Виктор Рабинович

Python для детей

Анимация с черепашьей графикой

12+

Виктор Рабинович

Python для детей. Анимация с черепашьей графикой

«ЛитРес: Самиздат»

2020

Рабинович В.

Python для детей. Анимация с черепашьей графикой /
В. Рабинович — «ЛитРес: Самиздат», 2020

В нашей книге, написанной для обучения детей 12+ анимационной технике с использованием современного языка программирования Python, мы используем простейшую графическую библиотеку языка: черепашью графику(Turtle library). Считается, что библиотека Turtle предназначена в основном для рисования геометрических фигур и анимаций с использованием стандартных, встроенных в библиотеку изображений таких как квадрат, круг, черепашка, стрелка (назовем эти изображения базовыми примитивами). Однако это не так. Простая и понятная для детей библиотека Turtle имеет в своем составе команды, позволяющие детям создавать отличные анимационные проекты, наподобие тем, которые создаются с помошью блочного языка программирования Scratch, широко распространенного в настоящее время для обучения детей. Мы научимся добывать из интернета нужные нам для проекта изображения, научимся вводить их в программу и контролировать движения этих изображений с помощью команд библиотеки.

Содержание

Введение

Анимация представляет собой способ, при котором последовательно показываемые на экране статические изображения сменяют друг друга так быстро, что в результате имитируется непрерывное движение. Каждая картинка называется кадром. Каждый кадр должен немного отличаться от предыдущего, и быстрое отображение кадров один за другим создает иллюзию непрерывного движения. Кадры сменяются с определенной скоростью около 12 или более кадров в секунду, чтобы человек мог воспринимать их как анимацию. Современный фильм обычно использует 24 кадра в секунду.

В нашей книге, написанной для обучения детей 12+ анимационной технике с использованием современного языка программирования Python, мы используем простейшую графическую библиотеку языка: черепашью графику(Turtle library). Считается, что библиотека Turtle предназначена в основном для рисования геометрических фигур и анимаций с использованием стандартных, встроенных в библиотеку изображений таких как квадрат, круг, черепашка, стрелка (**назовем эти изображения базовыми примитивами**). Однако это не так. Простая и понятная для детей библиотека Turtle имеет в своем составе команды, позволяющие детям создавать отличные анимационные проекты, наподобие тем, которые создаются с помошью блочного языка программирования Scratch, широко распространенного в настоящее время для обучения детей. Мы научимся добывать из интернета нужные нам для проекта изображения, научимся вводить их в программу и контролировать движения этих изображений с помощью команд библиотеки. Для того, чтобы освоить материал книги, необходимо познакомиться с командами библиотеки Turtle а также изучить базовые функции языка Python, такие как циклы, списки, переменные и т. д. Для этого мы рекомендуем несколько отличных книжек, специально написанных для детей:

a) Джейсон Бриггс, Python для детей;

b) Программирование, самоучитель: https://www.labirint.ru/books/575392/;

c) К. Вордерман, Программирование на Python: Иллюстрированное руководство для детей;

d) Ханс Георг Шуман, Python для детей.

Простейшая анимация с одним базовым примитивом

Основными стандартными (базовыми) графическими примитивами черепашьей библиотеки, как было сказано ранее, являются следующие: квадрат, треугольник, черепашка и круг. Под простейшей анимацией будем понимать перемещение одного или нескольких из этих изображений вдоль экрана компьютера. Контролируемое перемещение изображений обеспечивается командами библиотеки а также базовыми командами языка Python(Питон).

Движение черепашки по прямой

Рассмотрим простейшую анимацию – перемещение базового примитива чрепашки вдоль горизонтали по окну экрана. Соответствующий код и три отдельных анимационных кадра представлены ниже в таблице 1.

