.
2. Отобразите числа на сцене, установив флажок у переменной списка.
3. Спросите у человека номер элемента, значение которого он хотел бы заменить.
4. Спросите у человека новое значение.
5. Замените в списке старое значение новым в указанной позиции.

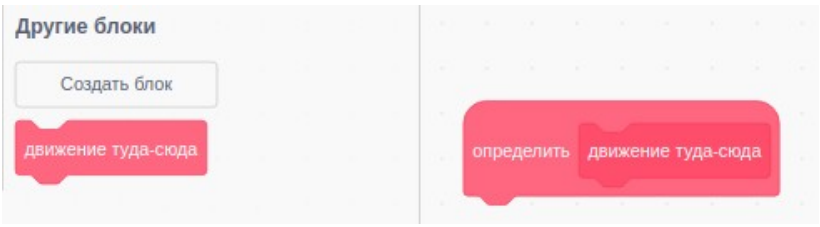
Функции

В программировании часто используются функции. Функция представляет собой отдельно выделенный участок программного кода, который имеет имя подобно имени переменной. Из основного скрипта мы можем вызывать функцию, обращаясь к ней по имени. Когда функция вызывается, составляющие ее команды выполняются.

В Scratch есть возможность создавать подобие функций. Делается это в разделе кода "Другие блоки". После нажатия кнопки "Создать блок", появляется диалоговое окно:



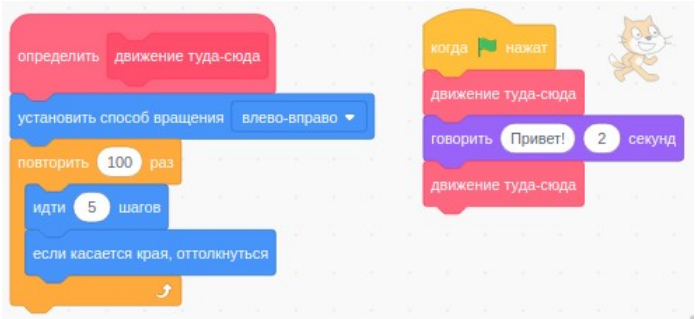
В самом простом случае достаточно вместо текста "название блока" вписать желаемое имя функции, например "движение туда-сюда", и нажать "OK". После этого в Scratch появятся два блока.



Один из блоков сразу оказывается в рабочей области среды программирования и содержит слово "определить". Данный блок является заголовком определяемой, то есть создаваемой, функции. После заголовка должны идти команды, которые будут выполняться, когда функция будет вызываться.

Чтобы вызвать функцию, надо использовать блок, который находится в разделе "Другие блоки".

Пример определения функции и ее вызов:



В скрипте, который начинается с команды "когда флажок нажат", два раза вызывается функция "движение туда-сюда". При запуске игры, когда поток выполнения программы доходит до вызова функции, происходит переход к определению функции и начинают выполняться ее команды.

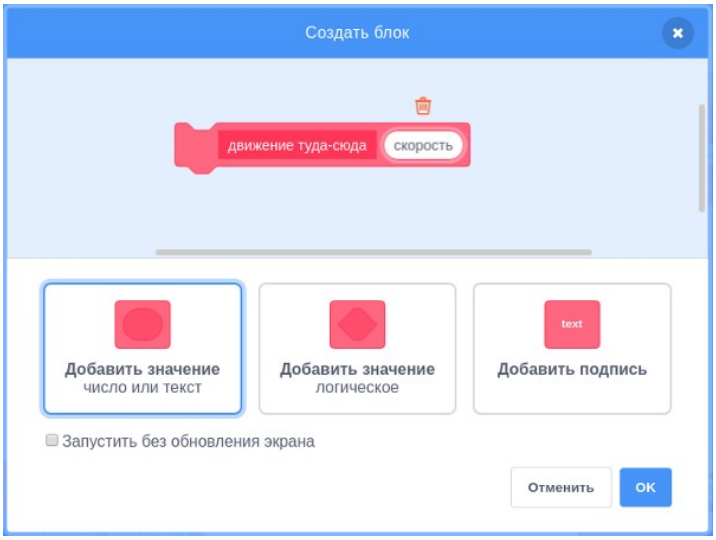
Когда команды функции заканчиваются, поток выполнения программы возвращается туда, откуда функция вызывалась.

В данном случае после первого вызова функции будет выполняться команда "говорить Привет 2 секунд". Потом произойдет второй вызов функции.

Глядя на этот скрипт, понятна выгода от функций. Мы избегаем повторения кода в основном скрипте кота. Однако есть ли разница по сравнению с циклами? Ведь то, что находится в теле функции, можно было бы разместить в цикле и повторить его 2 раза.

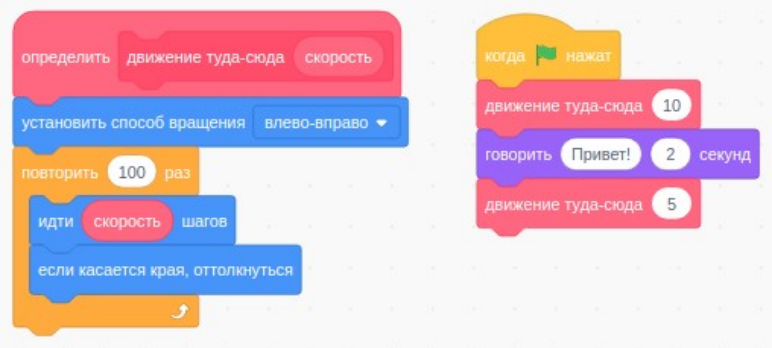
Разница в том, что функцию можно вызывать из разных мест программы. А проходы по телу цикла выполняются сразу друг за другом.

