Кузнецова Ирина Николаевна

МОДЕЛИРОВАНИЕ В СРЕДЕ ЛОГО ЗАНЯТИЕ 4. РАЗРАБОТКА ПРОЕКТА ПО ШКОЛЬНОМУ ПРЕДМЕТУ

После знакомства с основными инструментами среды, алгоритмическими конструкциями, переменными встает вопрос, на каких примерах совершенствовать умение составлять алгоритмы, какие задачи рассматривать.

Обратимся к геометрическим понятиям *точка, линия* и *плоскость* [5]. В школьной математике плоскость моделируется тетрадным листом или доской, а точка рисуется карандашом или мелом. Линия также ограничена плоскостью, на которой ее проводят.

Формализуем описания изображений этих объектов для среды Лого.

(1) Назовем *точкой* один или несколько пикселей, которые рисуются пером черепашки или штампуются формой. Количество пикселей зависит от толщины пера или формы.

(2) Назовем *плоскостью* область рабочего поля, ограниченную прямоугольником заданного размера.

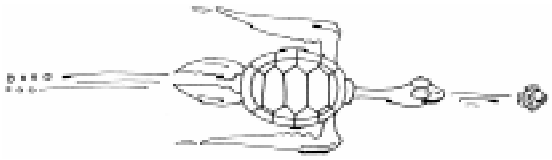
(3) Назовем *линией* множество смежных *точек*, рисуемых пером черепашки *Линия* не может выходить за пределы плоскости.

ЗАДАНИЕ 1.
Приключения точки.
Понятия точки, плоскости, окружности

... Жила была точка. Особенно не задумывалась, что она такое, где находится – не знала таких слов. Однажды точка попала в компьютер.

Компьютер видит – кто-то по экрану гуляет. Спрашивает – «Кто ты?».

«Не знаю», – отвечает точка. «Как так!? В компьютере должно быть все точно известно». Присмотрелся к ней и говорит «А, так это точка!». Обрадовалась точка, что дали ей имя. Стала носиться по полю как сумасшедшая. «Стой, стой», – сказал компьютер, – «раз уж ты у меня живешь, должна обучаться какому-нибудь делу». «Какому?». «Посмотрим. Для начала объясним, как ты выглядишь». «Как это объяснить?», – не поняла точка. «Используя язык программирования, чтобы было понятно и мне и людям! А поможет тебе в этом черепашка по имени **point**. *Она понимает язык Лого».*



Напишите процедуру рисования одной *точки* в соответствии с нашим определением (1).

Следующая процедура удовлетворяет этому определению:

```
to draw_point
forward 1
end
```

«А-а, понятно! Сделай шаг вперед, и нарисуется точка (если будет чем рисовать). А если я хочу в разные места экрана рисовать точки?», – тут же поинтересовалась любознательная точка.

«Ну, это просто, – на случайный угол повернуться и пройти вперед на случайное число шагов» – ответил компьютер:

```
rt random 360 pu fd random 600 pd draw_point
```

«Ой, как интересно! Я хочу много раз так делать!»

«Подожди, подожди!», – воскликнул компьютер, – «Ответь, если ты будешь долго так летать, что получится?»

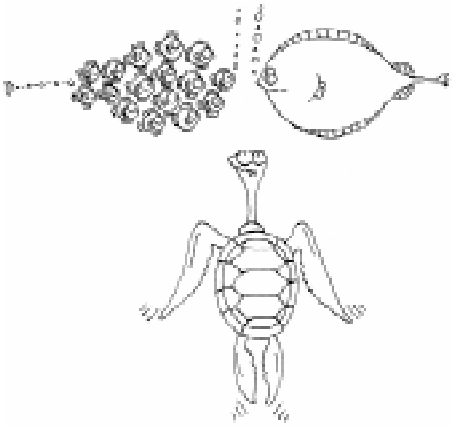
Напишите процедуру по данному описанию. Ответьте, что получится при ее выполнении?

