

Виктор Рабинович

Python  
для  
детей

Анимация  
с черепашьей  
графикой



12+

**Виктор Рабинович**

# **Python для детей. Анимация с черепашьей графикой**

*http://www.litres.ru/pages/biblio\_book/?art=57284673  
SelfPub; 2020*

## **Аннотация**

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 помощью 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	5
Простейшая анимация с одним базовым примитивом	7
Анимация с несколькими примитивами	27
Анимация изображений, построенных из полигонов	36
Особенности анимации с большим количеством спрайт-файлов	88
Анимация с управлением от кнопки компьютерной мыши	95
28. Вертолет с парашютистами	99
Заключение	103

# 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 помощью блочного языка про-

граммирования 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**

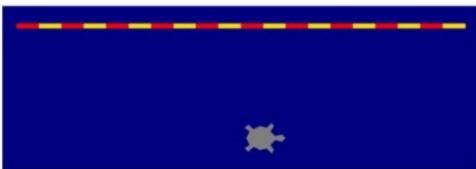
Рассмотрим простейшую анимацию – перемещение базового примитива черепашки вдоль горизонтали по окну экрана. Соответствующий код и три отдельных анимационных кадра представлены ниже в таблице 1.

**Таблица 1**

```

import turtle,time
wn=turtle.Screen()
wn.bgcolor('navy') *-----1
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. Объект, обозначенный латинской буквой t, определяет объект-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

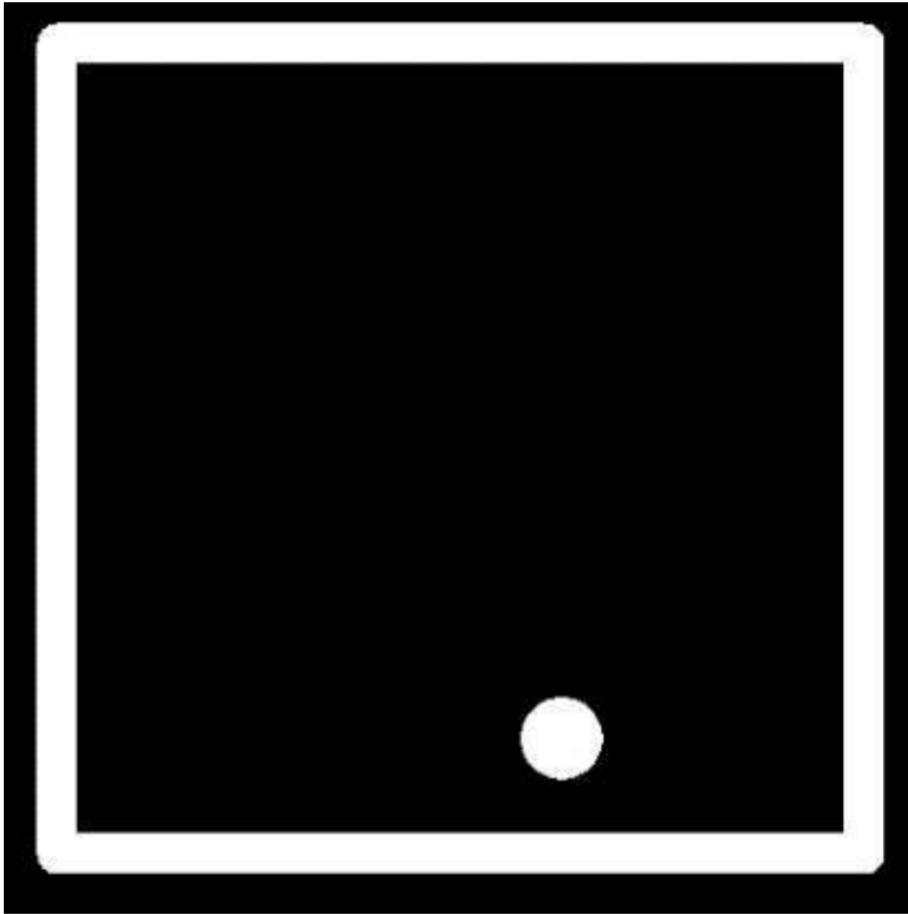
```
while True:
    for m in range (600):
        t.fd(1)
        t.shapesize(1+0.01*m)
        time.sleep(0.001)
    t.hideturtle()
    t.goto(-300,0)
    t.showturtle()
```

```
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.