Создадим более сложный блок – функцию с параметром. Параметры также называют аргументами.



В диалоговом окне "Создать блок", после ввода имени блока, нажмем на кнопку "**Добавить значение** число или текст". В блоке появится еще одно поле. Впишем туда имя параметра – в данном случае "скорость".

Составим такие скрипты:

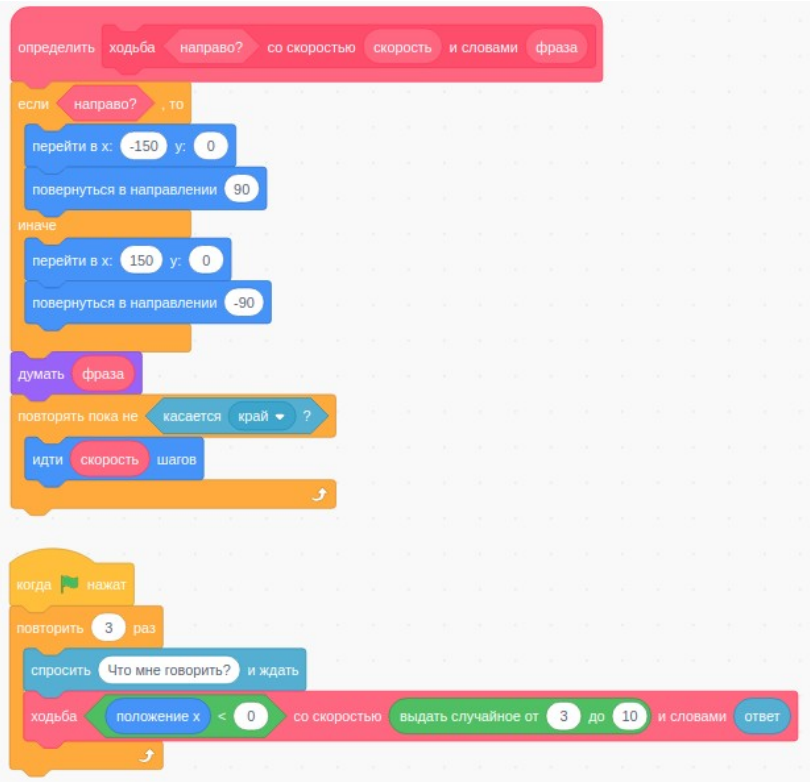


В отличие от первого примера здесь при вызове функции мы должны передать в нее одно значение. В заголовке функции это значение привязывается к переменной "скорость". В теле функции мы можем так или иначе использовать переменную. При выполнении кода вместо нее будет подставляться переданное в функцию значение.

Так при первом вызове функции переменная "скорость" равна 10, а при втором – равна 5. Чтобы вставить "скорость" внутрь команды "иди … шагов", надо сначала вытянуть переменную из заголовка функции. При этом автоматически создается копия блока "скорость".

Данный пример иллюстрирует, что функции помогают не только избегать дублирования кода. Их поведение можно видоизменять от вызова к вызову, передавая различные аргументы.

Рассмотрим более сложный пример с параметрами:



В данном случае функция принимает три аргумента – "направо?", "скорость" и "фраза". Причем первый добавлен как логическое значение. Вспомним, что в Scratch логическое значение – правду или ложь – возвращают блоки с острыми углами по сторонам.

Для остальных двух параметров при создании блока также были добавлены подписи. Это делать не обязательно. В Scratch они используются для пояснения.

При вызове функции переменной "направо?" присваивается результат логического выражения "положение x < 0". Это выражение возвращает правду, если спрайт находится в левой половине сцены. В этом случае в функции сработает ветка "если". В итоге спрайт будет идти слева направо.

Задание

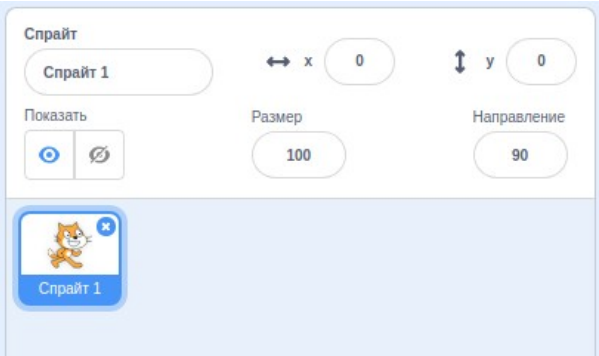
Создайте функцию, которая рисует на сцене правильный многоугольник. Функция должна принимать два аргумента – количество сторон многоугольника и длину одной стороны.

Ответы к заданиям

Что такое Scratch

Задание 1

В уроке мы не обсудили часть интерфейса среды Scratch, которая находится под холстом и в которой задаются свойства объекта.



Изучите представленные здесь поля самостоятельно. Что определяют их значения? Подумайте, почему важно задавать объектам, которые в Scratch по умолчанию называются Спрайтами, осмысленные имена.