Программа рисования плоскости

```
to flatness
repeat 10000 [rt random 360 pu fd random 600
                                pd draw_point]
end
```

И начала черепашка носиться по экрану, повторяя **draw_point**, **draw_point**, *гордая тем, что научилась новым словам и новым умениям.*

Компьютер следил, как постепенно проявляется плоскость, образуемая множеством точек. И тут он задумался – что



будет, если точка будет долго-долго идти вперед, не меняя направления. «Стой!», – сказал он Черепашке, – «я объясню тебе новое слово – линия...»

Уберем случайность из угла поворота, то есть угол поворота будет всегда постоянный (зададим параметром), и запретим черепашке летать (пусть точки лежат рядом друг с другом) – получится линия с постоянным поворотом.

Напишите процедуру движения точки с постоянным углом поворота.

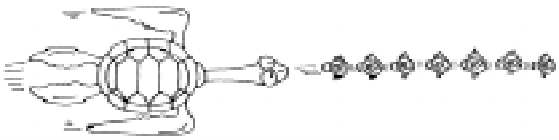
```
to line "angle
pd
repeat 10000 [rt :angle draw_point]
end
```

Оказывается, что точки ложатся на *окружность!*

Теперь пусть угол относительного поворота равен 0.

```
to straight_line
pd
repeat 10000 [rt 0 draw_point]
end
```

Множество точек, для каждой из которых относительный угол поворота равен 0, образует *прямую линию.*



Проверим результат выполнения процедуры **straight_line** при разных значениях начального угла:

```
seth 45 straight_line
seth 90 straight_line
seth 30 straight_line
```

Если начальный угол поворота не 0 или не делится на 90, то вместо одной прямой мы наблюдаем несколько. Это происходит из-за того, что граница экрана прозрачна для черепашки.

Компьютер задумался, как объяснить черепашке, что такое граница? «Послушай», – сказал он черепашке, – «я объясню тебе новое слово, и слово это – число».

ЗАДАНИЕ 2.
Координатная плоскость

Что означает измерить *длину* некоторого отрезка? Это означает, что мы хотим сравнить его с другим отрезком, выбранным в качестве эталона единицы измерения. Единица измерения может иметь собственное имя – метр, дюйм, аршин, парсек и т. п. ... Если единичный отрезок E укладывается в отрезке AB целое число раз, скажем 30 раз, то это число 30 прини-

мается в качестве *длины* измеряемого отрезка...»

На рабочем поле длину отрезка измеряют в *пикселях* или в *шагах черепашки*.

Проведите на рабочем поле горизонтальную прямую линию через точку **home**:

```
pu home seth 90 pd fd 650 home
```

Можно сказать, что каждому *числу* соответствует *точка* на этой прямой. И наоборот, каждой *точке* прямой соответствует определенное *число*. Эта линия есть *числовая ось* в среде Лого [*].

Чтобы узнать, какому числу соответствует точка, в которой находится перо черепашки, введите команду:

```
show xcor
```

Чтобы установить черепашку в точку, соответствующую определенному числу, введите команду:

```
setx 30
setx -100
setx 150 / 7
```

Числовую ось можно провести и вертикально. Выполните команды:

```
pu home pd fd 450 home
```

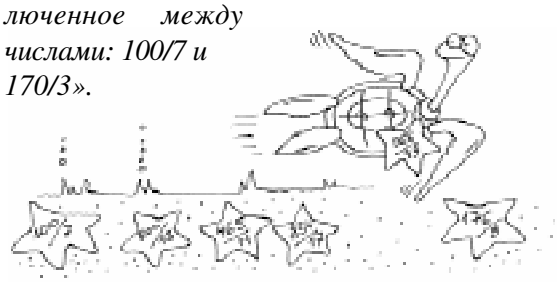
Чтобы узнать, какому числу соответствует точка, в которой находится перо черепашки, введите команду:

```
show ycor
```

Чтобы установить черепашку в точку, соответствующую определенному числу, введите команду:

```
sety 30
sety -100
sety 150 / 7
```

«Вот ка-а-ак!», – протянула точка. – «А я хочу найти какое-нибудь число, заключенное между числами: 100/7 и 170/3».



Для этого выполним команды:

```
setx 100 / 7 stamp setx 170 / 3 stamp
```

Мы ограничили заданный промежуток.