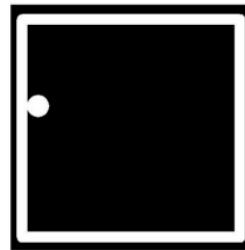
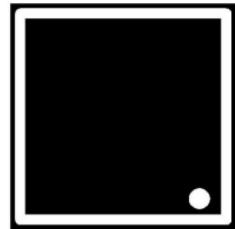


Соответствующий скрипт написанный на Python с использованием кодов черепашьей библиотеки представлен в нижней таблице 4. В качестве базового элемента черепашьей графики используем круг (строка, обозначенная цифрой #1). Программа составлена так, что шарик в виде круга непрерывно бегает внутри коробки,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

нам 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. Стока #1 определяет три <пустых> списка(массива) объектов: 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  $m=0,1,2,3\dots 17$ . 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

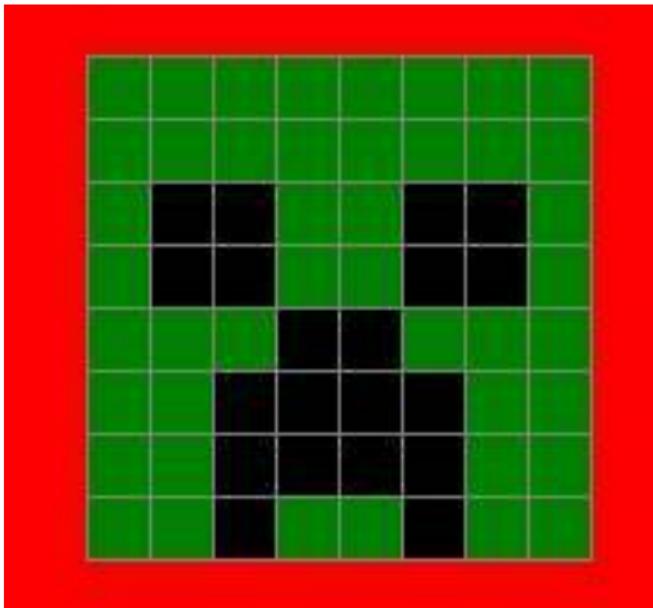
<pre>m=-1 c=0 while True:     m=m+1     m1=m%18     q[m1].circle(100,5)     p[m1].circle(100,-5)     r[m1].circle(100,-5)     if c==1:         break     def h(a,b):         global c         c=1     wn.onclick(h)</pre>	<pre>m=-1 c=0 go=True while go:     m=m+1     m1=m%18     q[m1].circle(100,5)     p[m1].circle(100,-5)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

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, расположеными 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 окаймляющий 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]
row1=[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 пример 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

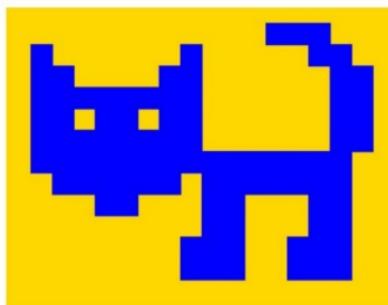
линиями #2 и #3. Эти строки описывают одномерные списки Pix1,Pix2,Pix3,...Pix12, и каждый из этих списков является элементом двумерного списка Pixels=[Pix1,Pix2,Pix3,...Pix12] (списки в списке), на английском(list). Цифра 0 в одномерных списках означает отсутствие пикселя в соответствующем месте экрана, цифра 1- присутствие пикселя. Строки между линиями 4 и 5 расставляют пиксели 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  $20 * \text{delta}$ .

## 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=[0,0,0,0,0,0,0,0,0,1,1,1,0,0]
pix2=[1,0,0,0,0,0,1,0,0,0,0,0,1,1,0]
pix3=[1,1,0,0,0,0,1,1,0,0,0,0,0,1,1]
pix4=[1,1,1,1,1,1,1,1,0,0,0,0,0,1,1]
pix5=[1,1,0,1,1,0,1,1,0,0,0,0,0,1,1]
pix6=[1,1,1,1,1,1,1,1,0,0,0,0,0,1,1]
pix7=[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=[0,0,0,0,0,0,0,1,1,0,0,0,1,1,0]
pix11=[0,0,0,0,0,0,1,1,1,0,0,1,1,1,0]
pix12=[0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,0]
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  $\text{delta}=5*(q+1)$ ,  $\text{t.shapesize}(0.25*(q+1))$ ). 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

