

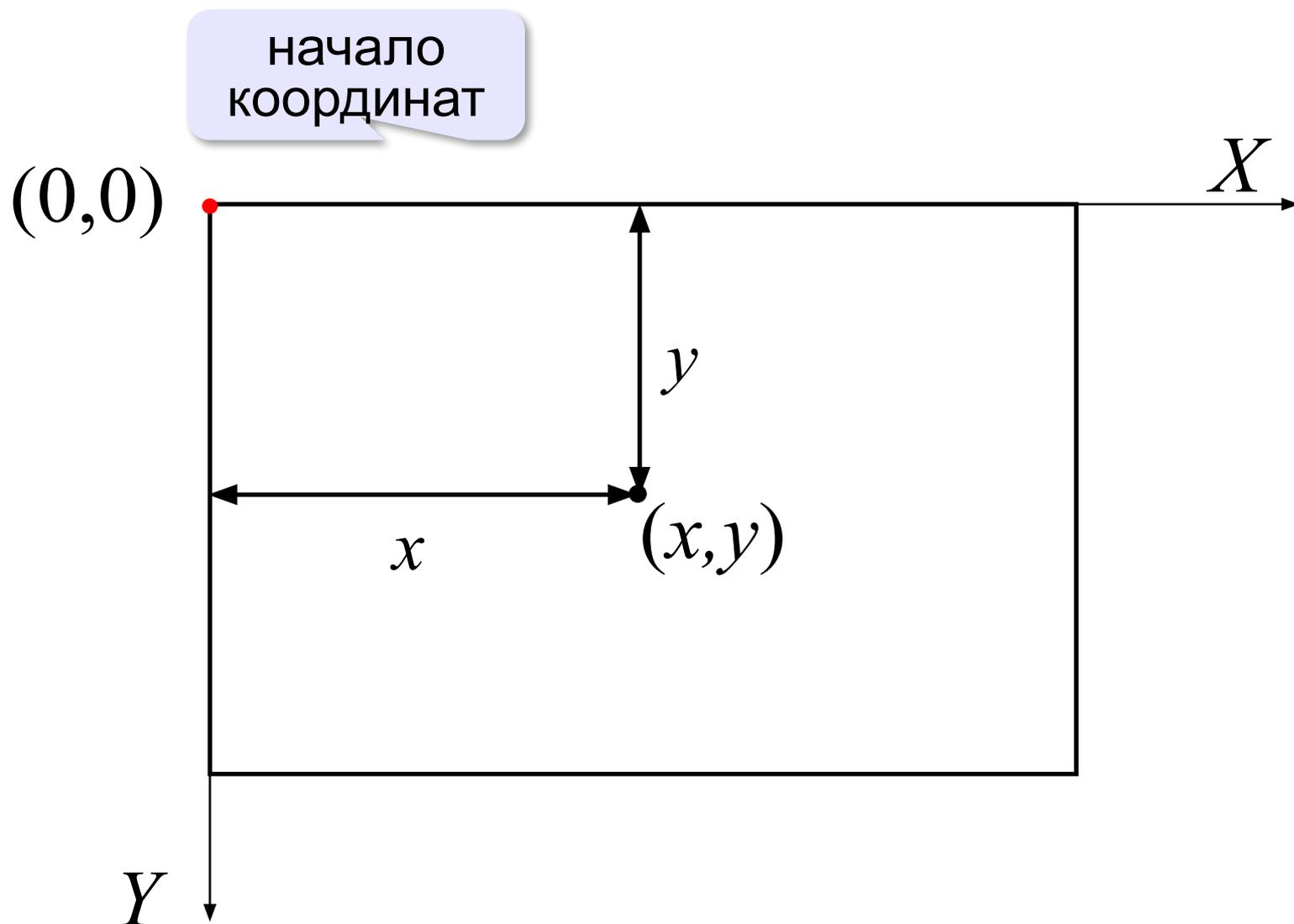
# Программирование на Python: графика

1. [Простые программы](#)
2. [Процедуры](#)
3. [Циклы](#)
4. [Штриховка](#)
5. [Закрашивание областей](#)
6. [Построение графиков функций](#)
7. [Анимация](#)
8. [Игры](#)

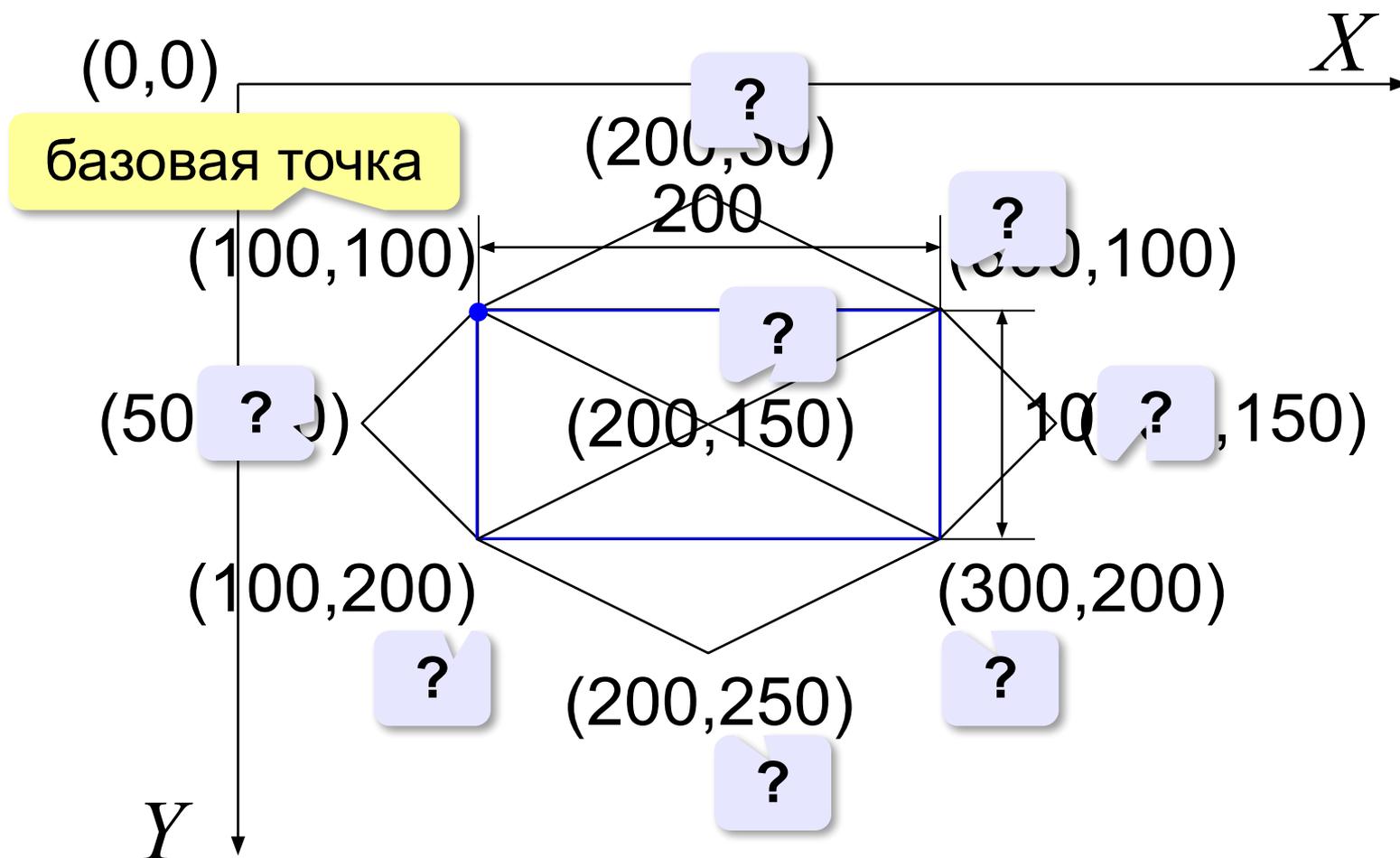
# Программирование на Python: графика

## 1. Простые программы

# Система координат



# Определение координат



# Управление цветом

---

## Подключение графического модуля:

```
from graph import *
```

подключить все функции модуля graph

## Цвет линий:

```
penColor ( "red" )
```

white, black, gray, navy, blue, cyan,  
green, yellow, red, orange, brown, maroon,  
violet, purple, ...

[http://www.science.smith.edu/dftwiki/index.php/Color\\_Charts\\_for\\_TKinter](http://www.science.smith.edu/dftwiki/index.php/Color_Charts_for_TKinter)

## Толщина линий:

```
penSize ( 2 )
```

## Цвет заливки:

```
brushColor ( "green" )
```

# Управление цветом (RGB)

Цвет в формате RGB:

"yellow"

```
penColor ( 255 , 255 , 0 )
```

**R**(red)  
0..255

**G**(green)  
0..255

**B**(blue)  
0..255

```
brushColor ( 255 , 0 , 255 )
```

"magenta"

```
penColor ( 0 , 255 , 255 )
```

"cyan"

```
brushColor ( 255 , 255 , 255 )
```

"white"

```
penColor ( 0 , 0 , 0 )
```

"black"

# Примитивы (простейшие фигуры)

$(x, y)$



```
penColor(0, 0, 255)  
point(x, y)
```

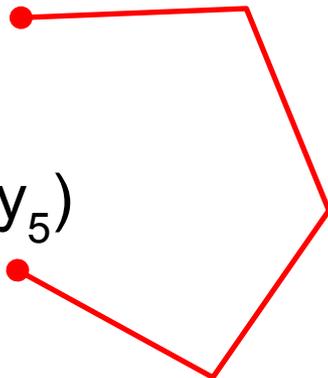
$(x_1, y_1)$



$(x_2, y_2)$

```
penColor(0, 255, 0)  
line(x1, y1, x2, y2)
```

$(x_1, y_1)$



$(x_2, y_2)$

$(x_3, y_3)$

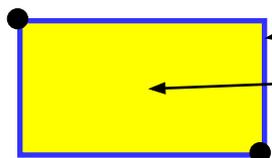
$(x_4, y_4)$

$(x_5, y_5)$

```
penColor(255, 0, 0)  
moveTo(x1, y1)  
lineTo(x2, y2)  
lineTo(x3, y3)  
lineTo(x4, y4)  
lineTo(x5, y5)
```

# Примитивы (простейшие фигуры)

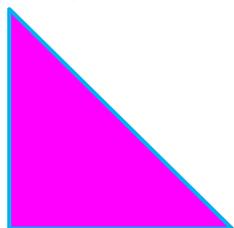
(10, 20)



(50, 40)

```
penColor("blue")  
brushColor("yellow")  
rectangle(10, 20, 50, 40)
```

(10, 10)

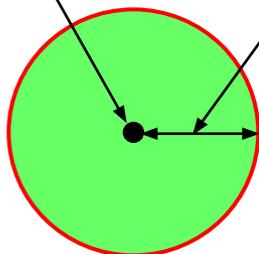


(10, 50)

(50, 50)

```
penColor("cyan")  
brushColor("magenta")  
polygon([ (10, 10), (50, 50),  
          (10, 50), (10, 10) ] )
```

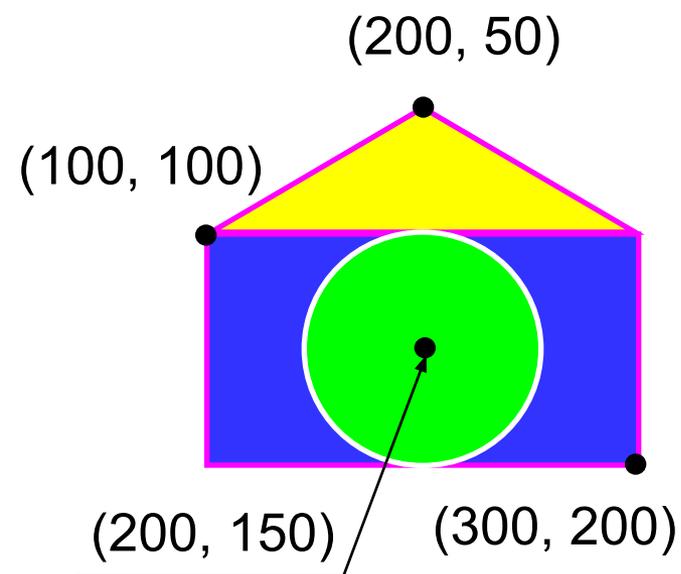
(50, 30)



R=20

```
penColor("red")  
brushColor("green")  
circle(50, 30, 20)
```

# Пример

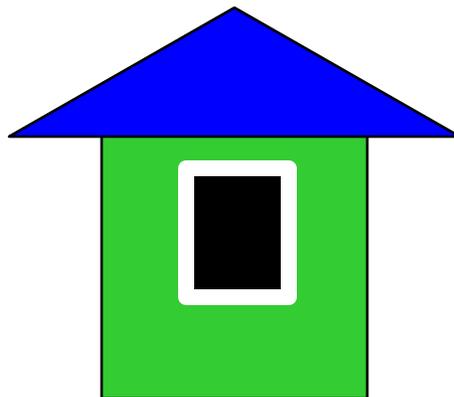


```
from graph import *  
penColor("magenta")  
brushColor("blue")  
rectangle(100, 100, 300, 200)  
brushColor("yellow")  
polygon([(100, 100), (200, 50),  
         (300, 100), (100, 100)])  
penColor("white")  
brushColor("green")  
circle(200, 150, 50)  
run()
```

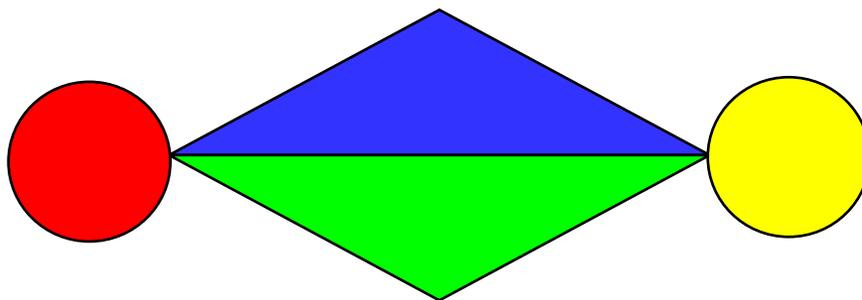
# Задачи

---

«3»: «ДОМИК»



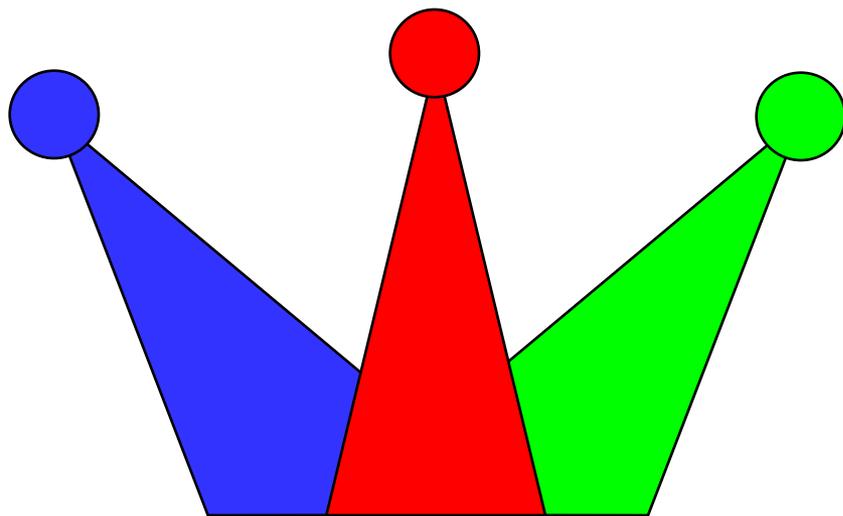
«4»: «Лягушка»