Далее установим черепашку внутри этого промежутка мышью или с помощью команды:

```
setx 100 / 7 + random (round 170 / 3)
```

А теперь определим полученное число с помощью команды:

```
show xcor
```

Имея две числовые оси, для каждой *точки* на *плоскости* можно найти пару *чисел* и притом только одну – расстояние от этой точки до **home** по горизонтальной оси и расстояние от этой точки до **home** по вертикальной [5] (рисунок 1).

Идея обозначать точку на плоскости парой чисел принадлежит французскому математику Рене Декарту (XVII век).

Декартова система координат на плоскости – это две взаимно перпендикулярные числовые оси с общим началом. В среде Лого взаимно перпендикулярные оси проходят через точку **home**. Горизонтальная ось называется осью *X*, вертикальная ось называется осью *Y*.

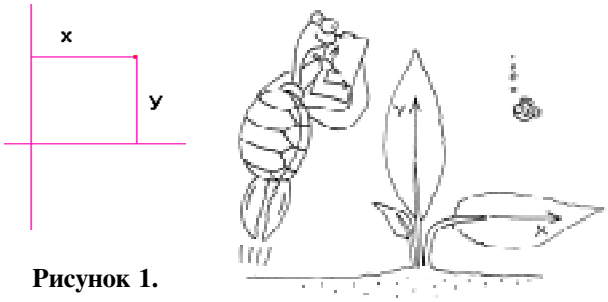


Рисунок 1.

Координаты точки **home**: [0 0].

В Декартовой системе координат можно совершать две операции [5]:

– каждой точке плоскости сопоставлять пару чисел – координаты этой точки;

– по данной паре чисел строить точку на плоскости с этими координатами.

Для измерения координат точки используются датчики *xcor*, *ycor*, *pos*.

[*] Для «настоящей» прямой это неверно, так как существуют иррациональные числа, а компьютер работает только с рациональными числами.

Для перемещения черепашки в точку, указанную координатами служат команды **setx**, **sety**, **setpos**.

ЗАДАНИЕ 3.
Множества точек на координатной плоскости

Выслушала эту историю точка и глубоко задумалась:

«Получается, я не по простой плоскости летаю, а по координатной – Декартовой! И я могу оказаться как в определенном месте, так и в случайном! А что получится, если у множества точек координата по оси X всегда будет равна 30, а по оси У – меняться случайно. Как нарисовать такое множество?».

```
to equ1
pu setx 30
repeat 1000 [pu sety random 450 pd fd 1 pu]
end
```

При выполнении команды **equ1** получилась вертикальная прямая линия, проходящая через точку с координатой $x = 30$.

Итак, все точки прямой, проходящей через точку оси X с координатой 30 и перпендикулярной оси *X*, связаны соотношением $x = 30$.

«Какая сообразительная точка!», – воскликнул компьютер, – «А теперь нарисуй нам множество точек, для которых выполняется соотношение: $-100 \le X \le 100$ *и* $-100 \le Y \le 100$*».*

```
to sys_inequ1
repeat 10000 [pu setx -100 + random 200
        sety -100 + random 200 pd fd 1 pu]
end
```

«О-о-о! Так ведь это квадрат!», – удивилась точка, – «и каждая сторона у него имеет длину 200 шагов».

«Интересно», – сказал компьютер, – «задаем числа, а получаем геометрический объект!»

Управляя областью расположения случайных точек на координатной плоскости, мы ограничиваем разные участки на плоскости. Например, можно посадить лес только в правой части рабочего поля, усыпать звездами только небо, нарисовать звездное небо в окне...

Правда, сначала требуется знать максимальные размеры нашей «плоскости». Для этого надо задать переменные, которые будут доступны всем процедурам – глобальные переменные. Воспользуемся командой **createprojectvar** для создания переменных **f_w** («ширины плоскости») и **f_l** («длины плоскости»):

```
createprojectvar "f_w
createprojectvar "f_l
```

Эти команды достаточно выполнить один раз, и созданные таким образом переменные будут всегда сохраняться вместе с проектом. Значения глобальных переменных проекта меняются командами:
setf_w <*значение1*> и **setf_l** <*значение1*>

Например:

```
setf_w 360
setf_l 388
```

Для наглядности ограничим «плоскость» оградой определенного цвета (пусть это будет цвет с номером 9 – черный).