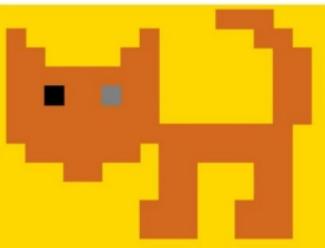
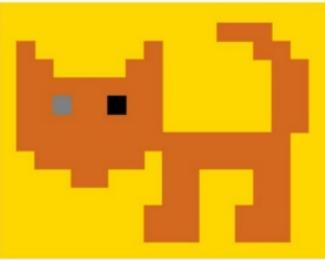
```

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=[0,0,0,0,0,0,0,0,0,0,1,1,0,0]
pix2=[1,0,0,0,0,0,0,1,0,0,0,0,1,0,0]
pix3=[1,1,0,0,0,0,1,1,0,0,0,0,0,1,1]
pix4=[1,1,1,1,1,1,1,0,0,0,0,0,1,1]
pix5=[1,1,0,1,1,0,1,1,0,0,0,0,0,1,1]
pix6=[1,1,1,1,1,1,1,1,0,0,0,0,0,1,1]
pix7=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,0]
pix8=[0,1,1,1,1,1,1,0,1,1,1,1,1,1,0]
pix9=[0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0]
pix10=[0,0,0,0,0,0,0,1,1,0,0,0,1,1,0]
pix11=[0,0,0,0,0,0,1,1,1,0,0,1,1,1,0]
pix12=[0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,0]
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+*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, объекты черепашьей графики t2, t3, t4 I t5 –глаза собаки и, наконец, коды между строками #5 и #6 в бесконечном цикле определяют нужное нам моргание.

# Анимация с несколькими примитивами

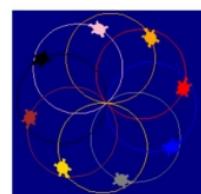
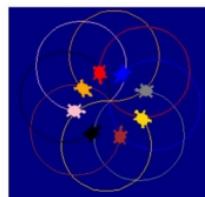
В этом разделе рассмотрены анимационные программы, позволяющие использовать одновременно несколько стандартных базовых графических примитивов библиотеки turtle.

## Чебурашки рисуют круги

В таблице 11 представлен код использующий несколько чрепашек, каждая из которых движется по своей траектории в плоскости экрана, создавая спиральный эффект.

Таблица 11

```
import turtle
turtle.tracer(2)
turtle.bgcolor('navy')
t=[]
clr=['orange','red','blue','grey','\n'gold','brown','black','pink']
for i in range(8):
    t.append(turtle.Turtle('turtle'))
    t[i].color(clr[i])
for j in range (8):
    t[j].right(j*45)
i=-1
while True:
    i=i+1
    i1=i%8
    t[i1].fd(10)
    t[i1].left(10)
```



Рассмотрим подробнее коды программы. Объекты черепашьей графики заданы в виде массива (строка #1а). Массив пустой и заполняется с помощью цикла, введенного строками #2,#3 и #4. Получили 8 черепашьих объектов, каждый из которых повернут относительно горизонтального положения на угол, равный  $45*j$ , где  $j$  принимает 8 разных значений (0,1,2,3...7). Таким образом головы всех восьми черепах перед началом движения развернуты по кругу на углы кратные 45 градусов. Бесконечное вращение каждой черепашки задается циклом в строке #8, причем величина  $i1$  принимает только 8 возможных значений, продвигая черепаху при каждом новом заходе в бесконечном цикле на 10 единиц вперед и поворачивая ее на 10 градусов влево. Два статических кадра представленной программы показаны в правой части таблицы 11.

### **Случайные блуждания частиц**

Таблица 12

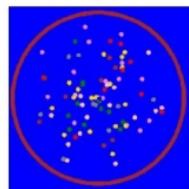
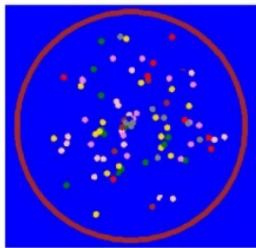
```

import turtle,random,time
turtle.bgcolor('blue')

Circle=turtle.Turtle()#-----1
Circle.hideturtle
Circle.pensize(10)
Circle.up()
Circle.goto(0,-200)
Circle.down()
Circle.color('brown')
Circle.circle(200)
turtle.tracer(30)
t=[]
clr=['brown','red','blue','gold']\
      'grey','pink','violet','green']
for i in range(100):
    t.append(turtle.Turtle('circle'))#-----2
    t[i].color(random.choice(clr))#-----3
    t[i].up()
    t[i].shapesize(0.5)

while True:
    for i in range(100):
        t[i].fd(random.randint(-20,20))
        t[i].rt(random.randint(0,360))
        Position=t[i].position()
        if abs(Position)>190:
            t[i].setposition(0,0)
        time.sleep(0.01)

```



Интересным применением базовых примитивов черепашьей графики является пример случайных блужданий (перемещений) частиц при условии, что направление движение каждой частицы на каждом последующем шаге случайно, не зависит от предыдущего, является случайным с равномерной функцией распределения в угловом интервале (0-360

градусов). Случайное блуждание частиц ограничено кругом с заданным радиусом. Если частица приближается к границе круга программа перебрасывает ее в точку с координатами (0,0), откуда она вновь начинает хаотическое движение. Соответствующая программа показана в таблице 12. В правой части таблицы показано два статических кадра анимации. Программа работает следующим образом. Черепаший объект Circle (строка #1) обозначает область в которой флюктуируют частицы. Каждая из 100-и частиц определяется объектом  $t[i]$ ( $i=0,1,2,3\dots99$ ) и задана в виде круга (строка #2), окрашенного своим цветом (строка #3). Бесконечный цикл, в котором задается хаотическое движение частиц, введен в программу последними 7-ю строками.

## **Движение автомобиля**

Таблица 13