# Задачи

---

## «5»: «Корона»



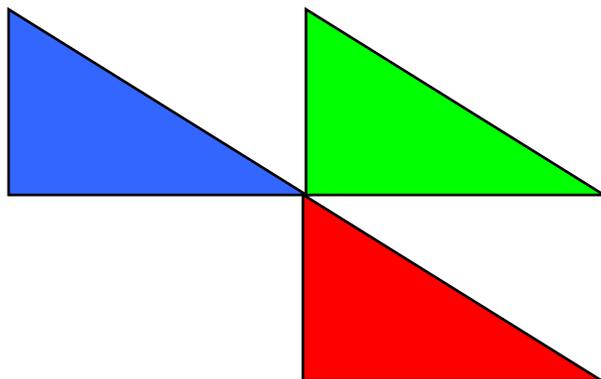
# Программирование на Python: графика

## 2. Процедуры

# Процедуры

---

**Задача:** Построить фигуру:



**?** Можно ли решить известными методами?

**Особенность:** Три похожие фигуры.

общее: размеры, угол поворота

отличия: координаты, цвет

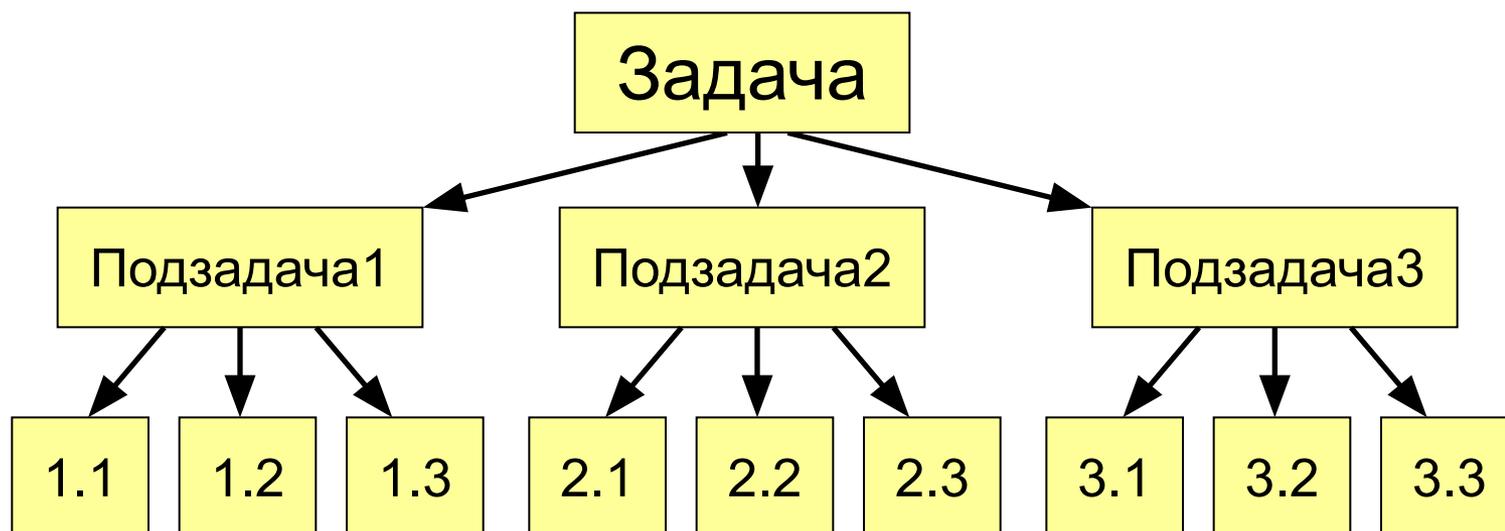
**?** Сколько координат надо задать?

# Процедуры (подпрограммы)

**Процедура** – это вспомогательный алгоритм, который предназначен для выполнения некоторых действий.

## Применение:

- выполнение одинаковых действий в разных местах программы
- разбивка программы (или другой процедуры) на подзадачи для лучшего восприятия

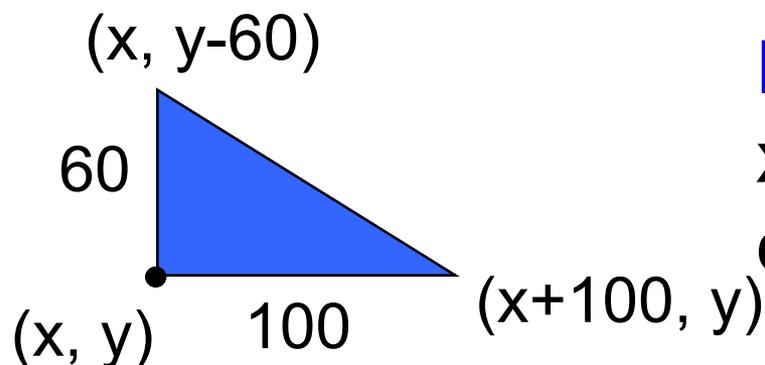


# Как построить процедуру?

- выделить одинаковые или похожие действия (*три фигуры*)
- найти в них общее (*размеры, форма, угол поворота*) и отличия (*координаты, цвет*)
- отличия обозначить как **переменные**, они будут **параметрами** процедуры



**Параметры** – это данные, от которых зависит работа процедуры.

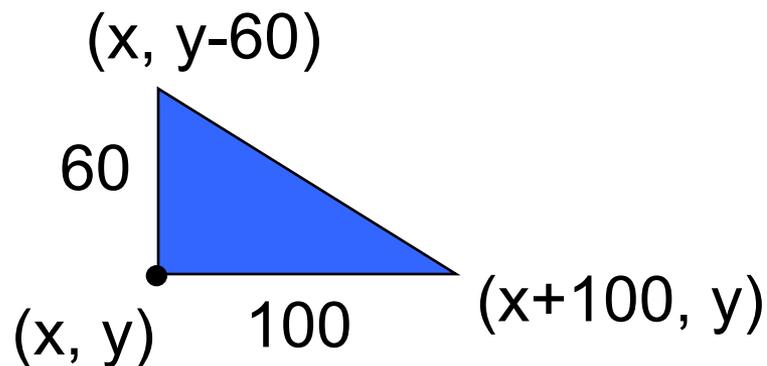


**Параметры:**

$x, y$  – координаты угла

$c$  – цвет заливки

# Процедура



определить  
(*define*)

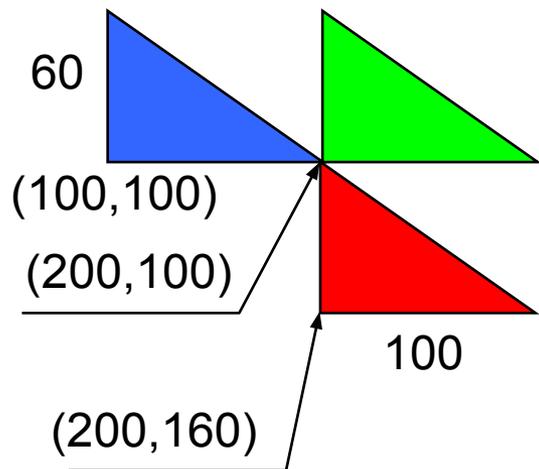
название

параметры

отступ

```
def treug(x, y, c):  
    brushColor(c)  
    polygon([ (x, y), (x, y-60),  
              (x+100, y), (x, y) ] )
```

# Программа с процедурой



ВЫЗОВЫ  
процедуры

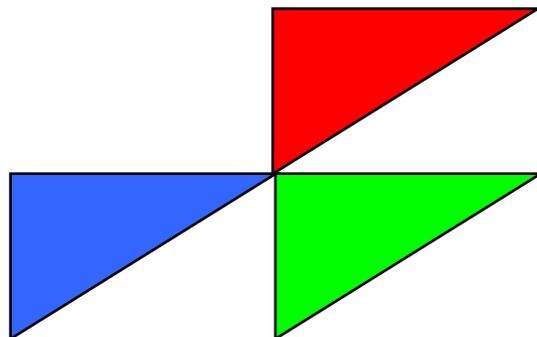
```
from graph import *  
def treug(x, y, c):  
    brushColor(c)  
    polygon([(x,y), (x,y-60),  
            (x+100,y), (x,y)] )  
    penColor("black")  
treug(100, 100, "blue")  
treug(200, 100, "green")  
treug(200, 160, "red")  
run()
```

аргументы (значения  
параметров)

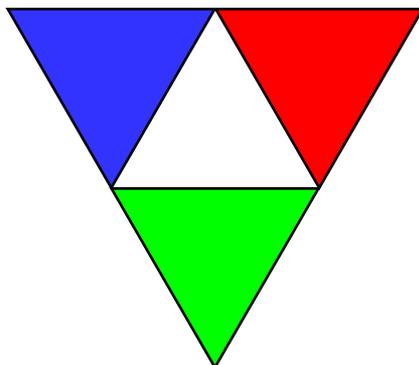
# Задания

---

«3»: Используя одну процедуру, построить фигуру.



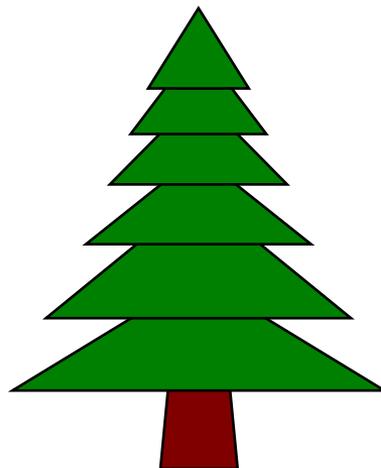
«4»: Используя одну процедуру, построить фигуру.



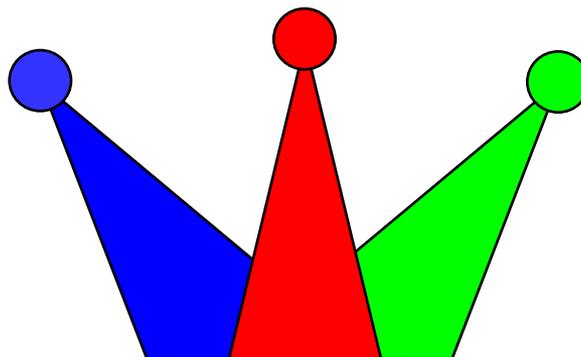
# Задания

---

«5»: Используя одну процедуру, построить фигуру.



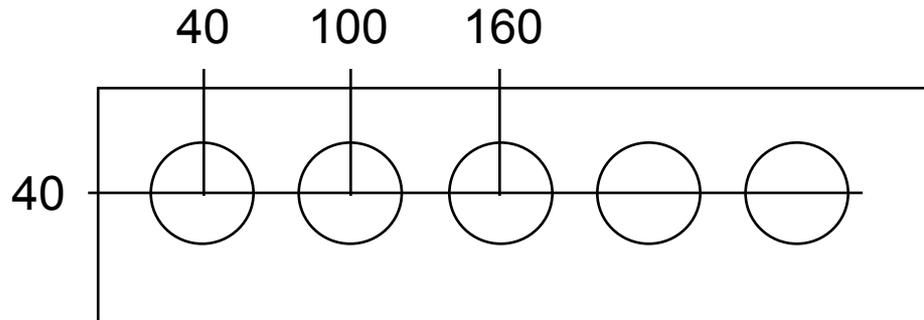
«6»: Используя одну процедуру, построить фигуру.



# Программирование на Python: графика

## 3. Циклы

# Использование циклов



? Что меняется?

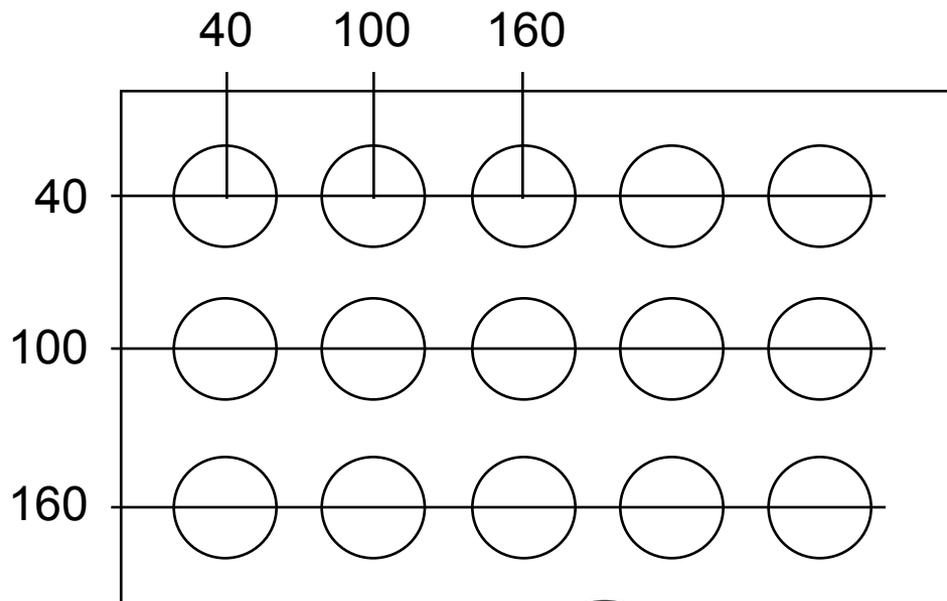
```
circle ( 40, 40, 20 )  
circle ( 100, 40, 20 )  
circle ( 160, 40, 20 )  
...
```

? Как меняется x?

```
x = 40  
for i in range(5):  
    circle(x, 40, 20)  
    x += 60
```

"сделай 5 раз"

# Использование циклов



1-й ряд:



Что меняется для 2-го ряда?

```
x = 40
for i in range(5):
    circle(x, 40, 20)
x += 60
```



Можно сделать это процедурой с параметром  $y$ !

# Использование циклов

```
from graph import *  
  
def row ( y ) :  
    x = 40  
    for i in range (5) :  
        circle (x, y, 20)  
        x += 60  
  
y = 40  
for k in range (3) :  
    row ( y )  
    y += 60  
run ()
```

процедура

ВЫЗОВ  
процедуры

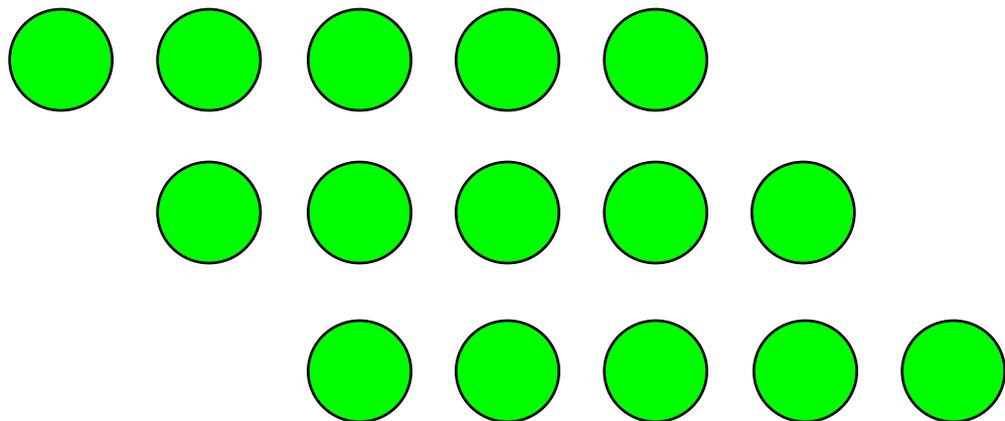
вниз на 60

# Задания

---

«3»: Ввести с клавиатуры число N и нарисовать N рядов по 5 кругов.