Напишите процедуру рисования прямоугольника, ограничивающего плоскость. Стороны прямоугольника лучше задать параметрами. Используя эти данные, вычислите координаты левой нижней вершины. Толщину пера задайте не менее 5 пикселями.

Рисование ограды

```
to fence :w :l
setc 9 setpensize 5 pu
; установка в левый нижний угол
setx minus :w / 2
sety minus :l / 2 pd
repeat 2[fd :l rt 90 fd :w rt 90]
setpensize 1 pu home
setc 9 + random 135
end
```

Неплохо иметь и процедуру стирания рисунка внутри ограды:

```
to f_cg
cg fence f_w f_l
end
```

Теперь разделим нашу плоскость на две вертикальные области таким образом:

– установим черепашку `point` в некоторое место экрана: `pu setx 50 sety -100`

– разделим рабочее поле вертикальной линией на две части:

```
pd sety -100 + (f_l / 2 -(-100))
sety -100
sety -100 - (f_l / 2 + (-100)) pu
```

«Ну, точка», – сказал компьютер, – «поработай художником! Твоя задача – узнавать координаты множества случайных точек в указанных пределах. В эти точки надо приклеивать указанный рисунок».

Пусть справа от линии $x = 50$ надо посадить лес из деревьев, а слева – засеять поле цветами.

Это лес:

```
to wood
setsh 6
repeat 50[pu setx 50 + random (f_w / 2 - 50)
                    sety  random f_l stamp]
setsh 0
end
```

А это цветы:

```
to flowers
setsh 7
repeat 50 [pu setx 50 - random (f_w/2 - (-50))
                     sety random f_l stamp]
setsh 0
end
```

Число `f_w / 2` является самым большим на оси *X* для нашей экранной координатной плоскости. Число 50 на оси *X* является границей, разделяющей области. Для получения координаты по оси *X* для правой области надо к числу 50 прибавить случайное число. Для получения координаты по оси *X* для левой области надо от числа 50 отнять случайное число.

Вычислите самое большое случайное число, которое может получиться в программе *wood*.

Вычислите самое большое случайное число, которое может получиться в программе *flowers*.

Для программы *wood* это число равно *f_w*/2 – 50 = 360/2 – 50 = 180 – 50 = 130. Как раз столько шагов осталось от границы до *правого* края ограды.

Для программы *flowers* это число равно *f_w*/2 + 50 = 360/2 + 50 = 180 + 50 = 210. Как раз столько шагов осталось от границы до *левого* края ограды.

Сотрите рисунок на рабочем поле командой `f_cg`.

Снова установите черепашку в начальное положение и разделите поле на 2 горизонтальные области:

– установим черепашку `point` в некоторое место экрана:

```
pu setx 50 sety -100
```

– разделим рабочее поле горизонтальной линией на две части:

```
pd setx 50 + (f_w / 2 -50))
setx 50
setx 50 - (f_w / 2 + 50)) pu
```

Теперь представим себе, что мы попали в старинный город (ниже линии $y = -100$), а над ним (выше линии $y = -100$) – звездное небо.

Рисунок 2.

Это звездное небо:

```
to sky
pu sety -100 + random (f_l / 2 - (-100))
setc 117 pd fill setpensize 5 setc 45
repeat 100 [pu sety -100 +
                  random (f_l / 2 - (-100))
                setx random f_w pd fd 1 pu]
setc 14
end
```

Это старинный город:

```
to town
setsh 27
repeat 20 [pu sety -100 - random (f_l / 2 -100)
                     setx random f_w stamp]
setsh 0
end
```

Исследуйте границы областей, в которых располагается звездное небо и замки старинного города при выполнении команд: **sky** и **town** (см. рисунок 2).

Сколько звезд сияет на небе? Сколько замков в старинном городе?

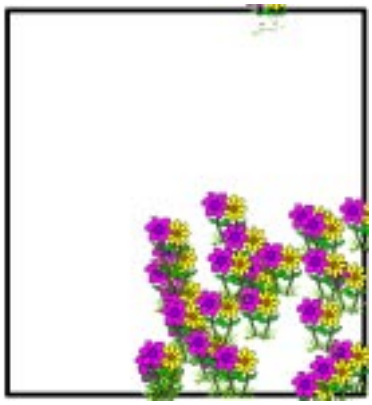


Рисунок 3.