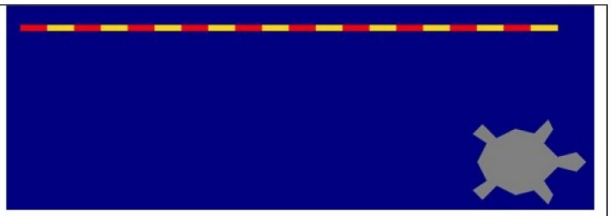
Таблица 1

```
import turtle,time
wn=turtle.Screen()
wn.bgcolor('navy') #----------------*
wn.setup(1000,700)
#---------------------------------------1
line=turtle.Turtle('square')
line.shapesize(0.4,2)
line.up()
line.hideturtle()
line.goto(-450,200)
line.showturtle()
for i in range(10):
   line.color('red')
   line.fd(40)
   line.stamp()
   line.color('gold')
   line.fd(40)
   line.stamp()
#-----------------------------------2
t=turtle.Turtle('turtle')
t.color('grey')
t.up()
t.hideturtle()
t.goto(-300,0)
t.shapesize(3)
t.showturtle()
#--------------------------------3
for m in range(600):
   t.fd(1)
   time.sleep(0.001)
#--------------------------------4
```

Строка, обозначенная символом *(wn.bgcolor('navy')) задает цвет окна экрана, строки кода, расположенные между цифрами 1 и 2 обеспечивают прорисовку пунктирной прямой линии, относительно которой передвигается черепаха. Обьект, обозначенный латинской буквой t, определяет обьект-черепашку, к которому применяются строчки кода, задающие ее движение (строчки кода расположены между цифрами 3 и 4). Скорость движения определяется строкой time.sleep(0.001), где время задержки каждого кадра цикла равно 0.001-й секунды. Для того, чтобы воспользоваться этой строкой, необходимо импортировать библиотеку time (смотри первую строку программы). Для замедления движения необходимо увеличить время задержки. Последнюю строку можно убрать вовсе для увеличения скорости движения черепашки. Однако скорость движения при этом заметно не уменьшается. Заметного увеличения скорости движения можно добиться, введя в код следующую строку: turtle.tracer(2). Эта строка может быть вставлена, например, перед циклом, определяющим движение черепахи (вместо строки #–3). Регулировать скорость движения в таком случае можно изменяя целое положительное число в строке turtle.tracer(число) и число в строке time.sleep(число).

Интересный анимационный эффект может быть получен добавлением в блок команд между цифрами 3 и 4 строки t.shapesize(1+m*0.01). Соответствующий измененный блок показан ниже.

Таблица 2

for m in range(600):
  t.fd(1)
  t.shapesize(1+0.01*m)
  time.sleep(0.001)
	

Строка t.shapesize(1+0.01*m) увеличивает размеры черепахи при ее движении. Последний кадр анимации с использованием приведенного кода показаны в правой части таблицы. Изменяя величину коэффициента при переменной m можно варьировать размером черепахи.

Показанная выше анимация ограничена во времени, поскольку последний кадр соответствует номеру m в цикле, равному 599. Анимацию можно сделать бесконечной во времени, например, изменив код цикла как показано в таблице 3:

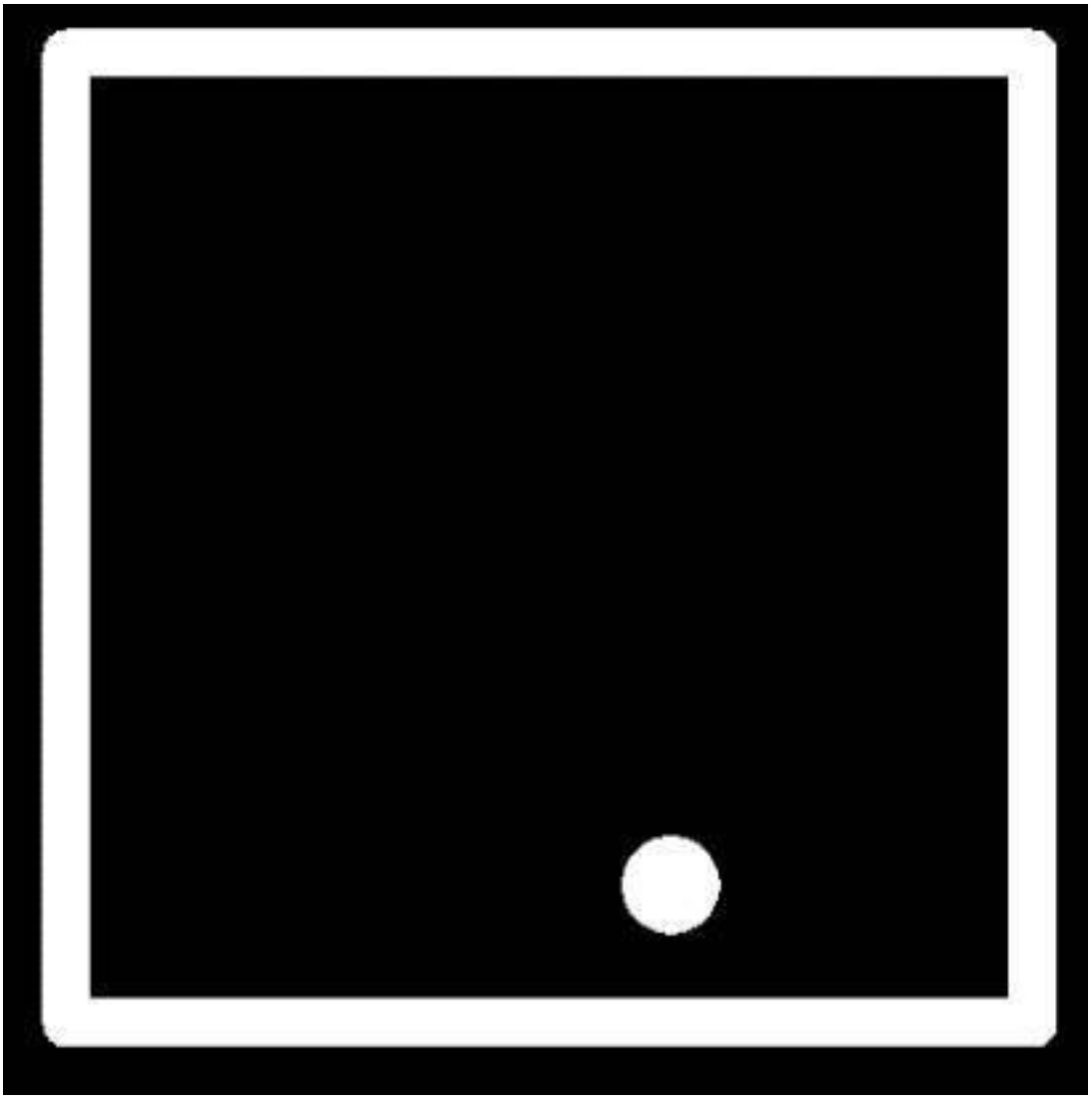
Таблица 3

while True:
  for m in range (600):
  t.fd(1)
  t.shapesize(1+0.01*m)
  time.sleep(0.001)
  t.hideturtle()
  t.goto(-300,0)
  t.showturtle()
	while True:
	  t.setheading(0)
	  for m in range (600):
	  t.fd(1)
	  t.shapesize(1+0.01*m)
	  time.sleep(0.001)
	  t.setheading(180)
	  for m in range (600):
	  t.fd(1)
	  t.shapesize(1-0.01*m+5.99)
	  time.sleep(0.001)

Левая часть таблицы соответствует движению черепахи слева направо, причем после окончания каждого цикла черепаха, прячется, возвращается в левую часть экрана, появляется и снова движется слева направо. Правая часть таблицы соответствует движению черепахи слева направо, после окончания цикла, определяющего движение черепахи слева направо, движение происходит справа налево, затем вновь слева направо и т. д.