Ответ

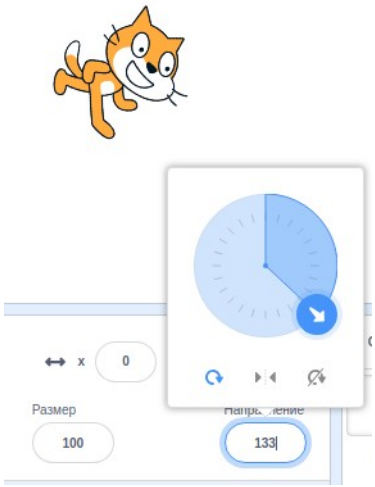
В поле *Спрайт* задается имя объекта. Важно давать объектам осмысленные имена, так как объектов-спрайтов в программе может быть много. Если все они будут называться "Спрайт 1", "Спрайт 2" и так далее, то легко запутаться.

Когда на сцене много объектов, иногда бывает удобно временно убрать видимость некоторых, чтобы они не мешали программировать остальные спрайты и следить за ними. Чтобы скрыть объект, надо переключиться на кнопку с зачеркнутым глазом.

Поля *x* (икс) и *y* (игрек) определяют местоположение спрайта на холсте. Координата (0; 0), то есть когда $x = 0$ и $y = 0$, соответствует середине холста.

Поле *Размер* определяет размер спрайта, который выражается в процентах. 100% соответствуют исходному размеру изображения спрайта. Значение меньше 100 будет уменьшать спрайт, а больше 100 – увеличивать.

Поле *Направление* сложное. После клика по нему появляется круг:



Обратите внимание на три маленькие кнопки под кругом. Когда в уроке мы использовали команду "установить способ вращения влево-вправо", то могли этого не делать. Достаточно было здесь переключиться на вторую кнопку с изображением двух маленьких треугольников. Она задает способ вращения слева направо.

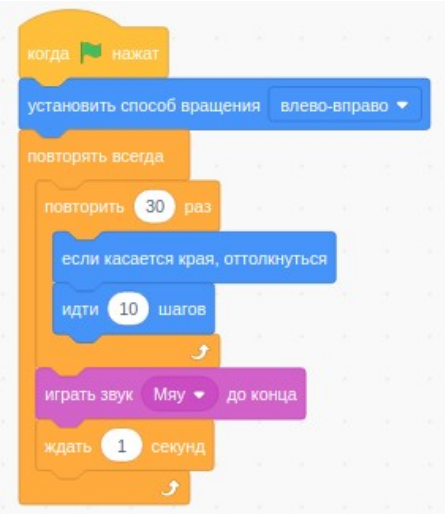
Первая кнопка включена по умолчанию и определяет способ вращения кругом. Если эта кнопка включена, и вы начнете перетаскивать стрелку на большом круге, то увидите как спрайт вертится вокруг своей оси.

Третья кнопка отменяет любое вращение.

Задание 2

Составьте программу, согласно которой кот ходит туда-сюда, но через каждые 300 шагов останавливается на 1 секунду и говорит "Мяу". Подсказка: среди прочего вам понадобятся две команды, которые не рассматривались в этом уроке. Одна из них находится в разделе "Управление", а вторая – в разделе "Звук".

Ответ



В программе надо использовать два цикла: "повторять всегда" и "повторить … раз". Второй цикл вкладывается внутрь первого. Внутри вложенного цикла содержатся команды "идти ... шагов" и "если касается края, оттолкнуться". Произведение количества повторов и

количества шагов должно давать число 300. Например, 30 повторов по 10 шагов в общей сложности дадут 300 шагов.

После каждых трехсот шагов кот должен мяукать и останавливаться. Для этого используются команды "играть звук … до конца" и "ждать ... секунд". Эти команды должны быть за пределами внутреннего цикла, но должны оставаться внутри внешнего цикла "повторять всегда".

То есть кот делает 300 шагов, мяукает, ждет. Потом снова делает 300 шагов, мяукает, ждет. И так далее до бесконечности.

Ставить команду "играть звук … до конца" впереди команды "ждать … секунд" или сзади – дело вкуса. Однако результат будет несколько различаться. Если поместить "ждать … секунд" выше, то кот сначала остановится и только через 1 секунду замяукает.

Вместо команды "играть звук … до конца" можно использовать "включить звук …". Отличие заключается в том, что вторая команда играет звук, когда кот выполняет уже следующую команду. То есть когда звук начинает проигрываться, поток выполнения программы не ждет его завершения. Звук играет как бы параллельно.

Спрайты

Задание

Составьте программу, в которой один спрайт постоянно ходит за указателем мыши, а второй – находится в центре сцены и, стоя на одном месте, медленно крутится по часовой стрелке. Слой первого объекта должен быть впереди второго.

Для выполнения задания вам потребуются две дополнительные команды – "повернуться к указатель мыши", "повернуть по часовой стрелке на … градусов" (вместо слов "по часовой стрелке" на блоке изображена дуговая стрелка).