Пример (N = 3):

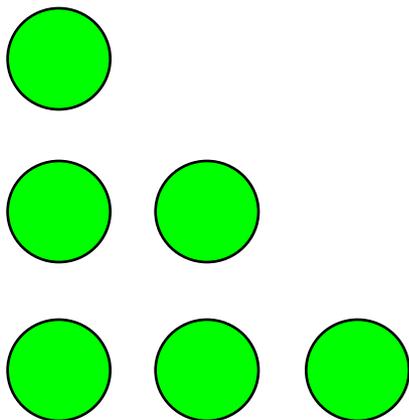


# Задания

---

«4»: Ввести с клавиатуры число  $N$  и нарисовать из кругов прямоугольный размер  $N$  на  $N$ .

**Пример** ( $N = 3$ ):

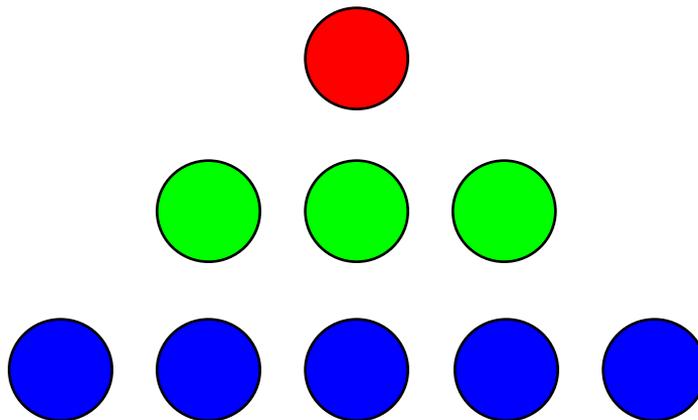


## Задания

---

«5»: Ввести с клавиатуры число  $N$  и нарисовать из кругов равнобедренный треугольник с высотой  $N$ . Каждый ряд должен быть покрашен в свой цвет.

Пример ( $N = 3$ ):

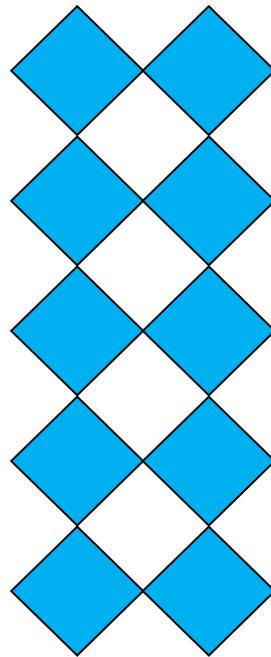


## Задания-2

---

«3»: Ввести с клавиатуры число  $N$  и нарисовать  $N$  вертикальных рядов по 5 ромбиков.

Пример ( $N = 2$ ):

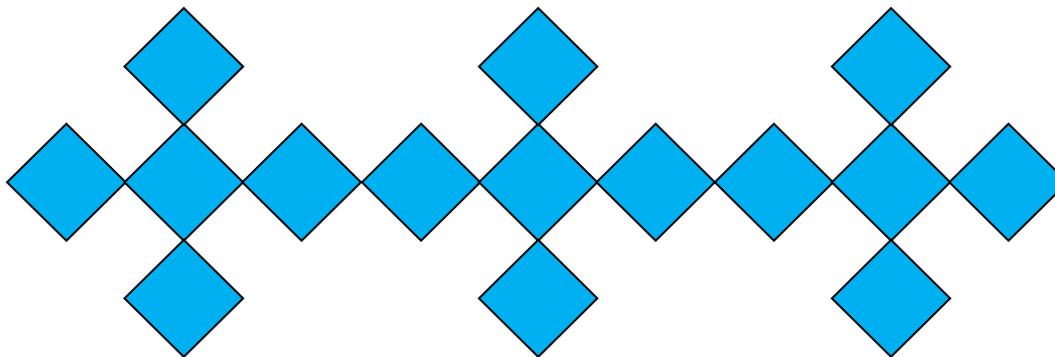


## Задания-2

---

«4»: Используя циклы и процедуры, нарисуйте узор. Число повторений рисунка  $N$  введите с клавиатуры.

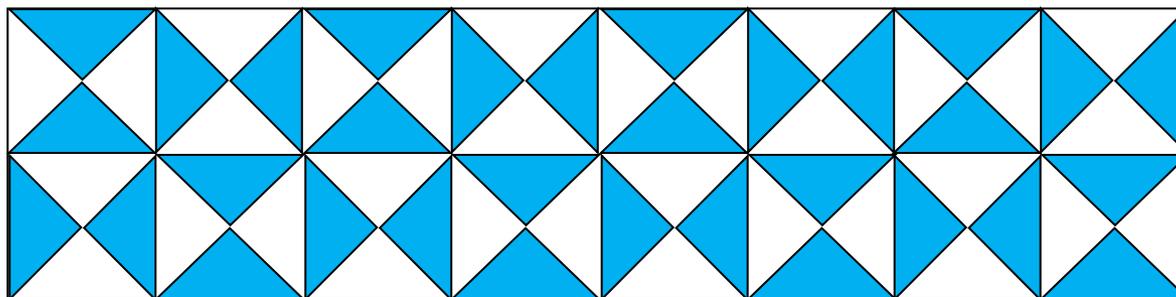
**Пример** ( $N = 3$ ):



## Задания-2

---

«5»: Используя циклы и процедуры, нарисуйте узор.

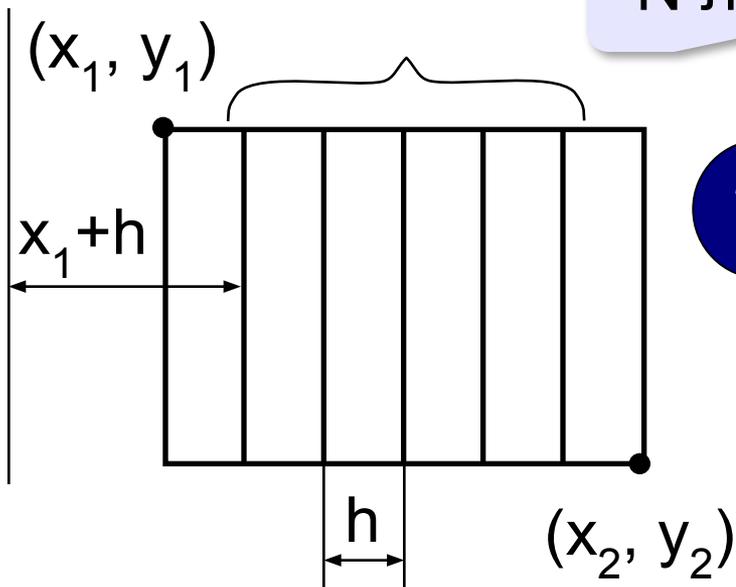


# Программирование на Python: графика

## 4. Штриховка

# Штриховка

N линий (N=5)



**?** Как найти  $h$ ?

$$h = \frac{x_2 - x_1}{N + 1}$$

В цикле менять  $x$ :

```
line ( x, y1, x, y2)
```

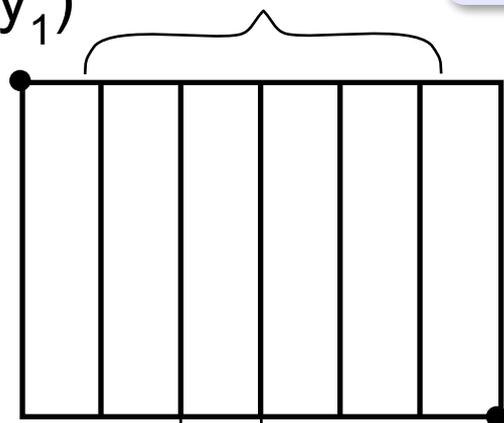
```
rectangle (x1, y1, x2, y2)
line ( x1+h, y1, x1+h, y2)
line ( x1+2*h, y1, x1+2*h, y2)
line ( x1+3*h, y1, x1+3*h, y2)
...
```

**x**                      **x**

# Штриховка

N линий (N=5)

$(x_1, y_1)$



меняется!

`line(x, y1, x, y2)`

?

Как меняется?

$x = ?$

$h$

$(x_2, y_2)$

для 1-й линии

```
x = x1 + h
```

```
for i in range(N):
```

```
    line(x, y1, x, y2)
```

```
    x += h
```

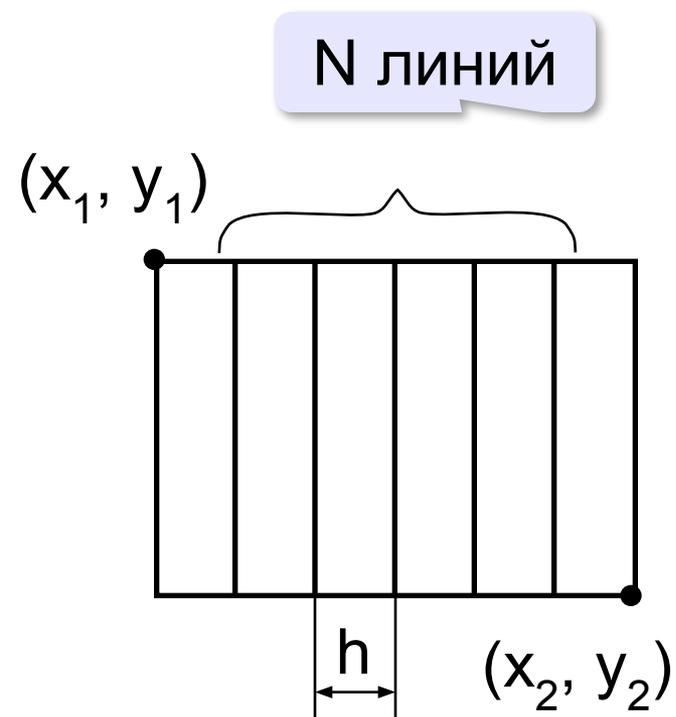
"сделай N раз"

для следующей  
линии

?

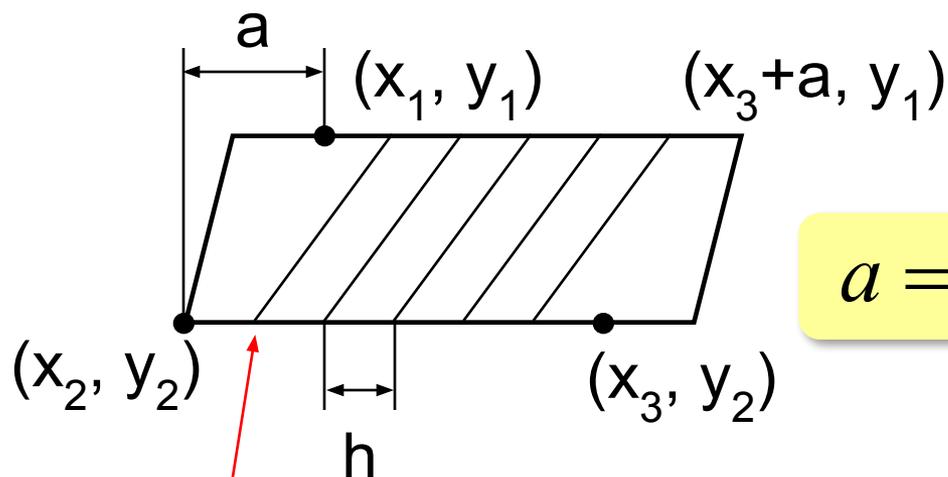
Что плохо?

# Штриховка



```
from graph import *  
x1 = 100; y1 = 100  
x2 = 300; y2 = 200  
N = 10  
rectangle(x1, y1, x2, y2)  
h = (x2-x1) / (N+1)  
x = x1 + h  
for i in range(N):  
    line(x, y1, x, y2)  
    x += h  
run()
```

# Сложная штриховка



Как найти  $a$  и  $h$ ?

$$a = x_1 - x_2$$

$$h = \frac{x_3 - x_2}{N + 1}$$

```
line ( x1+h,      y1,  x1+h-a,      y2 );
line ( x1+2*h,   y1,  x1+2*h-a,    y2 );
line ( x1+3*h,   y1,  x1+3*h-a,    y2 );
...
```

$x$

$x - a$



Как меняется  $x$ ?