Попробуйте теперь посеять цветы на части плоскости, расположенной левее

линии, для точек которой $X >$ **:bx**, и ниже линии, для точек которой $Y <$ **:by**.

Это сад:

```
to garden :bx :by
setsh 7
repeat 30 [pu setx :bx + random (f_w / 2 - :bx )
          sety :by - random (f_l / 2 - :by)
                                    stamp]
pu setx :bx sety :by setsh 0
end
```

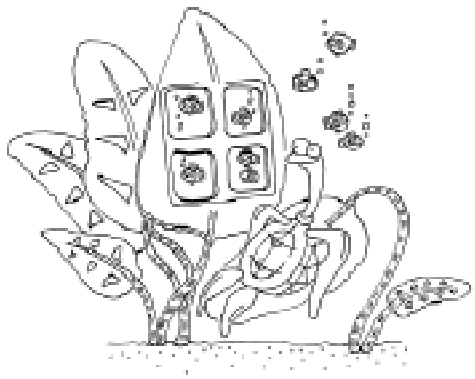
Выполняйте поочередно команды

```
garden 30 50
garden -30 50
garden 30 -10
garden -30 -10
```

(См. рисунок 3).

Исследуйте области расположения множества цветов в каждом случае.

Следующая задача. Вы смотрите в прямоугольное окно, а за окном – ночь и падает снег. Границы окна заданы коорди-



натами левого верхнего угла и правого нижнего угла (рисунок 4).

```
to snow :bxl :byl :bxr :byr
ht setc 95 pd fill ;стена
   setc 0 pu setx :bxl sety :byl ;окно
   pd setx :bxr sety :byr setx :bxl sety :byl
   pu setx :bxr / 2 sety :byl / 2
   pu setc 9 pd fill setpensize 4
```

```
repeat 100 [pu setx :bxl + random (:bxr - :bxl)
            sety :byr + random (:byl - :byr)
         pd setc 0 fd 1 wait 1 setc 9 bk 1 ]
pu setx :bxl sety :byl st
end
```

Введите команду:

```
snow -50 50 40 -100
```

Убедитесь, что черепашка перемещается только в пределах окна.

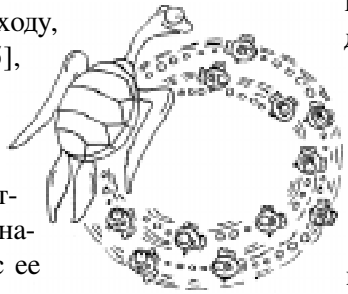
ЗАДАНИЕ 4.
Функции и графики

«Может ли одна координата, например Y, зависеть от другой – X? Я хочу сказать, что при изменении X как-то будет меняться и Y», – спросила точка. «Конечно!», – воскликнул компьютер, – «В этом случае X называется переменной. Когда математики придумали переменную, оказалось возможным на математическом языке описать движение тела и другие, очень важные понятия физики».

Математика рассматривает соотношения между координатами. Уравнение $y = f(x)$ определяет некоторую кривую. При этом, конечно, надо убедиться в том, что:

– для каждой точки этой кривой справедливо соотношение $y = f(x)$,

– любая точка, для которой $y = f(x)$, лежит на этой кривой.



Согласно подходу, изложенному в [5], составим уравнение *окружности*.

Пусть центр окружности находится в точке с координатами [0 0], а радиус ее задается параметром `:r`.

Напишите программу рисования осей координат и программу радиуса, исходящего из центра под случайным углом. Из второго конца радиуса опустите перпендикуляры на оси *X* и *Y* (рисунок 5).

Координатные оси:

```
to cooraxel
f_cg
pd setx 2 * f_w pu home
pd sety 2 * f_l pu home
end
```

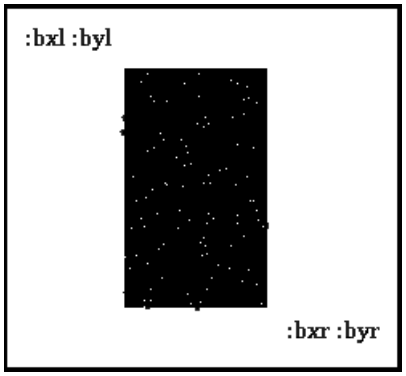


Рисунок 4.