Ответ

Последовательное и одновременное выполнение скриптов

Задание

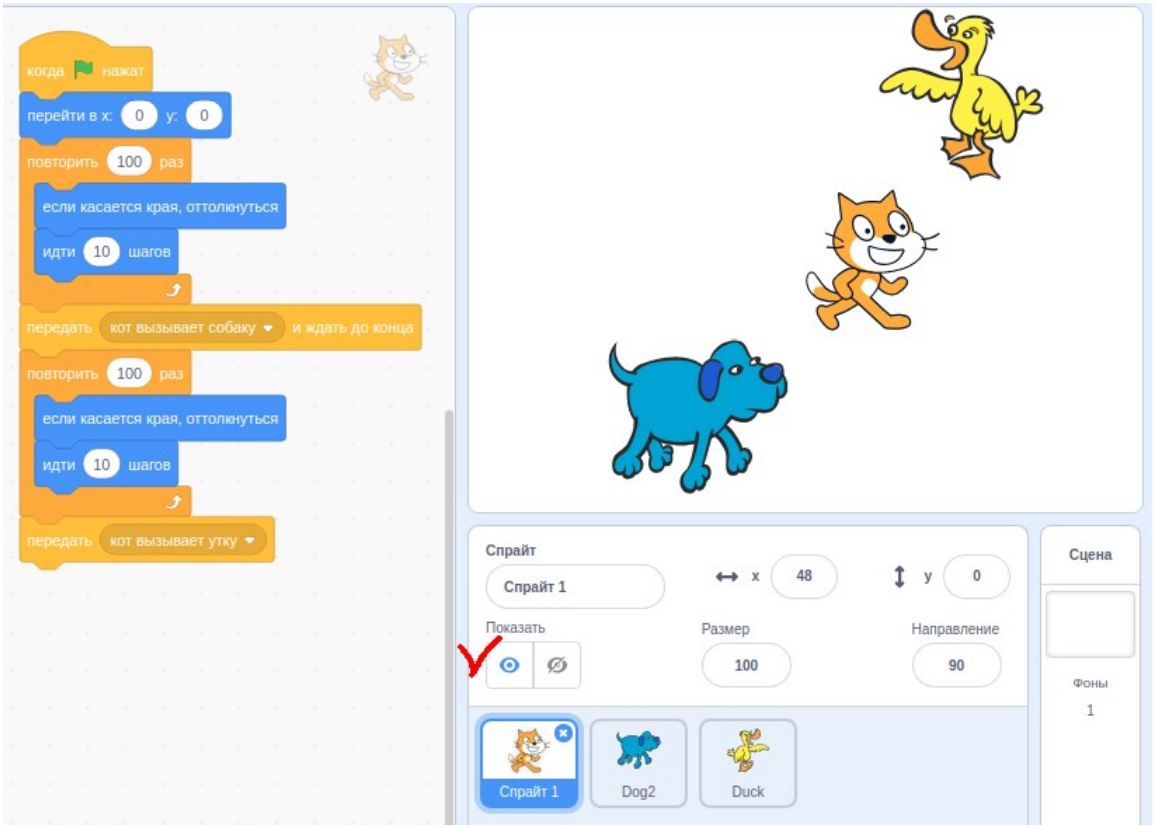
Составьте программу по следующему описанию:

1. Кот недолго ходит по сцене, потом останавливается, но не исчезает.
2. Появляется второй спрайт и тоже недолго ходит по сцене.
3. Когда второй спрайт останавливается, кот снова начинает недолго ходить по сцене.
4. Когда кот останавливается во второй раз, на сцене появляется третий недолго бегающий спрайт.

При выполнении задания также воспользуйтесь блоком "передать сообщение и ждать до конца". Эта команда останавливает текущий скрипт до тех пор, пока не выполнится скрипт, который получает сообщение.

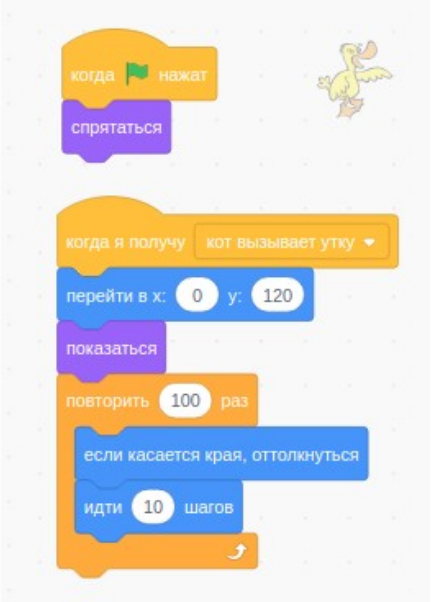
Ответ

Скрипт кота:



Обратите внимание на панель свойств. У кота должна быть включена видимость, потому что если вы ранее работали над проектом и использовали для кота команду "спрятаться", он может быть невидим. Другой вариант в таком случае – использовать команду "показаться" в начале скрипта.

Скрипты второго и третьего спрайта:



Условный оператор "Если …"

Задание

Составьте программу с говорящим котом:

1. Если зажимается клавиша 1 на клавиатуре, кот говорит "Один".
2. Иначе, если нажимается клавиша 2 на клавиатуре, кот говорит "Два".
3. Иначе, если нажимается клавиша 3 на клавиатуре, кот говорит "Три".
4. Когда не нажимается ни одна клавиша клавиатуры, кот должен молчать.

Чтобы кот говорил, используйте команду "сказать Привет!" из фиолетового раздела "Внешний вид".

Ответ

Первый способ (предпочтительный):

Второй способ:

В первом скрипте за один проход по телу цикла проверяется только одно условие. Здесь как только одна из веток "если" возвращает правду, все остальные не проверяются, поскольку вложены в ветки "иначе".

Во втором скрипте за одну итерацию цикла проверяются все четыре условия. В этом нет необходимости, так как и без этого следующий проход по циклу наступает очень быстро, а значит быстро наступит очередная проверка нажатия клавиши.