Сначала:

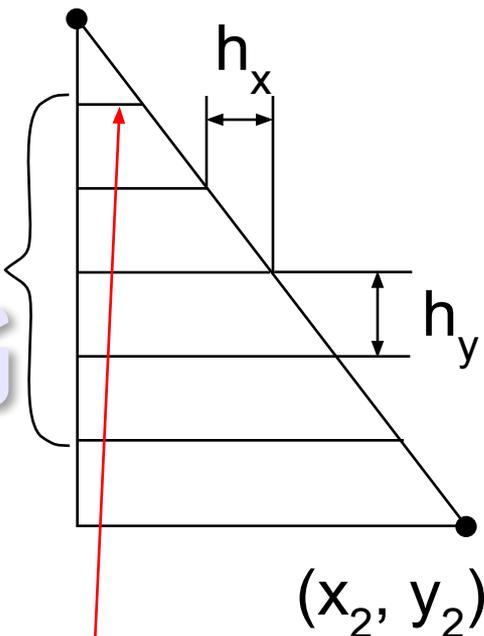
$$x = x1 + h$$

В цикле:

$$x += h$$

# Очень сложная штриховка

$(x_1, y_1)$



Как найти  $h_x$  и  $h_y$ ?

$$h_x = \frac{x_2 - x_1}{N + 1}$$

$$h_y = \frac{y_2 - y_1}{N + 1}$$

Сначала:

`x = x1+hx`

`y = y1+hy`

В цикле:

`x += hx`

`y += hy`

```
line ( x1, y1+hy, x1+hx, y1+hy) ;
line ( x1, y1+2*hy, x1+2*hx, y1+2*hy) ;
line ( x1, y1+3*hy, x1+3*hx, y1+3*hy) ;
...
```

y

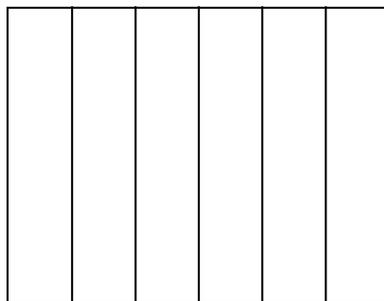
x

y

## Задания

---

«3»: Ввести с клавиатуры количество линий,  
построить фигуру и выполнить штриховку:



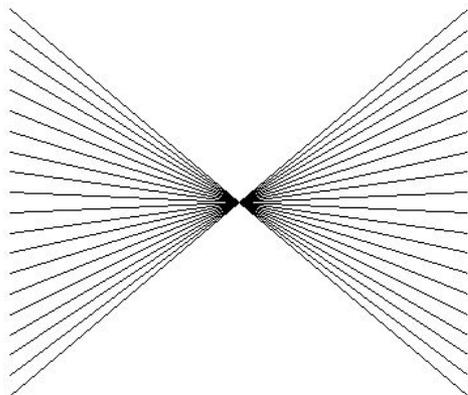
«4»: Ввести с клавиатуры количество линий,  
построить фигуру и выполнить штриховку:



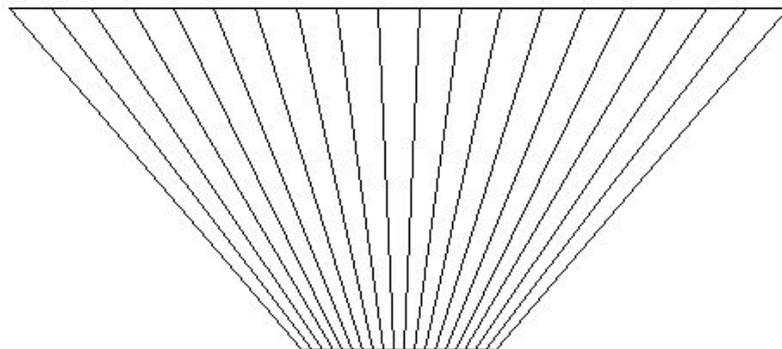
## Задания

---

«5»: Ввести с клавиатуры количество линий и построить фигуру:



«6»: Ввести с клавиатуры количество линий и построить фигуру:

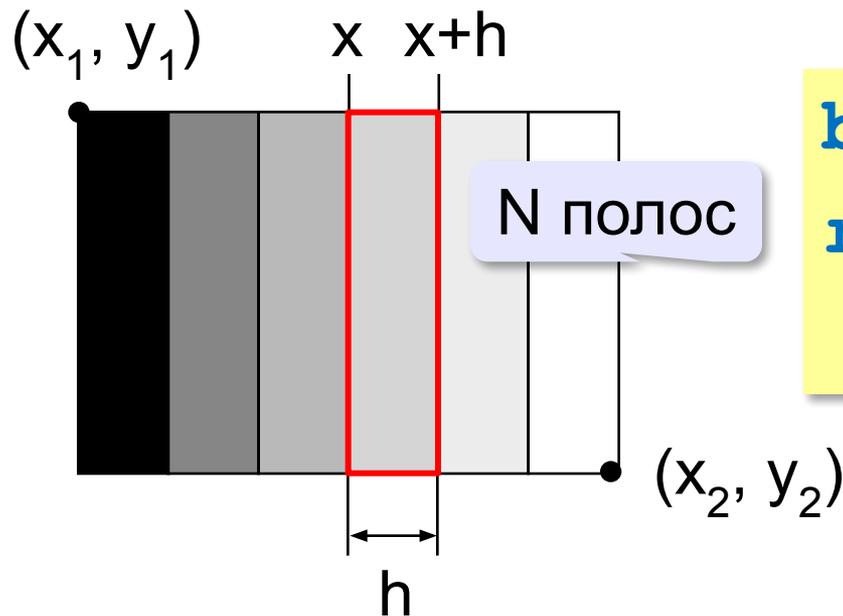


# Программирование на Python: графика

## 5. Закрашивание областей

# Заливка разными цветами

серый: R=G=B



```
brushColor(c, c, c)
rectangle(x, y1,
          x+h, y2)
```

Шаг изменения цвета:

$hc = 255 // N$

```
x = x1; c = 0
for i in range(N):
    brushColor(c, c, c)
    rectangle(x, y1, x+h, y2)
    x += h; c += hc
```

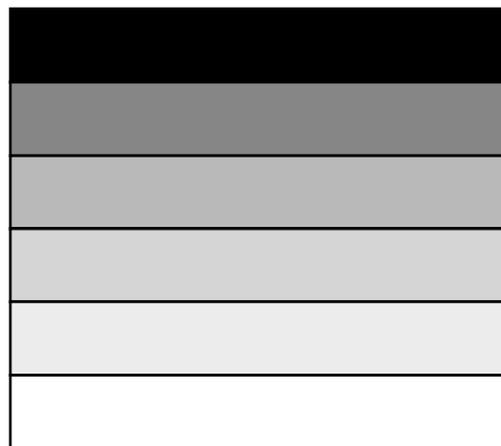
# Задания

---

«3»: Ввести с клавиатуры число полос и построить фигуру, залив все области разным цветом.



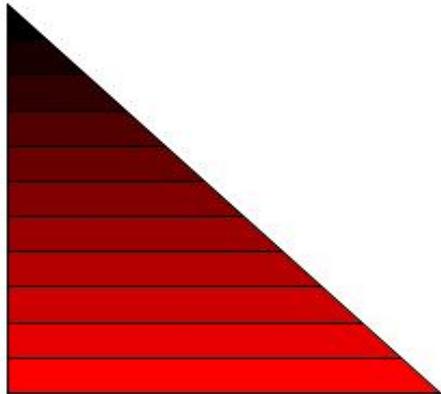
«4»: Ввести с клавиатуры число полос и построить фигуру, залив все области разным цветом.



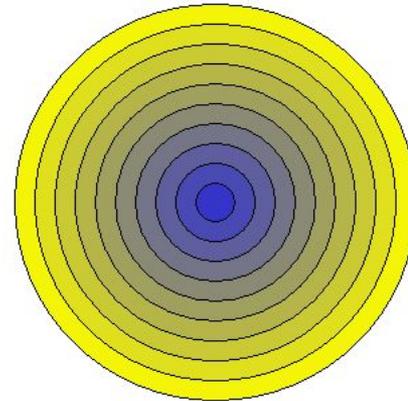
# Задания

---

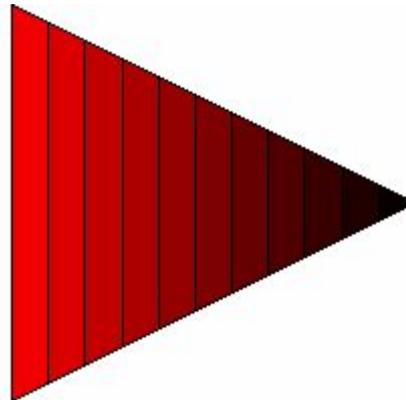
«5»: Ввести с клавиатуры число полос и построить фигуру, залив все области разным цветом.



или



«6»: Ввести с клавиатуры число полос и построить фигуру, залив все области разным цветом.



# Программирование на Python: графика

## 6. Построение графиков функций

# Графики функций

---

**Задача:** построить график функции  $y = x^2$  на отрезке от -2 до 2.

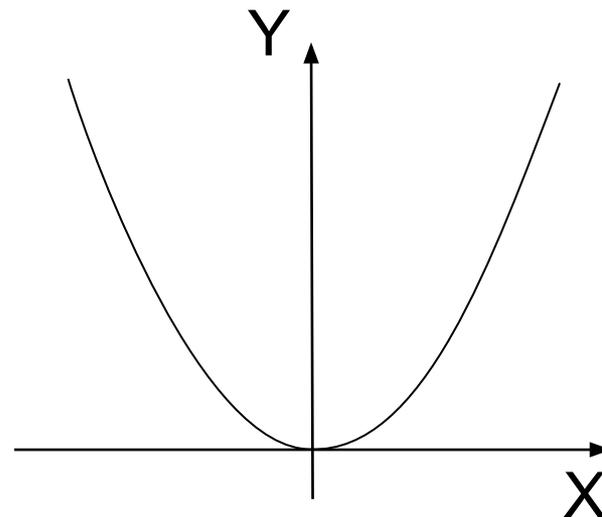
**Анализ:**

максимальное значение

$$y_{\max} = 4 \quad \text{при} \quad x = \pm 2$$

минимальное значение

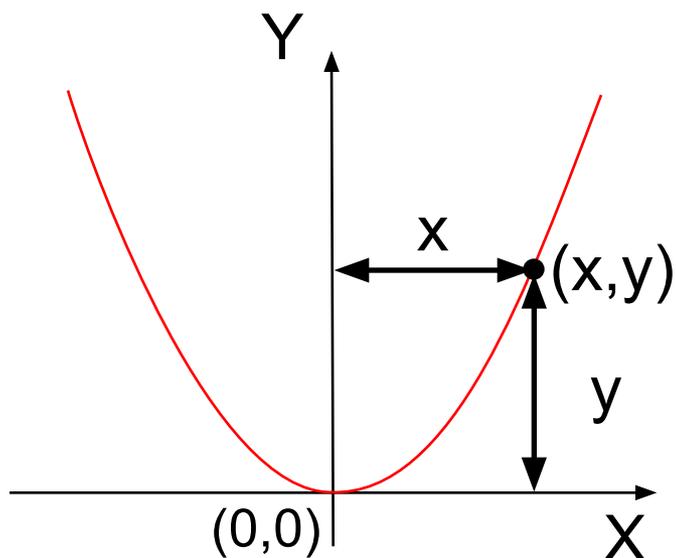
$$y_{\min} = 0 \quad \text{при} \quad x = 0$$



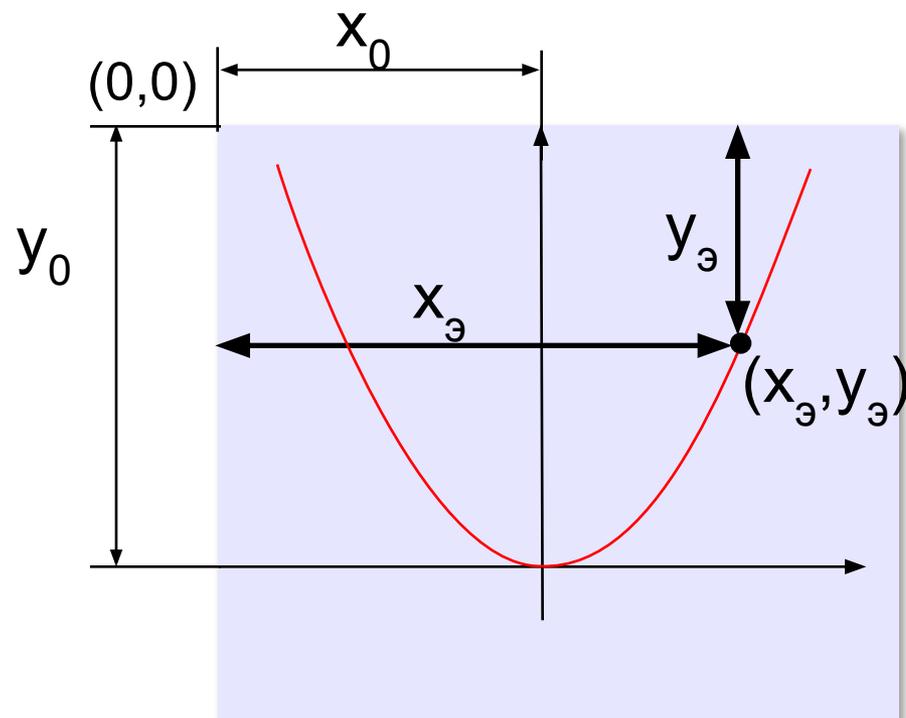
**Проблема:** функция задана в математической системе координат, строить надо на экране, указывая координаты в пикселях.

# Преобразование координат

Математическая  
система координат



Экранная система  
координат (пиксели)

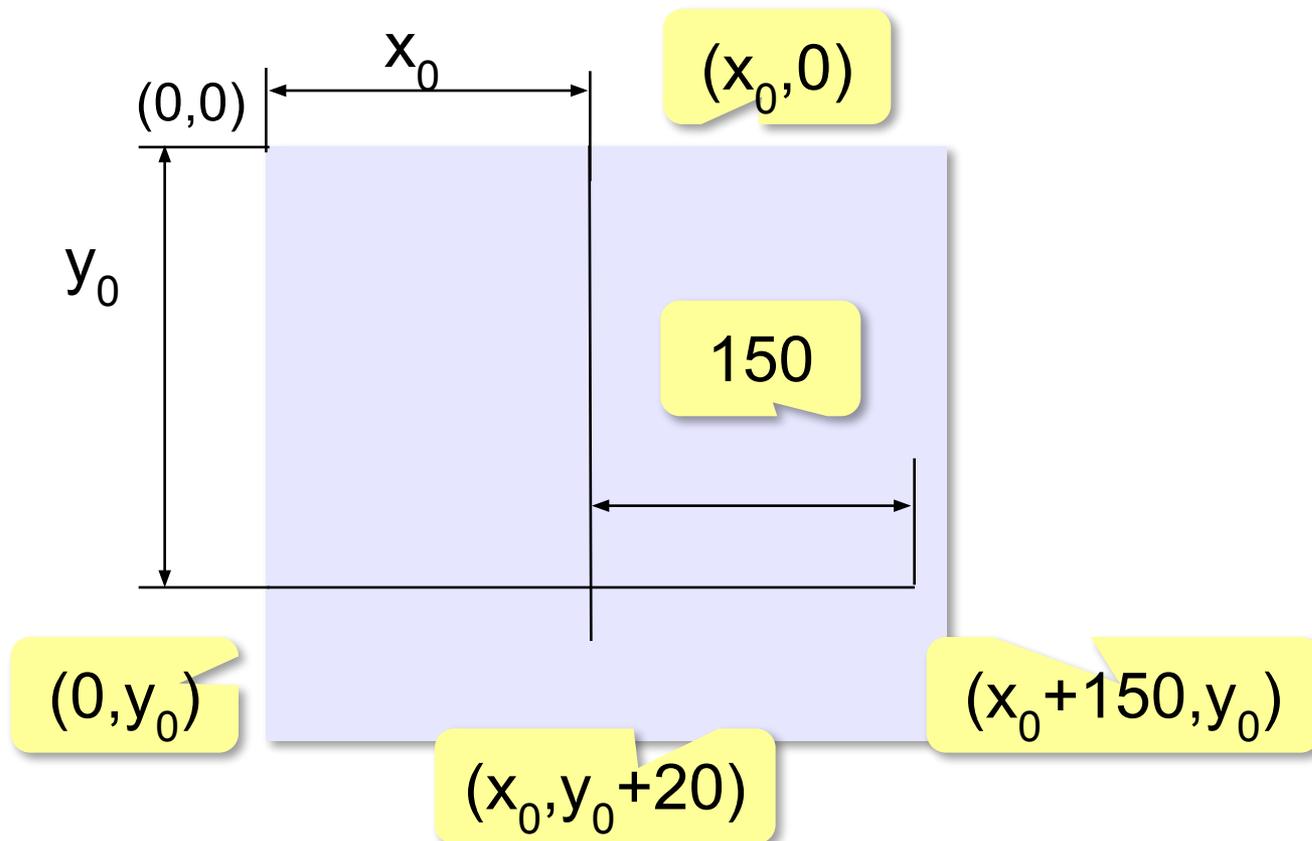


$k$  – масштаб (длина  
изображения единичного  
отрезка на экране)

$$x_{\text{э}} =$$

$$y_{\text{э}} =$$

# Оси координат



```
line(0, y0, x0+150, y0)
```

```
line(x0, 0, x0, y0+20)
```

# Рисуем оси координат

---

```
from graph import *
x0 = 150 # начало координат
y0 = 250
k = 50    # масштаб
xmin = -2; xmax = 2 # пределы по x
line(0, y0, x0+150, y0)
line(x0, 0, x0, y0+20)
...
```