Радиус из центра координатной плоскости:

```
to circ0 :r
seth random 360
pd fd :r make «rpos pos
setc 14 setx 0
pu setpos :rpos
pd sety 0
pu home
end
```

Из рисунка видно, что радиус есть гипотенуза прямоугольного треугольника, а катеты – это отрезки, равные по длине координатам второго конца радиуса.

Из теоремы Пифагора известно, что: $r^2 = x^2 + y^2$

Отсюда можно вывести зависимость координаты *Y* от текущего значения координаты *X* для точки, которая находится на конце радиуса, то есть *уравнение окружности* (сразу запишем его на языке Лого):

```
:y = sqrt ( :r * :r - xcor * xcor)
```

Давайте попробуем менять *X* от значения 0 до значения `:r`. При этом значение *Y* будет иметь два симметричных значения – со знаком «+» и знаком «–».

Напишите программу для вычисления *Y*.

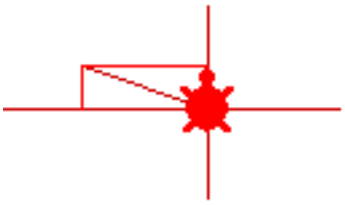
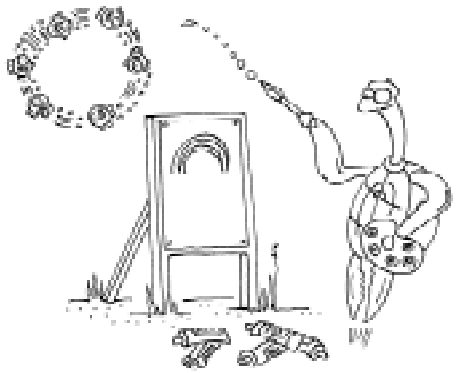


Рисунок 5.

```
to ciryx :r
pu home setsh 13
make "xt 0
repeat :r + 1 [make "yt sqrt (:r * :r -
                              :xt * :xt) pu
       setx :xt sety :yt stamp
       sety minus :yt stamp
       make "xt :xt + 1]
setsh 0
end
```

При выполнении команды черепашка рисует половину окружности!



Установим начальное значение *X* = `minus :r`. Тогда число повторений надо увеличить в 2 раза:

```
to ciryx :r
pu home setsh 13
make "xt minus :r
repeat 2 * :r + 1 [make "yt sqrt (:r * :r -
                              :xt * :xt) pu
      setx :xt sety :yt stamp
      sety minus :yt stamp
      make "xt :xt + 1]
setsh 0
end
```

Подумайте, почему число повторений равно выражению 2 * `:r` + 1?

Если координаты центра окружности выражены переменными `:a` по оси *X* и `:b` по оси *У*, то длина радиуса вычисляется по следующей формуле (запись на языке Лого):

```
:yt = sqrt (:r * :r - (:xt - :a) *
                          (:xt - :a)) + :b
```

Напишите программу, которая рисует такую окружность.

```
to ciryxab :r :a :b
pu home setsh 13
make "xt (minus :r) + :a
repeat 2 * :r + 1
       [make "ytp :b + sqrt ((:r * :r -
       (:xt - :a) * (:xt - :a)))
       pu setx :xt sety :ytp stamp
       make "ytm :b - sqrt ((:r * :r -
       (:xt - :a) * (:xt - :a)))
       sety :ytm stamp
       make "xt :xt + 1]
setsh 0
end
```

Проверьте работу программы, вводя команды (рисунок 6):

```
cooraxel ciryxab 50 20 40
cooraxel ciryxab 50 -20 -40
```

«Вот это да!», – восхитилась точка, – «а мы такую же линию рисовали в задании 1 другим способом, используя команды `rt :angle fd 1`*». «Правильно», – подтвердил компьютер, – «только здесь мы указываем расстояние до центра, а в задании 1 даже не умели его определить».*

В рассмотренном выше уравнении окружности несколько переменных величин связаны зависимостью – если меняется значение одной из переменных, то меняются значения других.