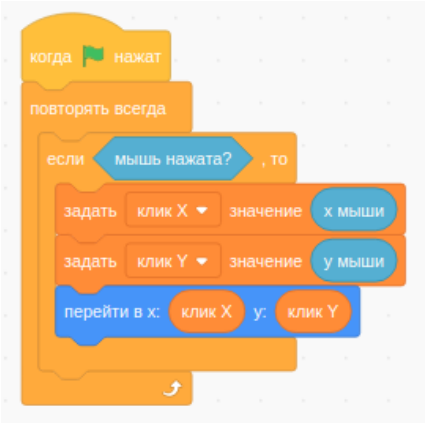
Переменные

Задание

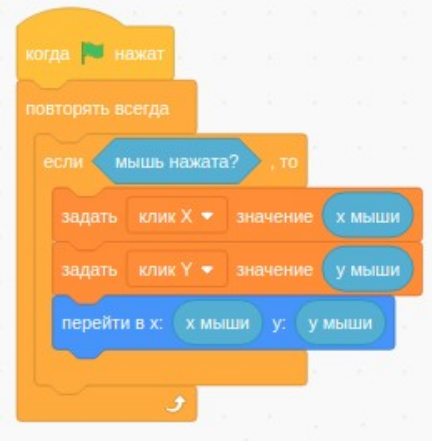
Составьте программу, в которой спрайт оказывается в том месте сцены, над которым была нажата клавиша мышки. При этом на сцене должны отображаться координаты клика (вместо клика также может быть зажатие клавиши мыши).

Ответ

Первый вариант:



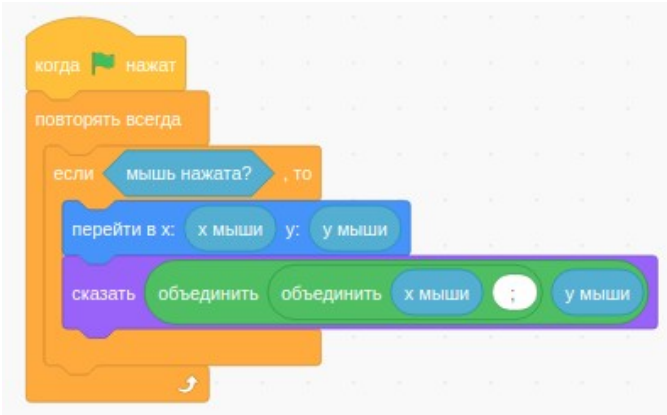
Второй вариант:



Для перемещения спрайта к месту клика создавать собственные переменные не обязательно. Достаточно воспользоваться встроенными блоками "x мыши" и "y мыши", которые возвращают текущее положение курсора мыши.

Однако у этих блоков нет флажка, который позволил бы отобразить переменные на сцене. Поэтому требуются собственные переменные, которым присваиваются значения координат курсора мыши. Чтобы их значения отобразились на сцене, у переменных должен быть включен флажок.

Задачу можно решить по-другому. В задании не сказано, как именно должны выводиться на сцену координаты. Значения переменных можно отобразить с помощью команды "сказать …".



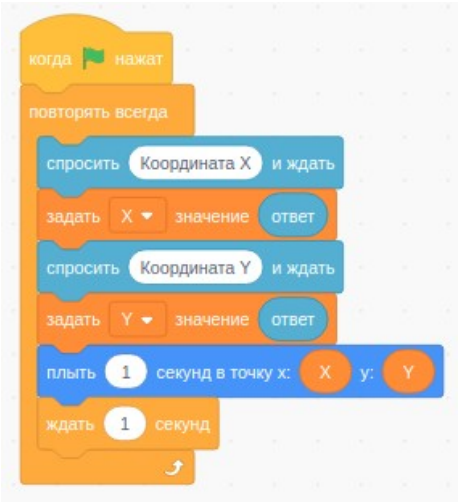
Блоки "объединить" соединяют отдельные части строки в единое целое.

Ввод данных

Задание 1

Составьте программу, в которой пользователь вводит координаты точки. После этого спрайт переходит в указанную точку, ждет секунду и опять запрашивает новые координаты.