# Строим по точкам

```
...  
x = xmin    # начальное значение x  
h = 0.02    # шаг изменения x  
penColor("red")  
while x <= xmax:  
    y = x*x  # функция  
    xe = x0 + k*x  
    ye = y0 - k*y  
    point(xe, ye) # точка на экране  
    x += h      # к следующей точке  
run()
```

экранные координаты  
(в пикселях)

# Соединяем точки линиями

**Идея:** сначала создаём в памяти массив точек, затем соединяем точки линиями (**polygon**)

```
points = [] # пустой массив
```

```
while x <= xmax:
```

```
    y = x*x
```

```
    xe = x0 + k*x
```

```
    ye = y0 - k*y
```

```
    points.append( (xe, ye) )
```

```
    x += h
```

добавляем точку  
в массив

```
penColor("red")
```

```
polyline(points) # рисуем линию!
```

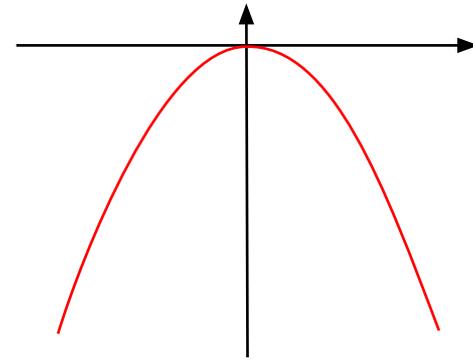
# Задания

---

«3»: Построить график функции

$$y = -x^2$$

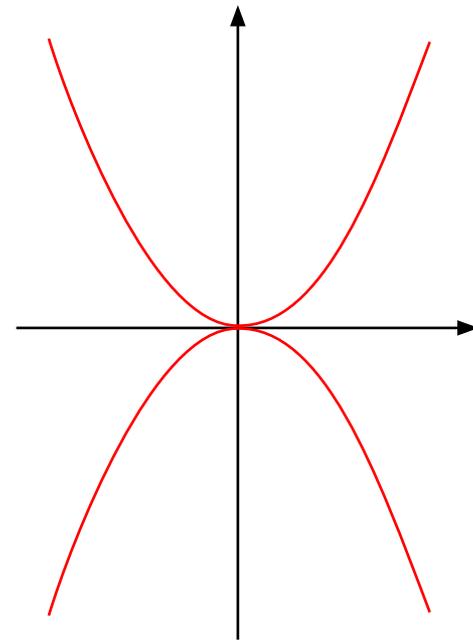
на отрезке  $[-2, 2]$ .



«4»: Построить графики функций

$$y = x^2 \text{ и } y = -x^2$$

на отрезке  $[-2, 2]$ .



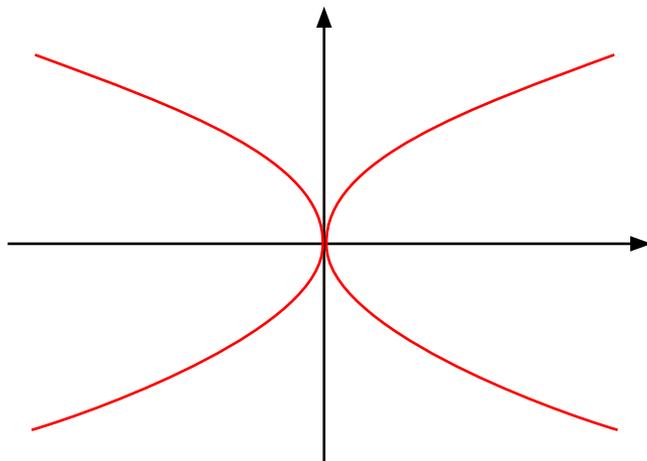
# Задания

---

«5»: Построить графики функций

$$x = y^2 \text{ и } x = -y^2$$

на отрезке  $[-2, 2]$ .



# Программирование на Python: графика

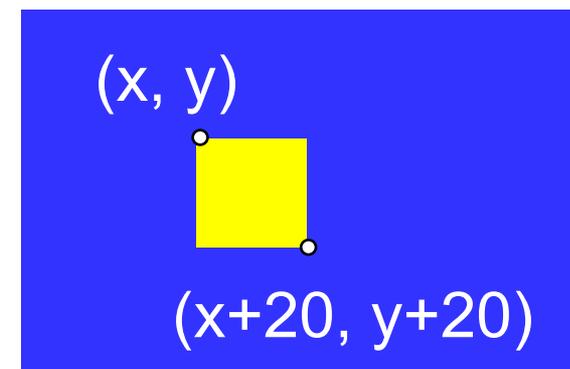
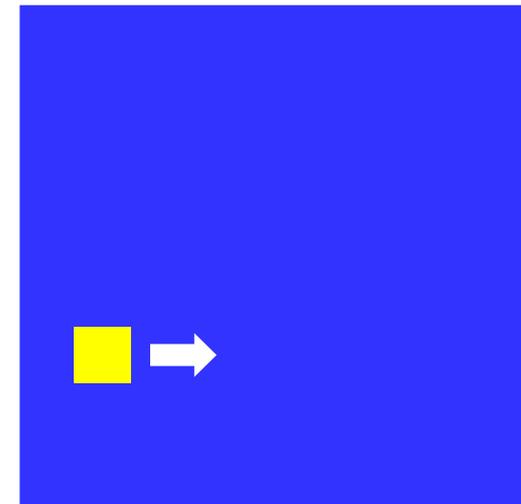
## 7. Анимация

# Анимация

**Анимация (англ. *animation*)** – оживление изображения на экране.

**Задача:** внутри синего квадрата 400 на 400 пикселей слева направо движется желтый квадрат 20 на 20 пикселей. Программа останавливается, если нажата клавиша *Esc* или квадрат дошел до границы синей области.

**Привязка:** состояние объекта задается координатами  $(x, y)$



# Принцип анимации

---

1. рисуем объект в точке (x,y)
2. задержка на несколько миллисекунд
3. стираем объект
4. изменяем координаты (x,y)
5. переходим к шагу 1



В Python все фигуры, из которых состоит рисунок, – **объекты** (умеют перерисовывать себя сами)!

объект

смещения по осям

```
moveObjectBy(obj, dx, dy)
```

# Начальная картинка

```
from graph import *  
brushColor("blue")  
rectangle(0, 0, 400, 400)  
  
x = 100  
y = 100  
  
penColor("yellow")  
brushColor("yellow")  
obj = rectangle(x, y, x+20, y+20)  
run()
```

СИНИЙ  
квадрат

начальные  
координаты

жёлтый  
квадрат

# Движение

```
def update():  
    moveObjectBy(obj, 5, 0)  
    if xCoord(obj) >= 380: # если вышел  
        close()           # за границу  
onTimer(update, 50)
```

x-координата

вызывать `update`  
каждые 50 мс

# Выход по Escape

**Событие (англ. *event*)** – изменение состояния какого-то объекта в программе (нажатие на клавишу, щелчок мышью, перемещение или изменение размеров окна и т.п.).

обработчик события

код клавиши  
Esc = 27

```
def keyPressed(event) :  
    if event.keycode == VK_ESCAPE :  
        close() # закрыть окно  
onKey (keyPressed)
```

вызывать при  
нажатии любой  
клавиши

установка  
обработчика события

# Полная программа

```
from graph import *
def update():
    ...
def keyPressed(event):
    ...
brushColor("blue")
rectangle(0, 0, 400, 400)
x = 100
y = 100
penColor("yellow")
brushColor("yellow")
obj = rectangle(x, y, x+20, y+20)
onKey(keyPressed)
onTimer(update, 50)
run()
```

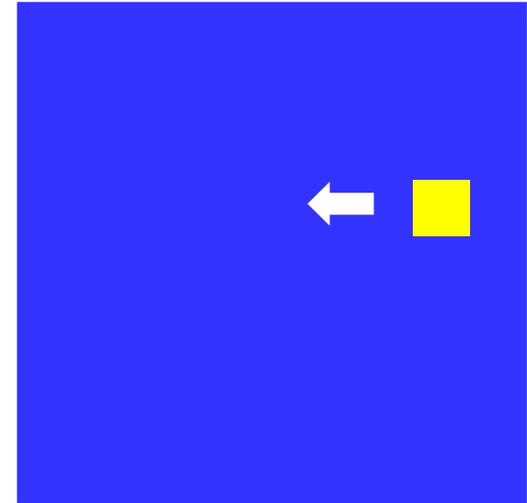
процедуры

обработка  
СОБЫТИЙ

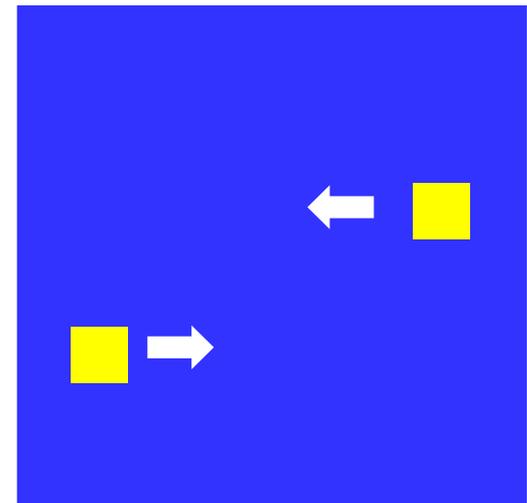
# Задания

---

«3»: Квадрат движется справа налево:



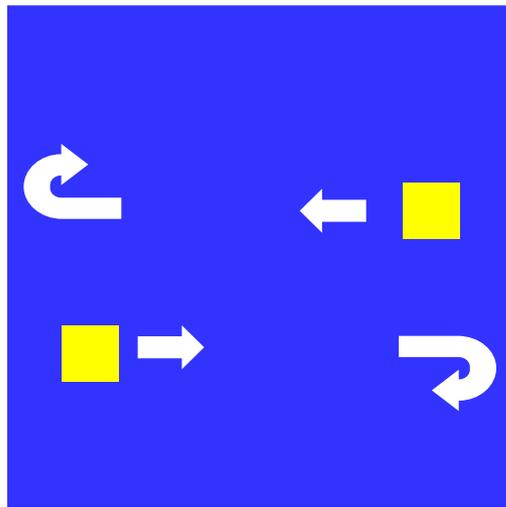
«4»: Два квадрата движутся в противоположных направлениях:



# Задания

---

«5»: Два квадрата двигаются в противоположных направлениях и отталкиваются от стенок синего квадрата:



# Управление клавишами

**Задача:** жёлтый квадрат внутри синего квадрата управляется клавишами-стрелками. Коды клавиш:

влево – 37          вверх – 38          Esc – 27  
вправо – 39        вниз – 40

VK\_ESCAPE

**Проблема:** как изменять направление движения?

**Обработчик события:**

```
def keyPressed(event) :  
    if event.keycode == VK_LEFT:  
        moveObjectBy(obj, -5, 0)  
    elif event.keycode == VK_RIGHT:  
        moveObjectBy(obj, 5, 0)  
    ... # дальше – сами...  
onKey(keyPressed) # установить обработчик
```

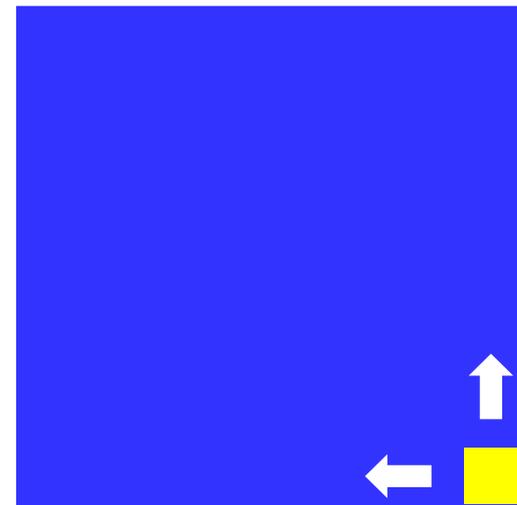
=37

=39

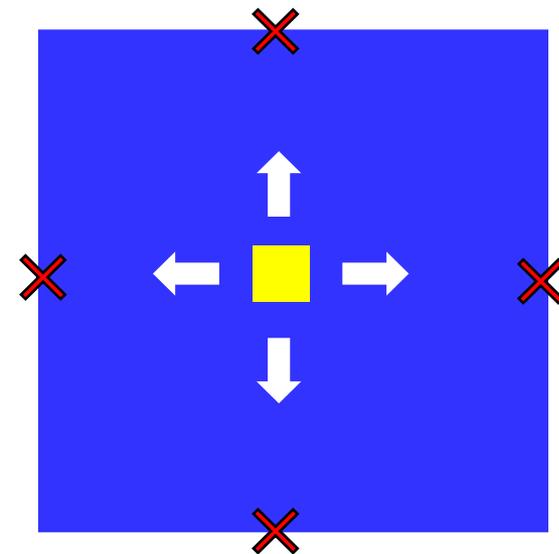
VK\_UP = 38  
VK\_DOWN = 40

# Задания

«3»: Квадрат в самом начале стоит в правом нижнем углу, и двигается при нажатии стрелок только вверх или влево:

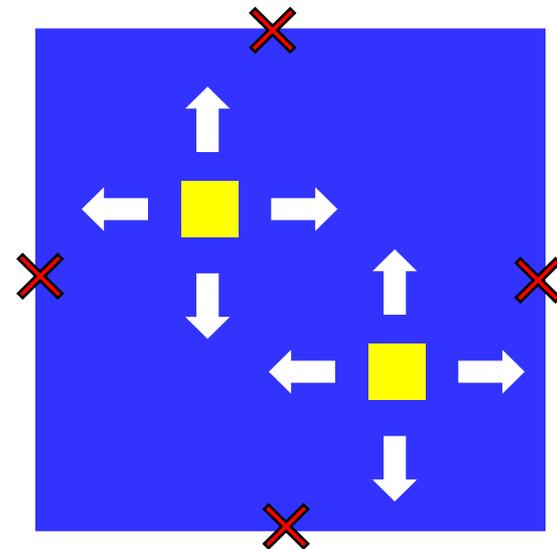


«4»: Квадрат двигается при нажатии стрелок, однако не может выйти за границы синего квадрата:



# Задания

«5»: Два квадрата, один управляется стрелками, второй – любыми другими клавишами. Оба не могут выйти за границы синего поля.



# Управление по требованию

**Задача:** жёлтый квадрат постоянно движется и меняет направление движения при нажатии клавиш-стрелок. При нажатии на пробел останавливается.

**Проблема:** как изменять направление движения?

**Решение:**

```
def update():  
    ...  
    moveObjectBy(obj, dx, dy)  
    ...  
onTimer(update, 50)
```



Что меняем?



Здесь должны быть переменные!