В реальных процессах участвует много связанных между собой переменных. Однако, если рассматривать основные из них и связи между ними, то удается построить математическую модель, то есть описать зависимость между перемен-

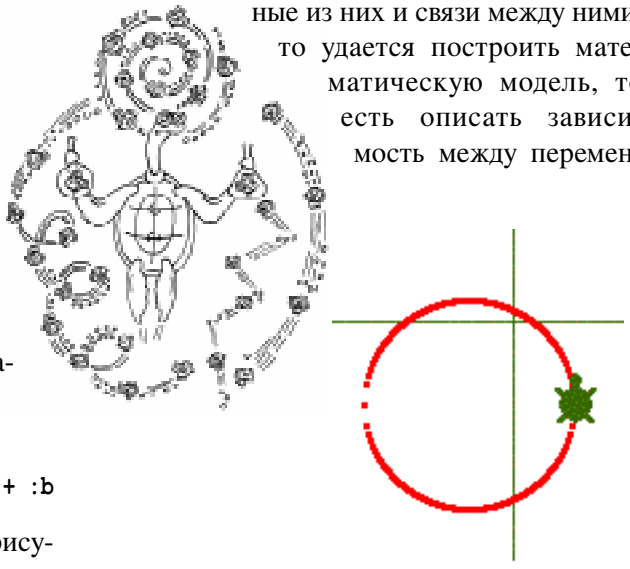


Рисунок 6.

ными в виде уравнения. Оказалось, что при описании самых разных процессов можно применять одни и те же простейшие зависимости [5]:

1) переменные x и y связаны прямо пропорциональной зависимостью, если их отношение постоянно: $x/y = k$ или $y = k * x$, где k – постоянное число, не равное 0;

2) переменные x и y связаны обратно пропорциональной зависимостью, если их произведение постоянно: $x * y = c$ или $y = c/x$, где c – постоянное число и не равно 0;

3) переменная y зависит от переменной x по квадратичному закону, если y можно вычислить по формуле $y = a * x * x$, где a – постоянное число и не равно 0.

Напомним еще два определения, которые вы изучали на математике [5].

«1) Переменная y является функцией от переменной x, если задана такая зависимость между этими переменными, которая позволяет для каждого значения x однозначно определить значение y.

2) Графиком функции f называется множество точек на плоскости с координатами $[x; f(x)]$, где x пробегает область определения функции f».

Программу для графического отображения зависимости

```
:yt = sqrt (:r * :r - (:xt - :a) * (:xt - :a)) + :b
```

мы и писали в соответствии с этим определением.

Составьте аналогичные программы для простейших зависимостей, перечисленных выше.

1) Прямо пропорциональная зависимость – функция $y = k * x$.

```
to kx :k
pu setsh 13
make "xt minus f_w / 2
repeat f_w [pu setx :xt make "yt :k * :xt
    if (abs :yt) < f_l / 2 [sety :yt stamp]
    make "xt :xt + 1]
end
```

Введите команды:

```
kx 2 kx -2 kx 1 / 2 kx 0,3
```

(См. рисунок 7)

2) Обратно пропорциональная зависимость – функция $y = c/x$. Здесь надо учесть, что значение x не может быть равно 0.

```
to c/x :c
pu setsh 13
make "xt minus f_w / 2
repeat f_w [pu setx :xt
     if not (:xt = 0)[make "yt :c / :xt]
     if (abs :yt) < f_l / 2 [sety :yt stamp]
     make "xt :xt + 1]
end
```

Введите команды:

```
c/x 200 c/x -200
```

(См. рисунок 8).

3) Квадратичная зависимость – функция $y = a * x * x$

```
to a*x*x :a
pu setsh 13
make "xt minus f_w / 2
repeat f_w [pu setx :xt
     make "yt :a * :xt * :xt
     if (abs :yt) < f_l / 2 [sety :yt stamp]
     make "xt :xt + 1]
end
```

Введите команды:

```
a*x*x 0,1 a*x*x 0,01 a*x*x 0,001
```

(См. рисунок 9).

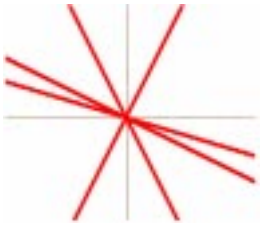


Рисунок 7.