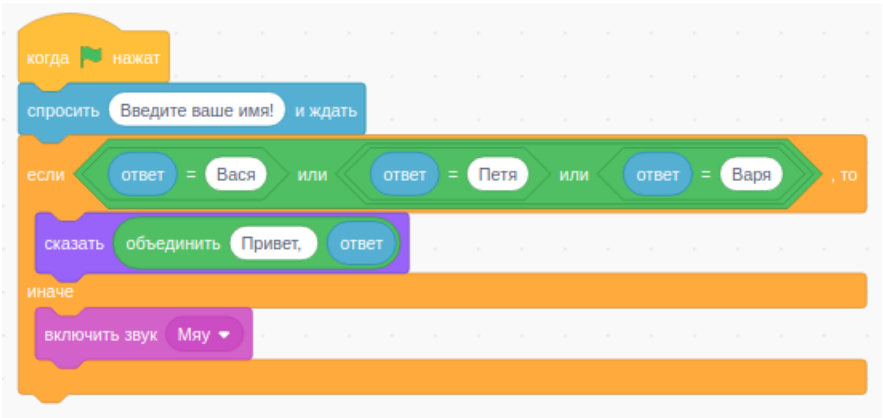
Ответ



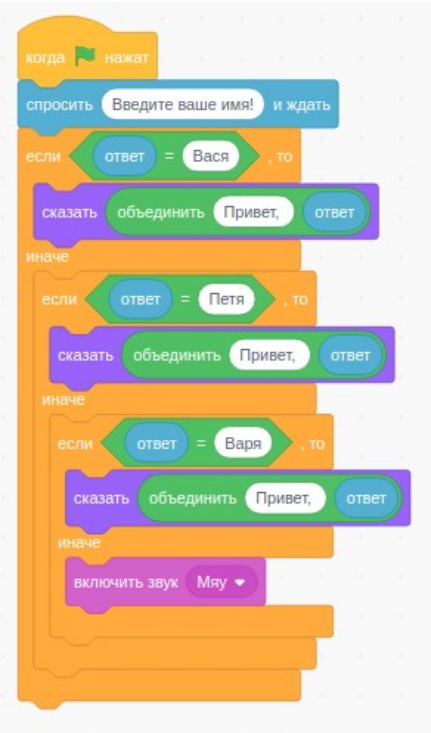
Задание 2

В системе зарегистрированы три пользователя – Вася, Петя и Варя. Если в систему пытается войти один из них, кот его приветствует, говоря его имя, к которому добавляет слово "привет". Если в систему пытается проникнуть посторонний, кот включает аварийную сигнализацию, то есть мяукает.

Ответ



Вместо логических блоков "… или …" можно также использовать несколько блоков "если … иначе …", вложенных друг в друга.



Здесь вместо подвыражения "объединить <Привет, > <ответ>" можно в поле команды "сказать …" просто вписывать "Привет, Вася" в первом случае, "Привет, Петя" во втором и т. д.

Существует более верное решение данной задачи с помощью списков. Хотя для него недостаточно информации из предшествующих уроков, приведем этот вариант.

Списки создаются в разделе "Переменные" и представляют собой коллекции. То есть если в обычной переменной хранится одно значение, то в списках их может быть множество.

Команда "<юзеры> содержит <ответ>" проверяет, есть ли хотя бы одно значение в юзерах, которое совпадает со значением переменной "ответ". Если совпадение найдено, то все выражение возвращает правду (true). Если совпадений не найдено, возвращается ложь (false).

Циклы

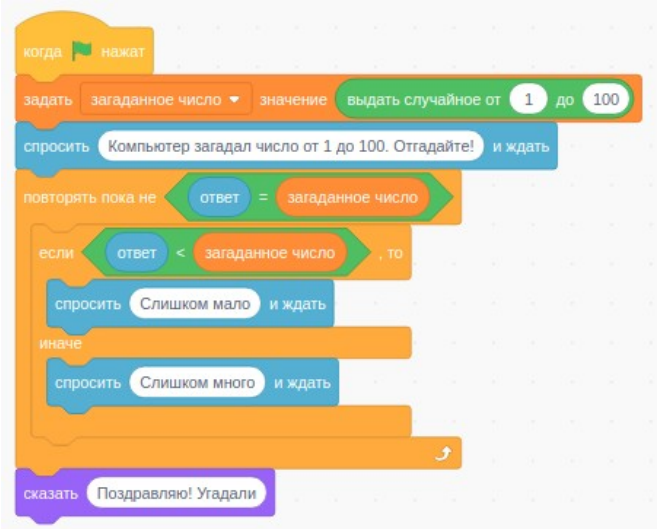
Задание

Переделайте скрипт про угадывание числа так, чтобы

1. компьютер загадывал число от 1 до 100.
2. Если человек вводит число, которое больше загаданного, то программа должна говорить ему "слишком много".
3. Если человек вводит число, которое меньше загаданного, то программа должна говорить ему "слишком мало".

Подумайте, когда программа выполняется, какой способ отгадывания числа самый оптимальный. То есть как нужно отгадывать число, чтобы угадать его за меньшее количество попыток?

Ответ



В цикле достаточно делать одну проверку. Например, что ответ меньше загаданного. Если это условие не выполняется, значит ответ больше загаданного (ветка "иначе"). Числа не могут быть равны, так как в этом случае поток выполнения программы вообще бы не зашел в тело цикла, а сразу бы перешел к команде "сказать Поздравляю! Угадали".

Самый быстрый способ отгадать – это постоянно отвечать числом из середины диапазона, который на каждом шаге уменьшается в два раза. Например, если загадано число в диапазоне от 1 до 100, то число середины – это 50. Тогда:

1. Даем ответ 50. Пусть компьютер говорит "слишком мало".
2. Значит, искомое число где-то в диапазоне от 51 до 100. Середина этого диапазона – это 75.
3. Даем ответ 75. Компьютер говорит "слишком много".
4. Значит, искомое число где-то в диапазоне от 51 до 74. Середина этого диапазона – число 58.
5. Даем ответ 58. Компьютер говорит "слишком много".

6. Значит, искомое число где-то в диапазоне от 51 до 57. Середина – число 54.
7. Даем ответ 54. Если компьютер скажет "слишком много", то загаданное число либо 52, либо 53. Если компьютер скажет "слишком мало", то загаданное число либо 55, либо 56.

При этом совпадение может произойти на более раннем шаге.

Создание спрайтов и костюмов

Задание

Реализуйте сценарий, в котором при каждом нажатии пробела два спрайта попеременно превращаются друг в друга. Например, при каждом нажатии пробела яблоко превращается в бананы, а бананы – в яблоко.

Ответ

В данном задании надо догадаться, что у каждого из спрайтов должно быть по два одинаковых костюма.

Добавить второй костюм, который будет идентичен другому спрайту, можно разными способами. Например, добавить пустой костюм, потом выделить и скопировать с холста рисунок другого спрайта и вставить его в пустой костюм первого спрайта.