Отражение шарика от стенок коробки

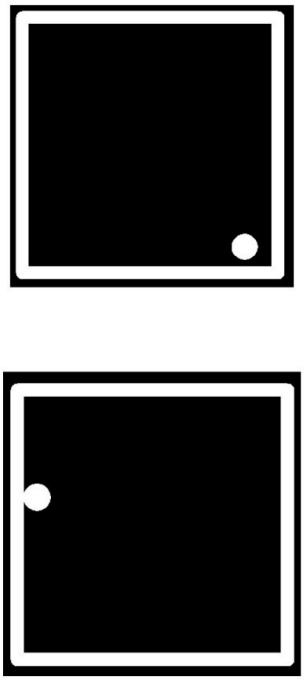
Следующим примером является анимационный проект, демонстрирующий шарик бегающий внутри квадратной коробки и отражающийся от ее стенок. Статический кадр, демонстрирующий предлагаемый сценарий, показан на ниже.



Соответствующий скрипт написанный на Python с использованием кодов черепашьей библиотеки представлен в нижней таблице 4. В качестве базового елемента черепашьей графики используем круг (строка, обозначенная цифрой #1). Программа составлена так, что шарик в виде круга непрерывно бегает внутри коробки, отражаясь от стенок по законам геометрической оптики.

Таблица 4

```
import turtle
wn=turtle.Screen()
wn.bgcolor('black')
t=turtle.Turtle('circle') #----------1
t1=turtle.Turtle()
t.turtlesize(2)
t.hideturtle()
t.color('white')
t.penup()
t1.hideturtle()
t1.pensize(20)
t1.penup()
t1.goto(-200,200)
t1.pendown()
t1.color('white')
for m in range(4):
   t1.fd(400)
   t1.right(90)
t.goto(100,100)
dx,dy=1.3,2.3
X,Y=50,50
t.speed('fastest')
t.showturtle()
while True:#----------2
   t.goto(X+dx,Y+dy)
   X,Y=t.xcor(),t.ycor()
   if X<-175 or X>175:# ----------3
      dx=dx*-1# ----------4
   if Y<-175 or Y>175:#----------5
      dy=dy*-1#----------6
```



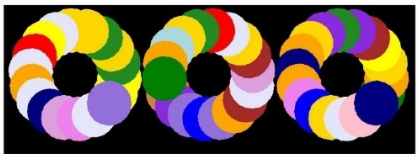
Непрерывное движение шарика обеспечивается бесконечным циклом, начинающимся со строки, обозначенной, как #2. Строки #3, #4, #5 и #6 отвечают за отражение шарика от стенок по законам геометрической оптики.

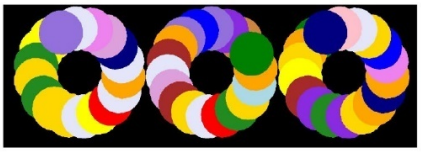
Вращающиеся окружности

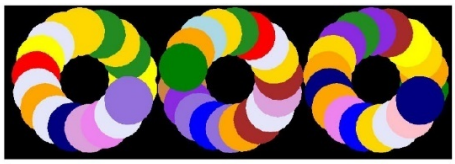
Рассмотрим пример, демонстрирующий непрерывное вращение трех окружностей, составленных в свою очередь из кружков небольшого радиуса. Все круги разного цвета образованы с помощью базового примитива: circle. Программа, реализующая этот алгоритм, показана ниже, несколько статических кадров – в правой части таблицы 5.

Таблица 5

```
import turtle,random,time
wn=turtle.Screen()
wn.setup(1000,800,100,0)
turtle.tracer(100)
turtle.bgcolor('black')
q,p,r=[],[],[]#---------------------------------1

clr=['red','blue','violet',\
    'green','navy','orange',\
    'pink','gold',\
    'brown','light blue'\
    ,'peru','blue violet','plum',\
    'lavender','forest green',\
    'medium purple','yellow']
#-------------------------------------------------2
def object(tu,x,y,i):
   t=turtle.Turtle('circle')
   tu.append(t)
   tu[i].shapesize(5)
   tu[i].up()
   tu[i].color(random.choice(clr))
   tu[i].goto(x,y)
   tu[i].circle(100,20*i)
#-------------------------------------------------3
for m in range(18):
   object(q,0,-100,m)
   object(p,-300,-100,m)
   object(r,300,-100,m)
#-------------------------------------------------4
m=-1
while True:
   m=m+1
   m1=m%18
   q[m1].circle(100,5)
   p[m1].circle(100,-5)
   r[m1].circle(100,-5)
#-------------------------------------------------5
```