# Как «поймать» нажатие клавиши?

```
def keyPressed(event) :  
    global dx, dy  
    if event.keycode == VK_LEFT :  
        dx = ?  
        dy = ?  
    ...  
onKey (keyPressed)
```

Это глобальные  
переменные!



Как для остальных клавиш?

Пробел: **VK\_SPACE**

# Полная программа

```
from graph import *
def update():
    ...
def keyPressed(event):
    ...
# рисуем синий квадрат
x = 100; y = 100
dx = 0; dy = 0
penColor("yellow")
brushColor("yellow")
obj = rectangle(x, y, x+20, y+20)
onKey(keyPressed)
onTimer(update, 50)
run()
```

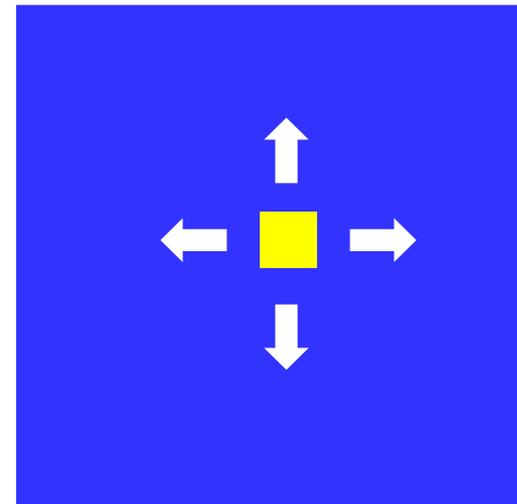
процедуры

глобальные  
переменные

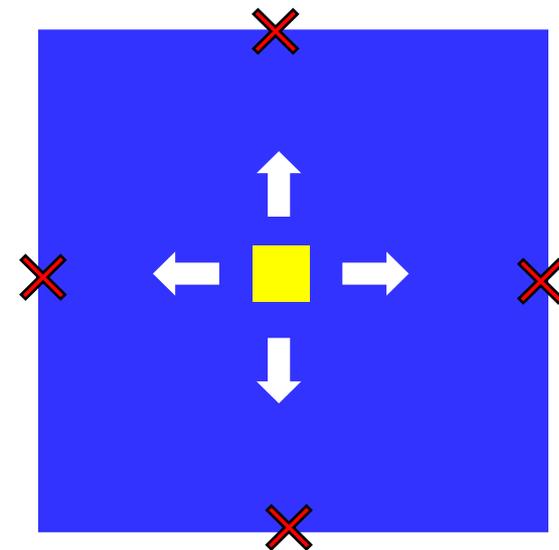
обработка  
событий

# Задания

«3»: Собрать и запустить программу.



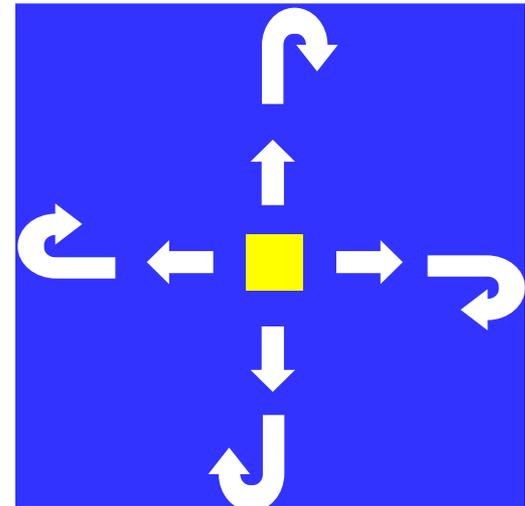
«4»: Квадрат не может выйти за границы синего квадрата, сразу останавливается при столкновении со стенкой.



# Задания

---

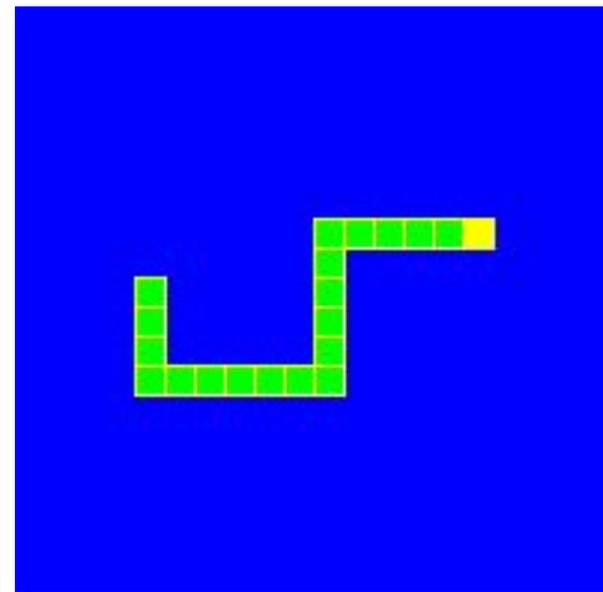
«5»: Квадрат отталкивается от стенок.



«6»: Квадрат может ходить по диагоналям (используйте ещё 4 клавиши) и отталкивается от стенок.

# «Змейка»

**Задача:** змейка состоит из головы и нескольких секций тела, постоянно движется и меняет направление движения при нажатии клавиш-стрелок. При нажатии на пробел останавливается.

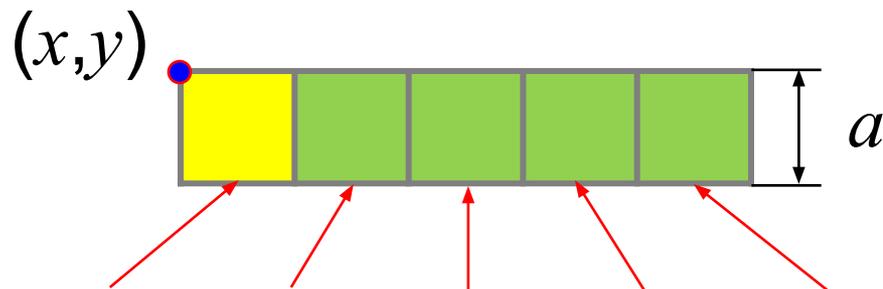


## Проблемы:

- 1) как хранить данные о змейке?
- 2) как двигать её в нужную сторону?

# Как хранить змейку?

Змейка = массив из звеньев-квадратов



snake = [obj0, obj1, obj2, obj3, obj4]

```
snake = []
```

```
penColor("yellow")
```

```
brushColor("yellow")
```

```
for i in range(N):
```

```
    obj = rectangle(x+i*a, y, x+i*a+a, y+a)
```

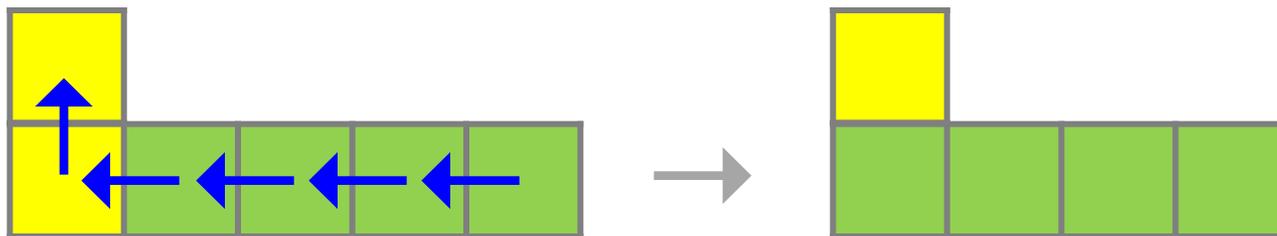
```
    snake.append(obj)
```

```
    brushColor("green")
```

цвет границы и заливки  
для головы

остальные зелёные

# Как двигать змейку?



Каждое следующее звено перемещается на место предыдущего!

Для  $k$ -ого звена:

```
newCoord = coords (snake [k-1])  
moveObjectTo (snake [k] , newCoord [0] ,  
              newCoord [1])
```

координаты  
предыдущего  
звена



В каком порядке перебирать звенья?

с последнего!

# Как двигать змейку?

Вся змейка:

```
def moveSnake (xNew, yNew) :  
    global x, y  
    for k in range (len (snake) - 1, 0, -1) :  
        newCoord = coords (snake [k - 1])  
        moveTo (snake [k], newCoord [0],  
                newCoord [1])  
    moveTo (snake [0], xNew, yNew)  
    x = xNew  
    y = yNew
```

перебор с  
последнего,  
кроме головы

двигаем голову

# Как двигать змейку?



Шаг перемещения всегда равен  $a$ !

Удобно так:  $dx, dy = -1, 0$  или  $1, 0$

```
def keyPressed(event) :  
    global dx, dy  
    if event.keycode == VK_LEFT :  
        dx = -1  
        dy = 0  
    ...  
def update () :  
    if dx or dy :  
        moveSnake ( x + dx*a , y + dy*a )
```

если оба нули,  
двигать не нужно!

# Полная программа

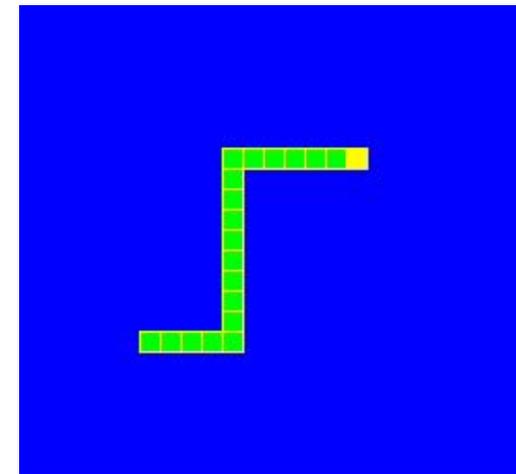
```
from graph import *
def moveSnake(xNew, yNew):
    ...
def update():
    ...
def keyPressed(event):
    ...

# рисуем синий квадрат
x = 100; y = 100 # координаты головы
dx = 0; dy = 0   # в начале стоит на месте
a = 10; N = 20   # размер и количество звеньев
# рисуем змейку в начальном положении
onKey(keyPressed)
onTimer(update, 50)
run()
```

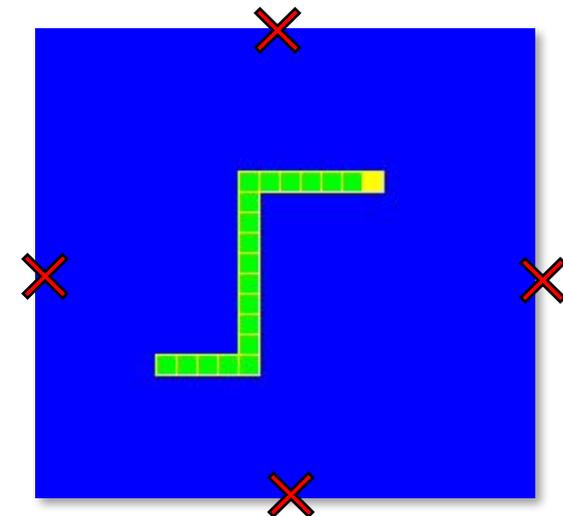
# Задания

---

«3»: Собрать и запустить программу.



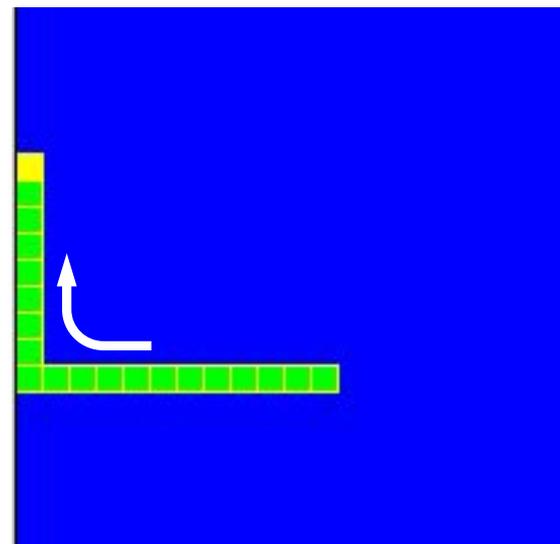
«4»: Змейка не может выйти за пределы синего квадрата (останавливается у стенки).



## Задания

---

«5»: Змейка при столкновении с границей поля начинает ползти вдоль этой границы.



«6»: Если змейка пересекает сама себя, игра заканчивается.

# Случайные числа

## Случайно...

- встретить друга на улице
- разбить тарелку
- найти 10 рублей
- выиграть в лотерею

## Случайный выбор:

- жеребьевка на соревнованиях
- выигравшие номера в лотерее

## Как получить случайность?



# Случайные числа на компьютере

## Электронный генератор



- нужно специальное устройство
- нельзя воспроизвести результаты

**Псевдослучайные числа** – обладают свойствами случайных чисел, но каждое следующее число вычисляется по заданной формуле.

## Метод середины квадрата (Дж. фон Нейман)

зерно

564321

в квадрате

- малый период  
(последовательность повторяется через  $10^6$  чисел)

318458191041

209938992481

# Линейный конгруэнтный генератор

$X := (a * X + b) \% c$  # интервал от 0 до  $c-1$

$X := (X+7) \% 10$  # интервал от 0 до 9

$X := 0 \rightarrow 7 \rightarrow 4 \rightarrow 1 \rightarrow 8 \rightarrow 5 \rightarrow 2$

$2 \rightarrow 9 \rightarrow 6 \rightarrow 3 \rightarrow 0$

зерно

заикливание



Важен правильный выбор параметров  $a$ ,  $b$  и  $c$ !

Компилятор GCC:

$a = 1103515245$

$b = 12345$

$c = 2^{31}$

# Генератор случайных чисел

---

```
import random
```

англ. *random* – случайный

Целые числа на отрезке [a,b]:

```
X = random.randint(1, 6) # псевдосл. число  
Y = random.randint(1, 6) # уже другое число!
```

Генератор на [0,1):

```
X = random.random() # псевдосл. число  
Y = random.random() # уже другое число!
```

# Генератор случайных чисел

```
from random import *
```

подключить все!