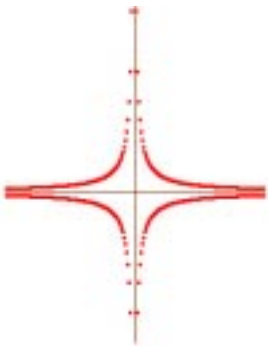


Рисунок 8.

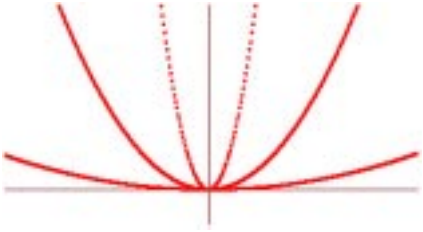


Рисунок 9.

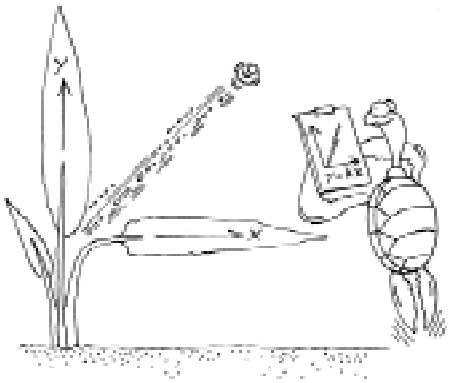
В приведенных выше программах область определения функции задана в пределах ограниченной координатной плоскости – от `minus f_w / 2` до `f_w / 2`.

Значение переменной *X* всегда меняется от наименьшего значения к наибольшему с шагом 1.

Значение функции – координата по оси *Y* вычисляется по указанной формуле. Если значение координаты *Y* выходит за пределы ограниченной плоскости (то есть `abs :yt > f_l/2`), то соответствующая точка не рисуется.

Домашнее задание

Используя уже рассмотренные алгоритмы построения графика простейших зависимостей, напишите программы построения графиков для более сложных случаев, запустите эти программы и охарактеризуйте полученные графики:



1) a) y = abs x; b) y = kx + b
2) a) y = c / (x + b); b) y = –c /(x – b)
3) a) y = a*x*x + bx – c; b) y = a*x*x – c

ЗАКЛЮЧЕНИЕ

Надеюсь, наше занятие убедило вас в том, что компьютер помогает лучше понять многие проблемы математики. Язык программирования очень близок математическому языку. Формулы записываются почти одинаково. В формулах можно использовать переменные, а числовой результат получится, если вы зададите переменным численные значения. На языке программирования можно записать и рассуждения, которые вы обычно делаете устно или записываете в тетради с помощью специальных обозначений. Рассуждения на языке программирования записываются с помощью команд разветвления и повторения.

Описывая математические зависимости на языке программирования, вы еще лучше разбираетесь в них. Ошибки, которые вы допускаете, приводят не к плохим оценкам, а помогают устранить неясности.

Компьютер работает, как старательный ученик, – то, что однажды выучил, выполняет всегда правильно, а если что-то не понимает, спокойно сообщает – «не могу выполнить».

Литература

1. Информатика. 7–9 класс. Базовый курс. Теория. / Под ред. *Н.В. Макаровой*. СПб.: Питер, 2000, тема 6.

2. Информатика. 7–9 класс. Базовый курс. Практикум по информационным технологиям. Тема 7 / Под ред. *Н.В. Макаровой*. СПб.: Питер, 2000, раздел 1 .

3. Информатика. 7 класс. Базовый курс. Задачник по моделированию. / Под ред. *Н.В. Макаровой.* СПб.: Питер, 2000, раздел 1.

4. Информатика. 7 класс. Методическое пособие для учителей / Под ред. *Н.В. Макаровой.* СПб.: Питер, 2003, части 1 и 3.

5. *М.И. Башмаков.* Математика. Издание второе, переработанное и дополненное, М.: «Высшая школа», 1994.

Кузнецова Ирина Николаевна, учитель информатики школы № 640, координатор секции Лого-Лего Международной конференции «Школьная информатика и проблемы устойчивого развития».