Поясним ключевые (основные) коды приведенной программы. Строка #1 определяет три <пустых> списка(массива) обьектов: базисных черепашьих примитивов, каждый из которых будет заполнен кругами разного цвета. Строки, расположенные между номерами 2 и 3 определяют функцию для рисования кругов, строки между номерами 3 и 4 формируют три массива q[m], p[m], r[m], где m=0,1,2,3…17. Наконец, строки между номерами 4 и 5 задают вращение массивов: первый и третий- почасовой стрелке и второй средний против часовой стрелки. Видоизменив последнюю часть программы, можно остановить вращение окружностей, кликнув стрелкой мышки на любую часть экрана. Соответствующая измененная часть кодов показана ниже:

Таблица 6

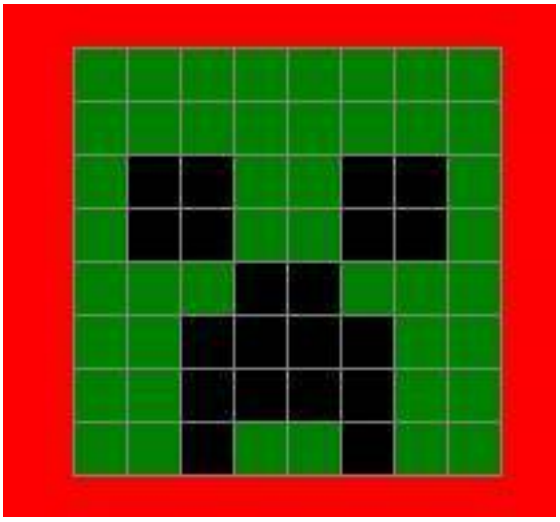
```
m=-1
c=0
while True:
    m=m+1
    m1=m%18
    q[m1].circle(100,5)
    p[m1].circle(100,-5)
    r[m1].circle(100,-5)
    if c==1:
        break
    def h(a,b):
        global c
        c=1
    wn.onclick(h)
```

```
m=-1
c=0
go=True
while go:

    m=m+1
    m1=m%18
    q[m1].circle(100,5)
    p[m1].circle(100,-5)
    r[m1].circle(100,-5)
    if c==1:
        go=False

    def h(a,b):
        global c
        c=1
    wn.onclick(h)
```

В левой и правой части таблицы показаны части кода, которые следует поменять на строки между номерами 4 и 5 основного кода (таблица с изображениями в правой части). Скрипты левой и правой части отличаются логикой приостановки кода. В левой части остановка осуществляется по команде <break>, в правой – используются логические операторы True и False. Обе части для остановки используют функцию onclick(), которая позволяет управлять движением на экране по щелчку мыши.

Простейшая пиксель анимация

Пиксельная графика – это форма цифровой графики, в которой с помощью программного обеспечения, изображения создаются с помощью пикселей. Пиксель это маленький квадратик на экране компьютера, который можно закрашивать тем или иным цветом. Когда маленькие разноцветные квадратики (пиксели) собраны вместе, они образуют изображение для человеческого глаза. Таким образом можно сформировать разнообразные изображения, например, изображения животных или человека. Собрав из пикселей изображение и используя компьютерную программу, можно создать анимацию, т. е. заставить изображение передвигаться, изменять выражение лица, увеличивать или уменьшать изображение в размерах и т. д. В этом разделе мы познакомимся, как с помощью простых команд Питона и черепашьей графики создать анимацию пиксельных изображений. Наш первый пример – создаем простейшее изображение, показанное ниже (слева)



Соответствующая программа представлена в таблице 7.