англ. *random* – случайный

Целые числа на отрезке [a,b]:

```
X = randint(10, 60) # псевдослучайное число  
Y = randint(10, 60) # это уже другое число!
```

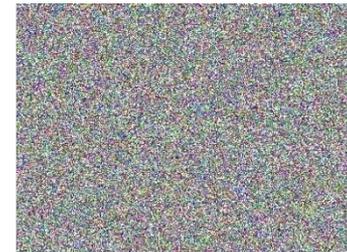
Генератор на [0,1):

```
X = random() ; # псевдослучайное число  
Y = random()   # это уже другое число!
```

# Случайные числа

---

**Задача:** заполнить прямоугольник 400 на 300 пикселей равномерно точками случайного цвета



**Как получить случайные координаты точки?**

```
x = randint(0, 399)
y = randint(0, 299)
```

**Как добиться равномерности?**

обеспечивается автоматически при использовании функции `randint`

# Точка случайного цвета из списка

```
from random import choice
```

все варианты

```
...
```

```
colors = ["red", "green", "blue",  
          "black", "#FFFF00"]
```

```
col = choice(colors)
```

случайный  
выбор

```
penColor( col )
```

```
point(x, y)
```

ПОСТАВИТЬ ТОЧКУ

# Точка случайного цвета (RGB)

Цвет в формате RGB:

"yellow"

```
penColor ( 255 , 255 , 0 )
```

**R**(red)  
0..255

**G**(green)  
0..255

**B**(blue)  
0..255

```
r = randint (0 , 255)
```

```
g = randint (0 , 255)
```

```
b = randint (0 , 255)
```

```
penColor ( r , g , b )
```

```
point (x , y)
```

# Вся программа

```
from graph import *
from random import choice
colors = ["red", "green", "blue",
          "black", "#FFFF00"]

def newPoint():
    x = randint(0, 399)
    y = randint(0, 299)
    col = choice( colors )
    penColor( col )
    point(x, y)

def keyPressed(event):
    if event.keycode == VK_ESCAPE:
        close()

onKey( keyPressed )
onTimer( newPoint, 10 )
run()
```

НОВАЯ ТОЧКА  
через 10 мс

ВЫХОД ПО  
Escape

установка  
обработчиков  
СОБЫТИЙ

# Задания

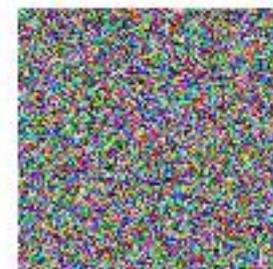
---

«3»: Заполнить квадрат точками случайного цвета. размер квадрата ввести с клавиатуры:

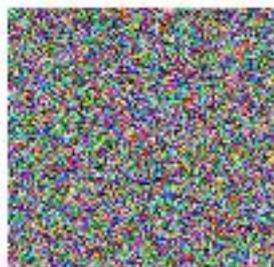
Пример:

Введите размер квадрата:

150



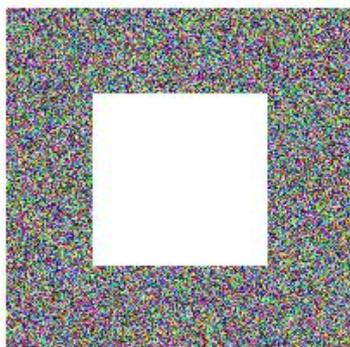
«4»: Заполнить две области точками случайного цвета:



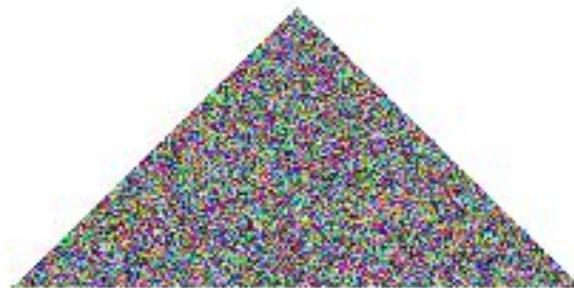
# Задания

---

«5»: Заполнить область точками случайного цвета:



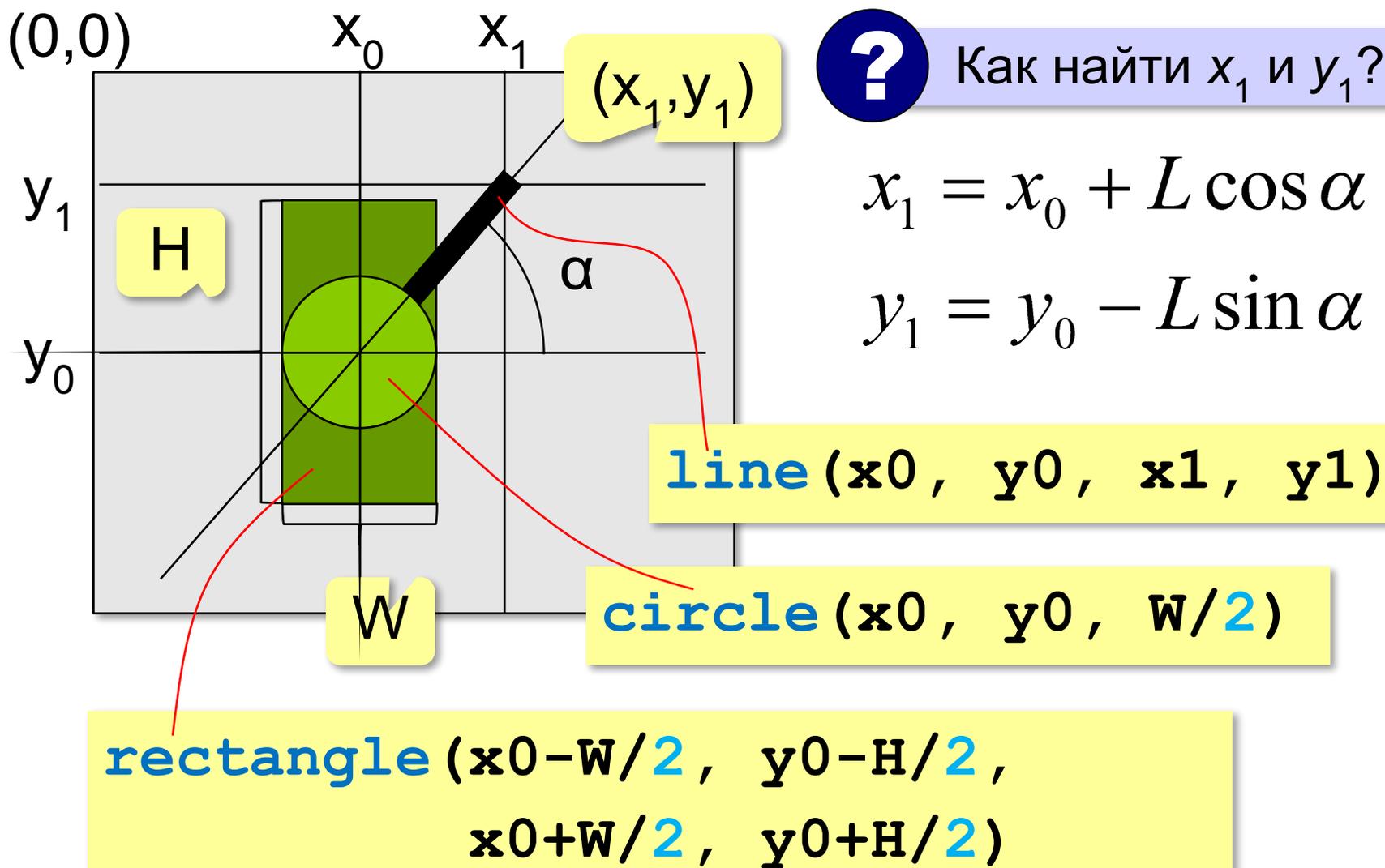
ИЛИ



# Программирование на Python: графика

## 8. Игры

# Танк с вращающейся пушкой



# Начальная картинка

```
from graph import *
import math
H = 60; W = 30; L = 40 # размеры танка
x0 = 200; y0 = 400; angle = 90 # пушка
brushColor("#6b8e23")
rectangle(x0-W/2, y0-H/2, x0+W/2, y0+H/2)
a = angle*math.pi/180 # в радианы
x1 = x0+L*math.cos(a)
y1 = y0-L*math.sin(a)
penSize(5)
line(x0, y0, x1, y1)
penSize(1)
brushColor("#556b2f")
circle(x0, y0, W/2)
run()
```

корпус

СТВОЛ

башня

# Анимация поворота пушки

```
def keyPressed(event) :  
    if event.keycode == VK_LEFT:  
        drawGun(angle+5) # влево на 5 градусов  
    elif event.keycode == VK_RIGHT:  
        drawGun(angle-5) # вправо на 5 градусов  
    elif event.keycode == VK_ESCAPE:  
        close()  
  
    ...  
onKey(keyPressed)
```



Нужно написать процедуру `drawGun!`

# Рисование и вращение пушки

**Идея:** в первый раз рисуем, потом просто меняем координаты.

сначала **None** –  
«ПУСТО»

```
def drawGun (angleNew) :  
    global angle, gun # глобальные переменные  
    angle = angleNew # запомнить новый угол  
    aRad = angle*math.pi/180 # в радианы  
    x1 = x0 + L*math.cos (aRad)  
    y1 = y0 - L*math.sin (aRad)  
    if gun == None: # если в первый раз...  
        gun = line (x0, y0, x1, y1)  
    else: # если пушка уже нарисована  
        changeCoord (gun, [ (x0, y0), (x1, y1) ] )
```

запомнить  
«адрес» линии

массив новых  
координат

# Рисование и вращение пушки

## Как это работает:

```
gun = None # еще не рисовали пушку
drawGun(angle) # рисуем в первый раз
└─ gun = line(x0, y0, x1, y1)
   # теперь gun содержит адрес линии
...
def keyPressed(event):
└─ drawGun(angle+5) # вращаем
   └─ changeCoord(gun, [(x0, y0), (x1, y1)])
      # просто меняем координаты
```

# Полная программа

```
from graph import *
import math
def keyPressed(event):
    ...
def drawGun(angleNew):
    ...
H = 60; W = 30; L = 40
x0 = 200; y0 = 400; angle = 90
gun = None
brushColor("#6b8e23")
rectangle(x0-W/2, y0-H/2, x0+W/2, y0+H/2)
penSize(5)
drawGun(angle)
penSize(1)
brushColor("#556b2f")
circle(x0, y0, W/2)
onKey(keyPressed)
run()
```

процедуры

начальные  
значения

корпус

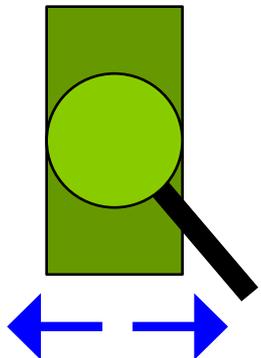
пушка

башня

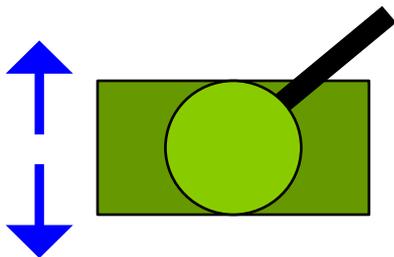
## Задания

---

«3»: Сделать танк с управляемой пушкой, развёрнутый в обратную сторону:



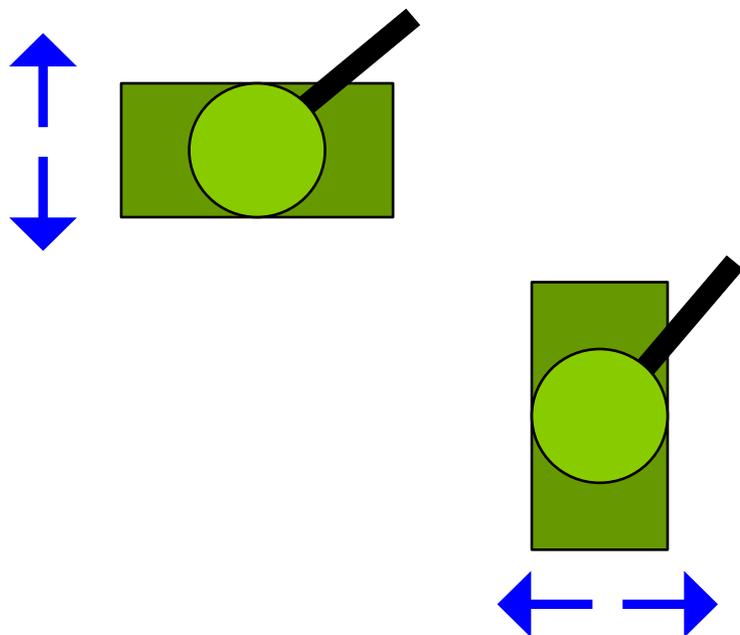
«4»: Сделать танк с управляемой пушкой, развёрнутый боком. Управление – клавишами "вверх" и "вниз":



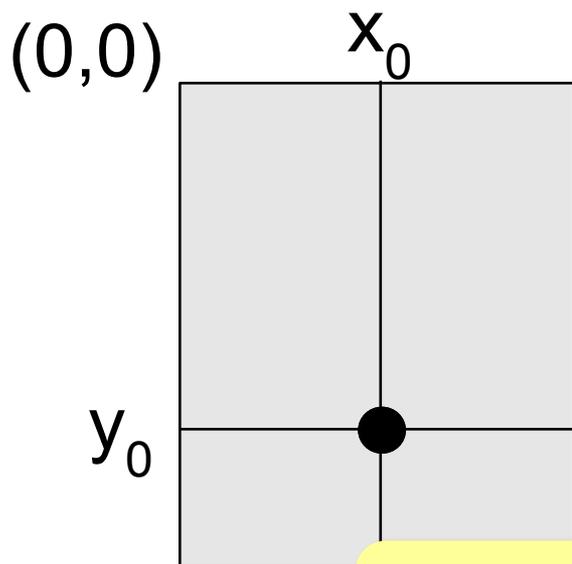
# Задания

---

«5»: Сделать два танка, у одного пушка управляется клавишами "влево" и "вправо", у второго – клавишами "вверх" и "вниз".



# Стрельба из пушки



нажали  
пробел"

## Создание объекта:

```
x0 = 200; y0 = 400  
r = 3 # радиус снаряда  
brushColor("black")  
bullet = circle(x0, y0, r)
```

запомнили  
код объекта

## Движение:

```
moveObjectBy(bullet, 0, -5)
```

по оси X

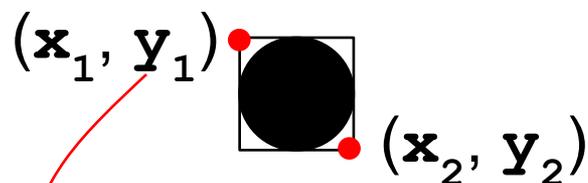
по оси Y

## Остановка при выходе за границу окна

1. определить y-координату объекта
2. если она меньше 0, то
  - остановить движение
  - вернуть снаряд в исходное положение

координаты объекта:

$(x_1, y_1, x_2, y_2)$



```
y = coords(bullet)[1]
```

```
if y < 0:
```

```
    isFlying = False
```

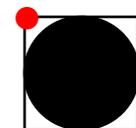
```
    moveTo(bullet, x0-r, y0-r)
```

ЛЕВЫЙ  
ВЕРХНИЙ УГОЛ

$(x_0-r, y_0-r)$

`isFlying` – логическая переменная

`True` – снаряд летит, `False` – не летит.



# Как организовать анимацию?

вызывается  
каждые 30 мс

```
def update():
    global isFlying, bullet
    if isFlying: # если летит...
        y = coords(bullet)[1]
        if y < 0: # если улетел...
            isFlying = False
            moveTo(bullet, x0-r, y0-r)
        else: # летит дальше...
            moveObjectBy(bullet, 0, -5)
    ...
onTimer(update, 30)
```

# Как запустить движение?

вызывается при  
нажатии клавиши

```
def keyPressed(event) :  
    global isFlying  
    if event.keycode == VK_SPACE:  
        isFlying = True # полетели!  
    elif event.keycode == VK_ESCAPE:  
        close() # закончить работу  
    ...  
onKey(keyPressed)
```

# Полная программа

```
from graph import *
def update():
    ...
def keyPressed(event):
    ...
x0 = 200; y0 = 400; r = 3
brushColor("black")
bullet = circle(x0, y0, r)
isFlying = False

onKey(keyPressed)
onTimer(update, 30)

run()
```

процедуры

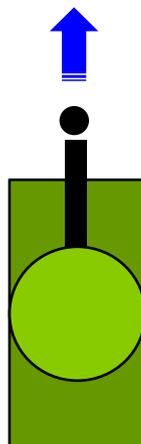
## Задания

---

«3»: Моделирование стрельбы слева направо.  
После выхода за границу экрана снаряд  
возвращается в исходную точку.