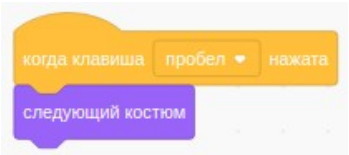
Второй вариант – экспортировать костюм на компьютер, после чего загрузить его.

В обоих случаях (создание нового пустого костюма или загрузка с компьютера) надо пользоваться раскрывающимся меню с иконкой головы кота, которое находится на вкладке "Костюмы" внизу. Выбирать надо пункты "Рисовать" или "Загрузить костюм".

Экспорт костюма осуществляется через правый клик по нему.

Третий вариант – создать один спрайт с двумя костюмами, а потом продублировать его на панели спрайтов (под сценой). При желании можно поменять местами костюмы дубля простым перетаскиванием.

Скрипты же обоих спрайтов просты:



При этом активные костюмы спрайтов на момент нажатия флажка должны отличаться.

Задание можно выполнить по-другому, не используя костюмы, а изменяя координаты спрайтов. Однако в этом случае код получается существенно сложнее, так как для реализации попеременной смены позиций требуется блок "если … иначе …" и попеременное изменение значений переменных.

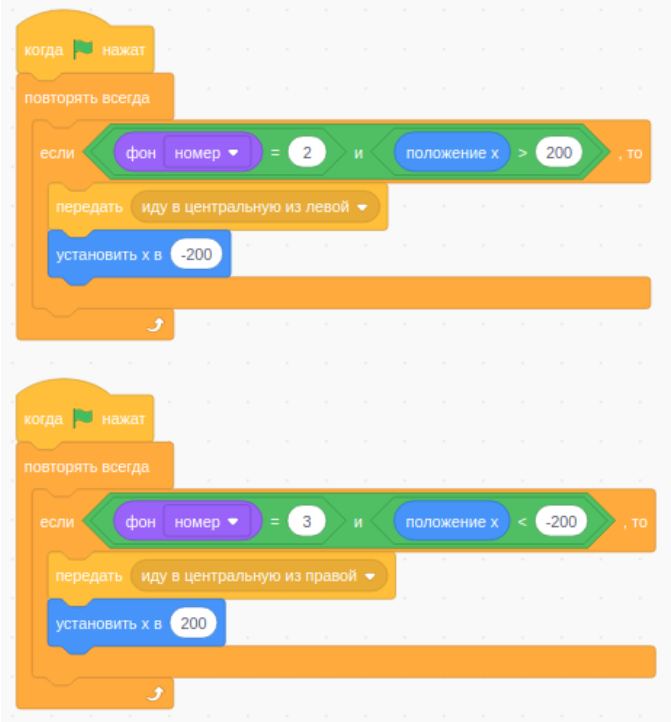
Сцена

Задание

Доделайте приведенный в уроке сценарий так, чтобы спрайт мог возвращаться из левой и правой комнат в центральную.

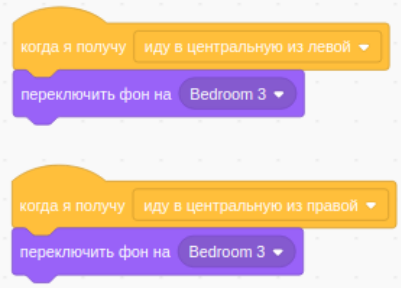
Ответ

Для спрайта следует добавить два скрипта:



Здесь нужно быть внимательным и не перепутать знаки "больше" или "меньше" в условии.

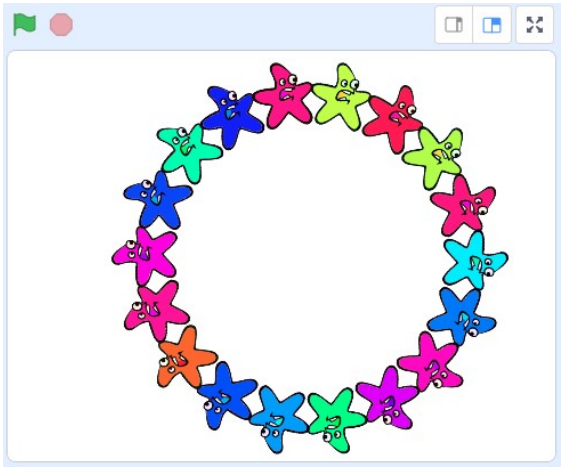
Для сцены также добавляются два скрипта:



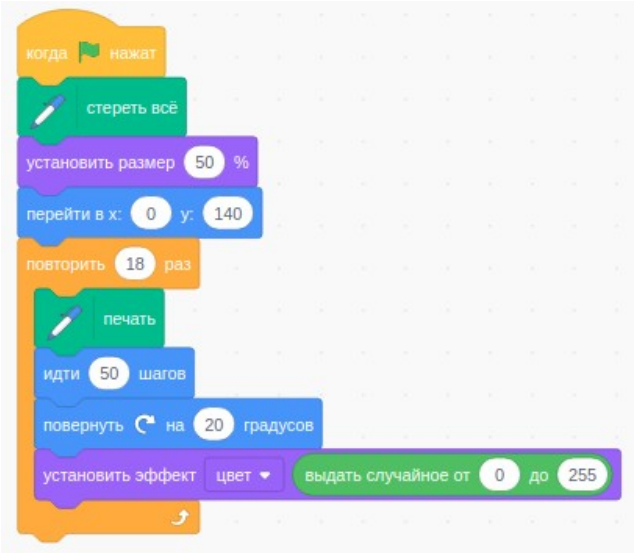
Расширение "Перо"

Задание

Составьте программу, которая при запуске выводит на сцене окружность, состоящую из печатей спрайта.