Поясним коротко алгоритм работы программы. На первом шаге создаем, пиксель на экране монитора в виде квадрата, который имеется в черепашьей библиотеке (коды между строками 1 и 2). Размер квадрата задан переменной delta. Изменяя ее, можно увеличивать либо уменьшать размеры всего изображения. Расстановка пикселей на экране задается строками программы, расположенными между линиями 3 и 4. Эти строки описывают одномерные списки row0, row1, row2,…row7, и каждый из этих списков является элементом двумерного списка Pixels=[row0,row1,row2,…row7] (списки в списке), на английском(list). Строки между линиями 6 и 7 определяют функцию, которая расставляет пиксели в соответствии с картой двумерного массива-списка. Обратим внимание на строку, обозначенную *. Эта строка служит для раскрашивания пикселя с координатой в ряду по номеру i и координатой в столбце по номеру j. В нашей мозаике цветов всего два цвета: зеленый и черный, как задано кодом строки 5: цифра 0 соответствует зеленому цвету, цифра 1-черному цвету. Если в строке, обозначенной * убрать слово grey, исчезнет окоймляющий пиксель ободок серого цвета.

Таблица 7

```
import turtle,random
import time
wn=turtle.Screen()
wn.setup(1000,1000,500,0)
turtle.bgcolor('red')
#-------------------------------------------------1
t= turtle.Turtle('square')
delta=1
t.shapesize(delta)
t.penup()
#-------------------------------------------------2
wn.tracer(4)
#-------------------------------------------------3
row0=[0,0,0,0,0,0,0,0]
row1=[0,0,0,0,0,0,0,0]
row2=[0,1,1,0,0,1,1,0]
row3=[0,1,1,0,0,1,1,0]
row4=[0,0,0,1,1,0,0,0]
row5=[0,0,1,1,1,1,0,0]
row6=[0,0,1,1,1,1,0,0]
row7=[0,0,1,0,0,1,0,0]
pixels=[row0,row1,row2,row3,row4,row5,row6,row7]
#-------------------------------------------------4
clr= ['green','black']#-----------------------------5

t.hideturtle()
#-------------------------------------------------6
def pix_art(x,y):
   for i in range (0,len(pixels)):
      t.goto(x,y-i*20*delta)
      for j in range (0,len(pixels)):
         t.color('grey',clr[pixels[i][j]])#----------------------*

         t.showturtle()
         t.stamp()
         t.fd(20*delta)
   t.hideturtle()
#-------------------------------------------------7
pix_art(0,0)
```

К показанному выше коду добавим следующие строки:
Таблица 8

```
while True:
   wn.update()
   t.clear()
   pix_art(random.randint(-300,300),random.randint(-300,300)) #----------------------**
   t.hideturtle()
   time.sleep(0.3)
```