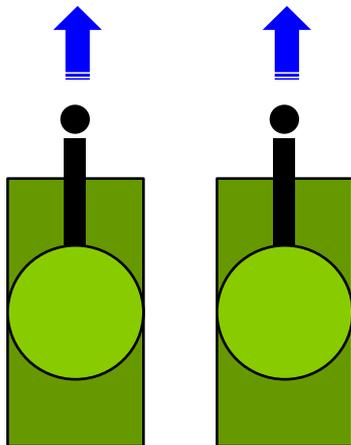
«4»: Дорисовать танк, из дула которого вылетает снаряд:



# Задания

---

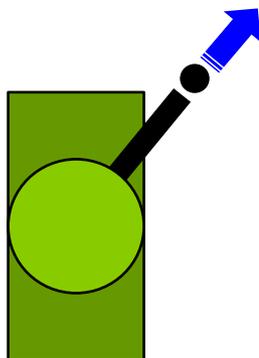
«5»: Два танка стреляют одновременно.



# Задания

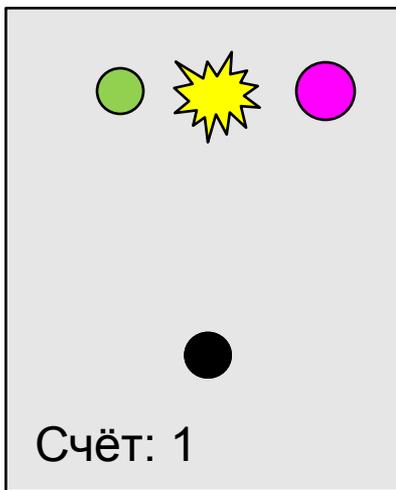
---

«6»: Танк с поворотной башней. Выстрел происходит в ту сторону, в которую повернут ствол:



# Стрельба по тарелкам

(0,0)



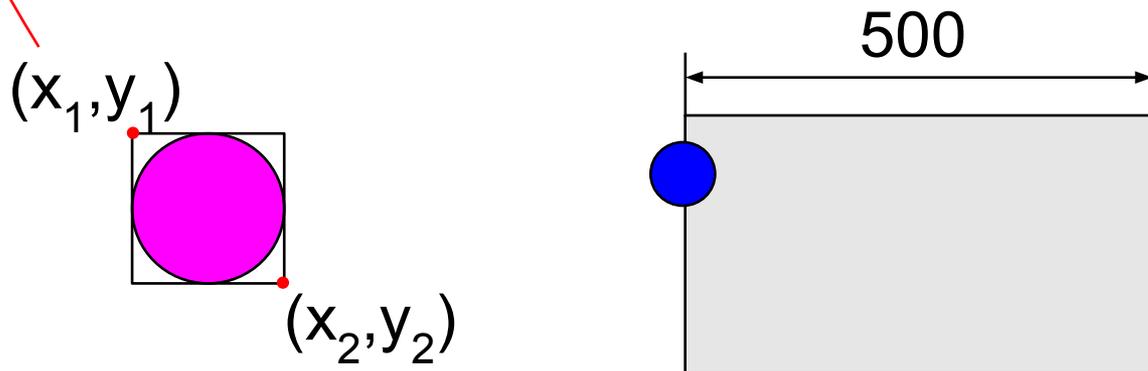
1. создать объекты-тарелки  
`createPlates`
2. двигать тарелки  
`movePlates`
3. проверить попадание в какую-нибудь тарелку  
`checkCollision`
4. проверить попадание в конкретную тарелку  
`hit`

## Создание массива тарелок

```
def createPlates ( N ) :
    global plates # глобальный массив
    yPlates = 100 # у всех одна у-координата
    plates = [] # пока массив пустой
    for i in range (N) :
        brushColor( randColor() )
        p = circle( randint(0, 500) , # x центра
                   yPlates, # у центра
                   randint(10, 20) ) # радиус
        plates.append(p) # добавить в массив
    ...
createPlates ( 5 ) # вызов процедуры
```

# Движение тарелок

```
def movePlates():  
    global plates # глобальный массив  
    for p in plates: # для каждой тарелки  
        moveObjectBy(p, -2, 0) # сдвиг на 2 влево  
        x1, y1, x2, y2 = coords(p)  
        if x1 < 0: # если вышла за границу...  
            # перескочить вправо на ...  
            moveObjectBy(p, randint(500, 600), 0)
```

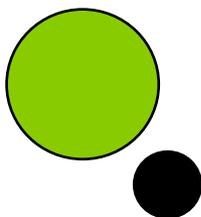


## Попадание в какую-нибудь тарелку

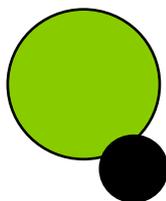
```
def checkCollision():
    global isFlying, bullet, plates
    for p in plates:
        if hit(p):
            # перекинуть тарелку направо
            moveObjectBy(p, randint(500, 600), 0)
            # снаряд в начальную точку
            moveObjectTo(bullet, x0-r, y0-r)
            isFlying = False # остановить снаряд
            break # только одну тарелку за раз
```

`hit(p)` – логическая функция, которая возвращает `True`, если снаряд (`bullet`) столкнулся с тарелкой `p`, и `False`, если не столкнулся.

# Попал ли снаряд в данную тарелку?



не попал



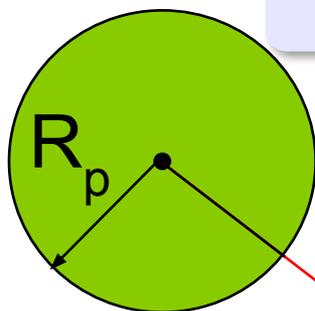
попал



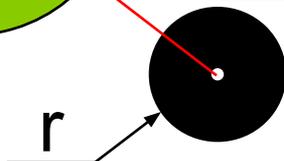
Как записать условие «попал» в виде формулы?

$(x_p, y_p)$

расстояние между центрами



$$d = \sqrt{(x_b - x_c)^2 + (y_b - y_c)^2}$$



$(x_b, y_b)$



«Попал»:  $d^2 \leq (r + R_p)^2$

# Попал ли снаряд в данную тарелку?

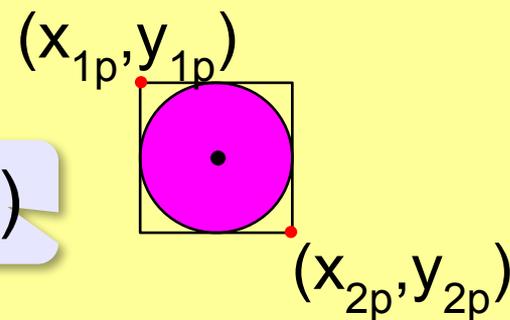
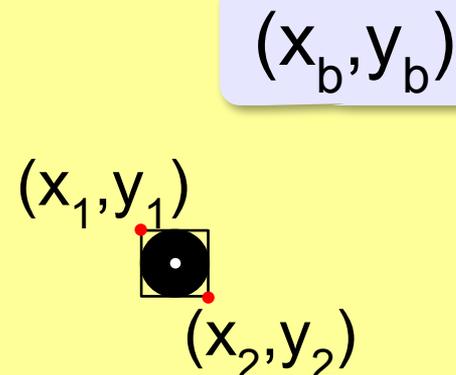
```
def hit(p):  
    ...  
    if d2 <= (Rp + r)**2:  
        return True  
    else:  
        return False
```



```
def hit(p):  
    ...  
    return d2 <= (Rp + r)**2
```

# Попал ли снаряд в данную тарелку?

```
def hit(p):  
    global bullet  
    # координаты снаряда  
    x1, y1, x2, y2 = coords(bullet)  
    xb = x1 + r # центр снаряда  
    yb = y1 + r  
    # координаты тарелки  
    x1p, y1p, x2p, y2p = coords(p)  
    xp = (x1p + x2p) / 2  
    yp = (y1p + y2p) / 2  
    Rp = (x2p - x1p) / 2  
    d2 = (xb - xp)**2 + (yb - yp)**2  
    return d2 <= (Rp + r)**2
```



# Как вызывать эти функции?

```
def update():
    global isFlying, bullet
    movePlates()
    if isFlying:      # если летит...
        y = coords(bullet)[1]
        if y < 0:    # если улетел...
            isFlying = False
            moveTo(bullet, x0-r, y0-r)
        else:        # летит дальше...
            moveObjectBy(bullet, 0, -5)
            checkCollision()
    ...
onTimer(update, 30)
```

вызывается  
каждые 30 мс

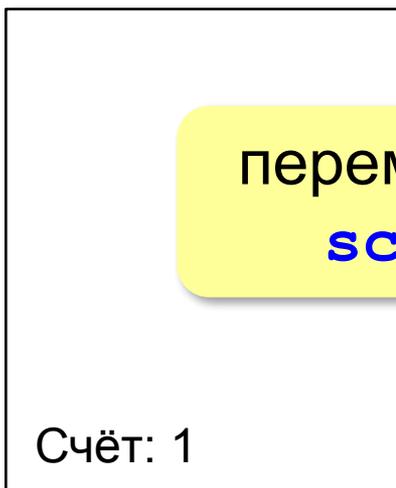
# Полная программа

```
from graph import *
    # тут все функции
    ...
x0 = 200; y0 = 400; r = 3
createPlates( 5 )
brushColor("black")
bullet = circle(x0, y0, r)
isFlying = False

onKey(keyPressed)
onTimer(update, 30)

run()
```

# Как вывести счёт игры?



переменная  
`score`



Как и когда  
изменяется `score`?

Сначала: `score = 0`

При попадании:

`score += 1`

Метка (элемент типа `label`)

(`x,y`)

фон

Создание метки:

```
lbl = label("Счёт: 0", 10, 200, bg="white")
```

Изменение текста метки:

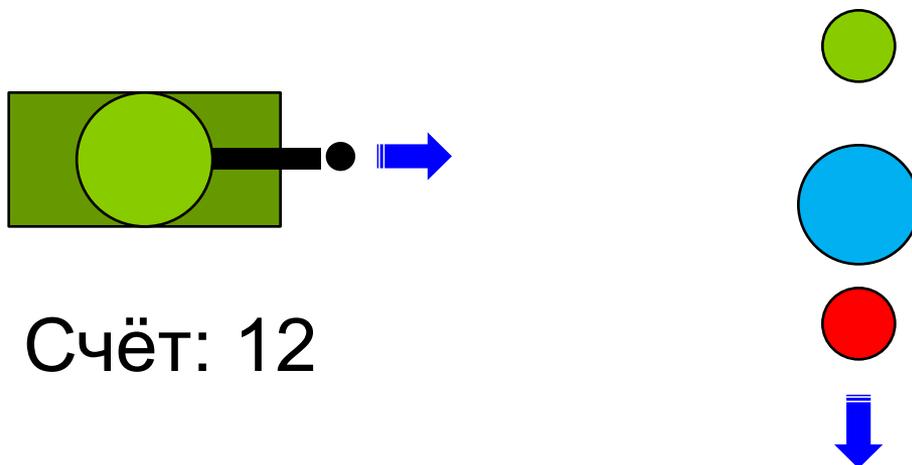
строка из числа

```
lbl["text"] = "Счёт: " + str(score)
```

## Задания

---

- «3»: Собрать и запустить программу. Увеличить скорость снаряда.
- «4»: Выполнить задание на «3» для случая стрельбы слева направо (тарелки летят сверху вниз). Дорисовать танк, из дула которого вылетает снаряд.

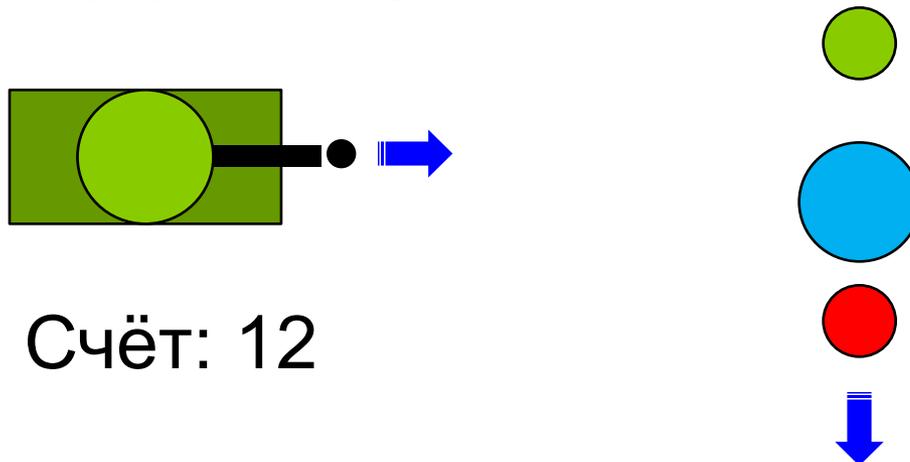


Счёт: 12

# Задания

---

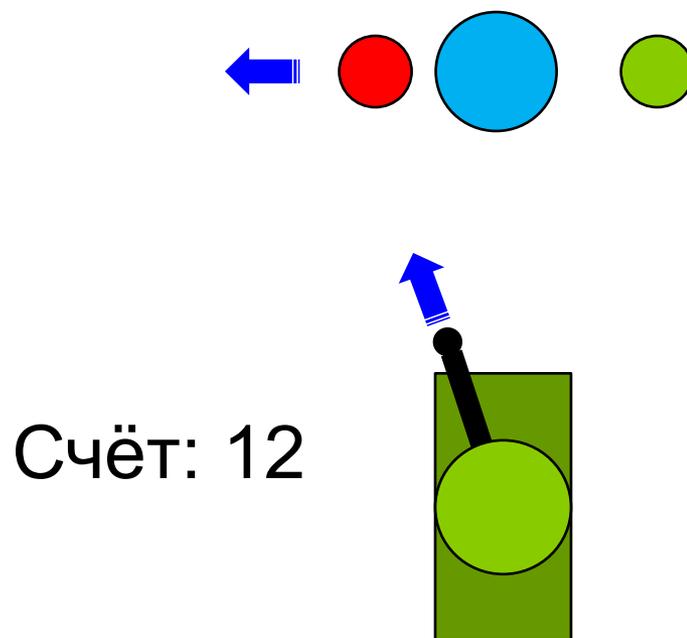
«5»: Дополнить задание на «4»: за попадание в более мелкую тарелку игрок получает больше баллов.



## Задания

---

«6»: Сделать танк с вращающейся пушкой.  
Снаряд вылетает из ствола в том же направлении. За попадание в более мелкую тарелку игрок получает больше баллов.



# Конец фильма

---

**ПОЛЯКОВ Константин Юрьевич**

д.т.н., учитель информатики

ГБОУ СОШ № 163, г. Санкт-Петербург

[kpolyakov@mail.ru](mailto:kpolyakov@mail.ru)