Ответ



Окружность можно представить как правильный многоугольник с большим количеством сторон. Как известно, сумма внутренних углов правильного многоугольника должна составлять 360 градусов.

Таким образом, у квадрата каждый угол равен 90 градусов. Если мы берем восьмиугольник, то его углы будут по 45 градусов. Если взять 18-ти угольник, его углы будут по 20 градусов, так как 18 * 20 = 360.

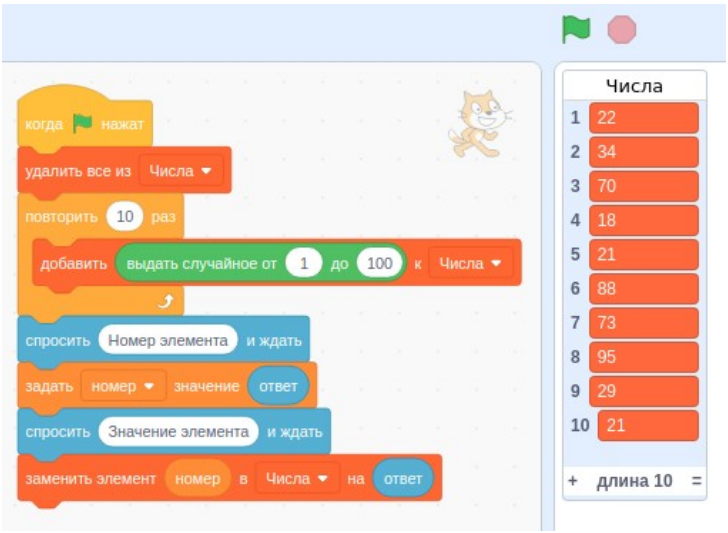
Чем больше повторов, тем меньше должен быть угол поворота, и тем чаще будет отпечатываться спрайт. При этом также надо уменьшить количество шагов. Иначе окружность не поместится на сцене.

Списки

Задание

1. Заполните список десятью случайными числами.
2. Отобразите числа на сцене, установив флажок у переменной списка.
3. Спросите у человека номер элемента, значение которого он хотел бы заменить.
4. Спросите у человека новое значение.
5. Замените в списке старое значение новым в указанной позиции.

Ответ



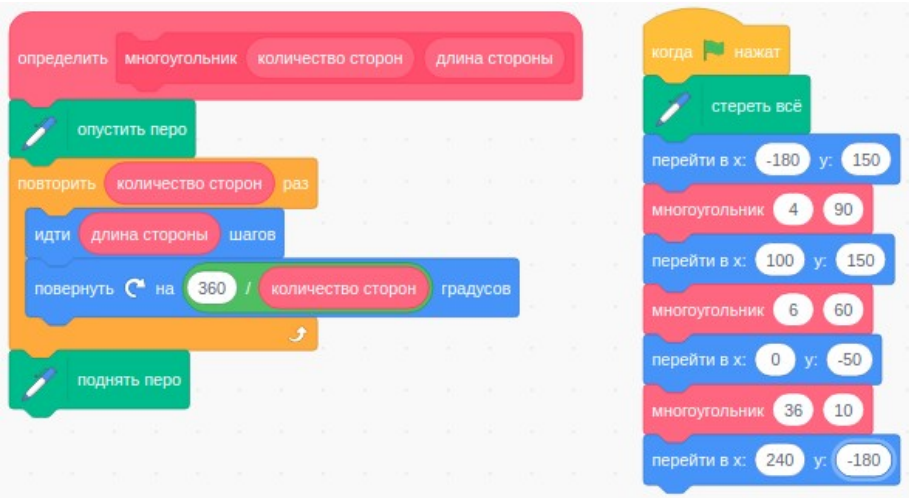
Команда "заменить элемент … в <*списке*> на …" используется, чтобы поменять значение уже существующего элемента, а не добавлять новый. Сначала указывается позиция, в которой происходит изменение, затем – список, последним – новое значение.

Функции

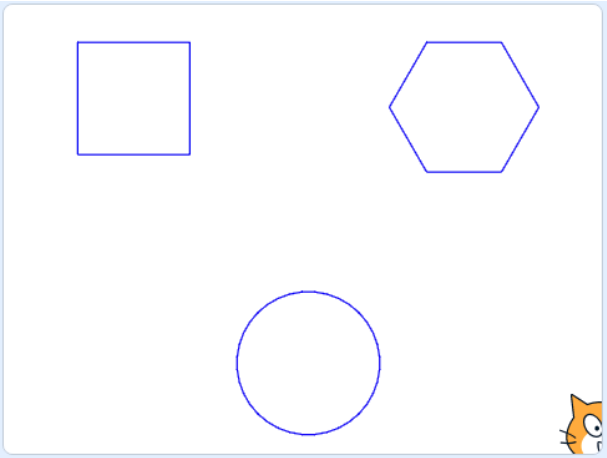
Задание

Создайте функцию, которая рисует на сцене правильный многоугольник. Функция должна принимать два аргумента – количество сторон многоугольника и длину одной стороны.

Ответ



Результат выполнения сценария:



Сумма внутренних углов правильного многоугольника равна 360 градусов. При этом все углы равны между собой. Следовательно, можно вычислить угол поворота спрайта, разделив 360 на количество сторон, которых столько же сколько углов.