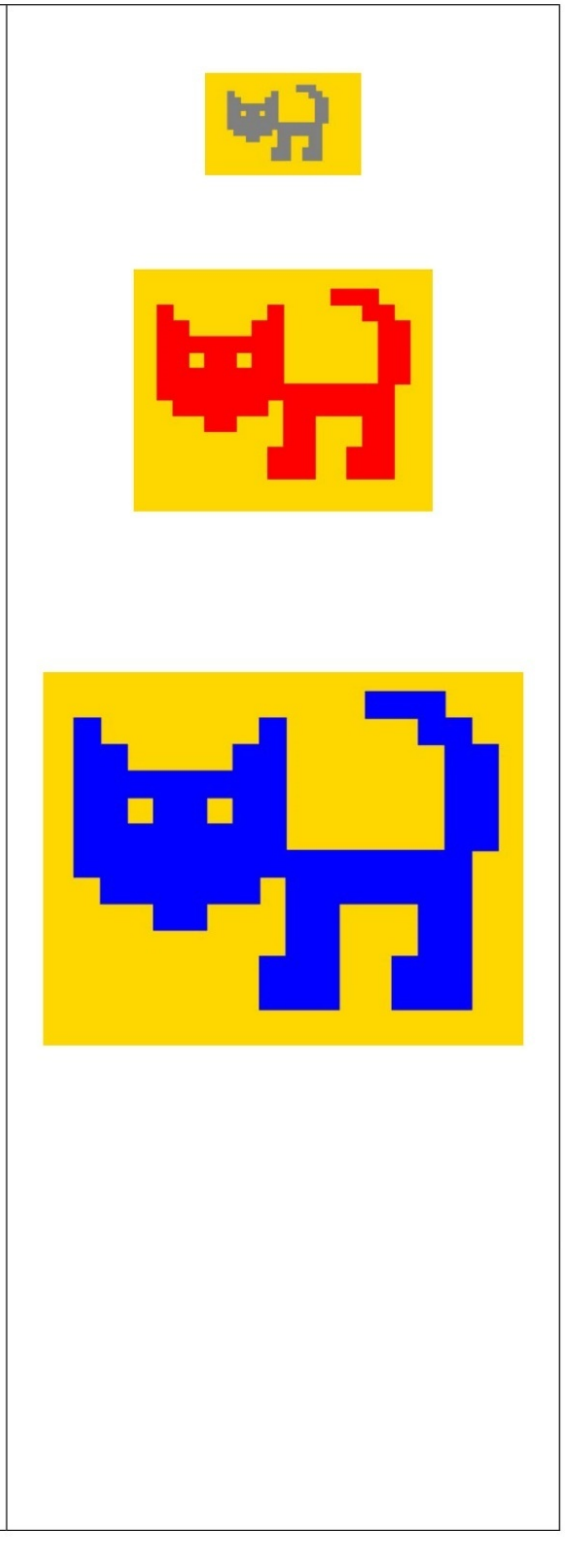
В результате получим изображение, которое появляется и исчезает в различных точках экрана. Место расположение изображения задается датчиком случайных чисел, определяемых строкой, обозначенной двумя звездочками **. Для того, чтобы воспользоваться датчиком случайных чисел, мы импортировали библиотеку random.

Анимация собаки в стиле пиксель-арт

Приведем еще один промер анимационного изображения построенного с помощью базового примитива черепашьей графики. Используем примитив квадрат для построения собаки в стиле пиксель арт. Соответствующий код показан ниже в левой части таблицы 9. С правой стороны показаны несколько статических кадров полученного изображения. Поясним коротко алгоритм работы программы. На первом шаге создаем, пиксель на экране монитора в виде квадрата, который имеется в черепашьей библиотеке (строка #1). В нашем случае изображением, которое мы хотим построить, является собака, состоящая из множества(мозаики) собранных в определенном порядке пикселей. Расстановка пикселей задается строками программы, расположенными между линиями #2 и #3. Эти строки описывают одномерные списки Pix1,Pix2,Pix3,…Pix12, и каждый из этих списков является элементом двумерного списка Pixels=[Pix1,Pix2,Pix3,…Pix12] (списки в списке), на английском(list). Цифра 0 в одномерных списках означает отсуттствие пихеля в соответствующем месте экрана, цифра 1- присутствие пикселя. Строки между линиями 4 и 5 расставляют пихели в соответствии с картой двумерного массива-списка.В результате мы получаем изображение собаки. Размер изображения определяется количеством пикселей, размером пикселя delta и шагом между пикселями, который всегда равен 20*delta.

Таблица 9

```
import turtle,time
t= turtle.Turtle('square')              #1
t.penup()
t.color('black')
t.hideturtle()
wn=turtle.Screen()
wn.setup(700,500,400,100)
wn.bgcolor('gold')
wn.tracer(5)
#-------------------------------------------2
pix1=[0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0]
pix2=[1,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0]
pix3=[1,1,0,0,0,0,1,1,0,0,0,0,0,0,1,1]
pix4=[1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1]
pix5=[1,1,0,1,1,0,1,1,0,0,0,0,0,0,1,1]
pix6=[1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1]
pix7=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0]
pix8=[0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0]
pix9=[0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0]
pix10=[0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0]
pix11=[0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,1,0]
pix12=[0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,1,0]
pixels=[pix1,pix2,pix3,pix4,pix5,pix6,\
      pix7,pix8,pix9,pix10,pix11,pix12]
#-------------------------------------------3
clr=['grey','black','red','blue','brown','gold']
while True:
   for q in range(6):
      t.color(clr[q])
      delta=5*(q+1)
      t.shapesize(0.25*(q+1))
#-------------------------------------------4
      for i in range (0,12):
         t.goto(-100,100-i*delta)
         for j in range (0,16):
            if pixels[i][j]==1:
               t.showturtle()
               t.stamp()
            t.forward(delta)
      time.sleep(0.5)
      wn.update()
      t.clear()
t.hideturtle()
#-------------------------------------------5
```



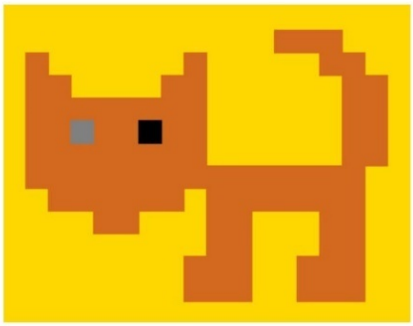
Как видно из приведенного кода (строки между номерами 3 и 4), программа повторяется бесконечное число раз для 6-ти разных размеров пиксельной ячейки (строки delta=5*(q+1), t.shapesize(0.25*(q+1)). При каждом заходе в цикл по q изменяется цвет собаки, и таким образом создается анимационный эффект.

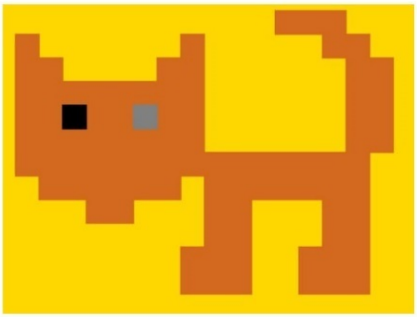
Моргающие глаза

Видоизменим программу, показанную выше, и получаем собаку с моргающими глазами. Соответствующий код показан в таблице 10

Таблица 10

```
import turtle,time
t= turtle.Turtle('square')
t.penup()
t.color('black')
t.hideturtle()
wn=turtle.Screen()
wn.setup(700,500,400,100)
wn.bgcolor('gold')
wn.tracer(1)
#-----------------------------------------------------1
pix1=[0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0]
pix2=[1,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0]
pix3=[1,1,0,0,0,0,1,1,0,0,0,0,0,0,1,1]
pix4=[1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1]
pix5=[1,1,0,1,1,0,1,1,0,0,0,0,0,0,1,1]
pix6=[1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1]
pix7=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0]
pix8=[0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0]
pix9=[0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0]
pix10=[0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0]
pix11=[0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,1,0]
pix12=[0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,1,0]
pixels=[pix1,pix2,pix3,pix4,pix5,pix6,\
      pix7,pix8,pix9,pix10,pix11,pix12]
#-----------------------------------------------------2
t.color('chocolate')
delta=20
#-----------------------------------------------------3
for i in range (0,12):
   t.goto(-100,100-i*delta)
   for j in range (0,16):
      if pixels[i][j]==1:
         t.showturtle()
         t.stamp()
      t.forward(delta)
t.hideturtle()
#-----------------------------------------------------4
t1=turtle.Turtle('square')
t1.shapesize(1)
t1.up()
t1.color('black')
t1.goto(-60,20)

t2=turtle.Turtle('square')
t2.shapesize(1)
t2.up()
t2.color('black')
t2.goto(0,20)

t3=t1.clone()
t3.color('grey')
t4=t2.clone()
t4.color('grey')
#-----------------------------------------------------5
while True:
   t3.showturtle()
   t4.hideturtle()
   time.sleep(0.5)
   t3.hideturtle()
   t4.showturtle()
   time.sleep(0.5)
#-----------------------------------------------------6
```





Строки между номерами #1 и #2 определяют, как и в предыдущем примере, пиксельную карту собаки, коды между строками #3 и #4 устанавливают пиксели в нужное положение на окне экрана, обьекты черепашьей графики t2, t3, t4 I t5 –глаза собаки и, наконец, коды между строками #5 и #6 в бесконечном цикле определяют нужное нам моргание.

Конец ознакомительного фрагмента.