```

import turtle
import time
t=turtle.Turtle()
t.hideturtle()
wn=turtle.Screen()
wn.setup(1000,500)
wn.bgcolor('dark blue')
turtle.tracer(3)
t=[]#-----1
shapes=['square','circle','circle','square']#---2
clr=['coral','gray','gray','green']#-----3
dx=[4,2,2,0.5]#-----4
dy=[12,2,2,50]#-----5

for i in range (4):
    t.append(turtle.Turtle())
    t[i].up()
    t[i].shape(shapes[i])
    t[i].color(clr[i])
    t[i].shapesize(dx[i],dy[i])#-----6
    t[i].showturtle()
t[3].goto(0,-85)
i=-1
while True:
    i=i+1
    t[0].goto(-400+5*i,0)
    t[1].goto(-460+5*i,-60)
    t[2].goto(-340+5*i,-60)
    time.sleep(0.02)
    if t[0].xcor()>350:
        i=-1

```



Следующий пример, который мы рассмотрим, использует два разных базовых примитива для создания анимации: квадрат и круг. Используя эти примитивы, построим движущийся автомобиль. Для построения используем два базовых примитива: квадрат и круг. В таблице 13 показан соответствующий код. Рассмотрим структуру программы. Стро-

ка, обозначенная индексом #1, определяет пустой лист базовых примитивов. Стока #2 задает форму четырех примитивов: 2 квадрата и два круга. Стока #3 задает массив, определяющий цвета примитивов. Цикл `<for i in range(4):>` заполняет пустой лист примитивами, причем строка #6 `t[i].shapesize(dx[i],dy[i])` ( $i=0,3$ ) преобразуют квадраты в прямоугольники,  $i=1$  и  $i=2$  соответствуют кругам с удвоенным размером по сравнению с исходным размером. Цикл `while True` с кодами внутри бесконечного цикла определяют движение автомобиля вдоль горизонта. Несколько статических кадров анимации показаны в правой части таблицы 13.

## Прыгающий клоун

В этом примере участвуют следующие базовые примитивы черепашьей графики: круг, квадрат и треугольник. Используя эти три примитива, а также подобрав занятную логику их движения можно создать интересную композицию. Код программы представлен в таблице 14. Представленная программа состоит из 4-ех основных блоков. Коды, расположенные между линиями 1 и 2 определяют функцию которая вводит объекты (примитивы) черепашьей графики. Каждый объект имеет следующие параметры:

`primitive` – параметр в виде строки, который может принимать названия основных базовых форм черепашьей графики(черепашка, квадрат, круг, треугольник);

`w, h` – ширина и высота объекта. Если величины `w` и `h` отличаются друг от друга, квадрат превращается в прямоуголь-

ник, круг – в овал, равносторонний треугольник – в равнобедренный, черепачка- растягивается вдоль горизонтальной и вертикальной осей;

параметр clr определяет цвет объекта;

x,y – координаты, определяющие положение объекта на координатной плоскости.

#### Таблица 14

```

import turtle,time
wn=turtle.Screen()
wn.setup(800,900,100,0)
wn.bgcolor('orange')
turtle.tracer(3)
#-----1
def object(primitive,w,h,clr,x,y):
    t=turtle.Turtle(primitive)
    t.up()
    t.shapesize(w,h)
    t.color(clr)
    t.goto(x,y)
    return t
#-----2
head=object('circle',12,12,'gold',0,0)
eye1=object('circle',2,2,'white',-50,35)
eye2=object('circle',2,2,'white',50,35)
pupil1=object('circle',1.3,0.8,'black',-50,35)
pupil2=object('circle',1.3,0.8,'black',50,35)
pupil3=object('circle',1.3,0.8,'gray',-50,35)
pupil4=object('circle',1.3,0.8,'gray',50,35)
mouth1=object('circle',2,6,'red',0,-50)
mouth2=object('circle',1.5,5,'pink',0,-50)
hat=object('triangle',10,10,'green',0,140)
nose=object('triangle',1.5,1.5,'orange',0,0)
arm1=object('square',10,2,'blue',-50,-190)
arm2=object('square',10,2,'blue',50,-190)
arm1.right(45)
arm2.left(45)
leg1=object('square',10,2,'red',-30,-300)
leg2=object('square',10,2,'red',30,-300)
body=object('circle',11,8,'blue',0,-210)
hat.left(90)
nose.rt(90)
#-----3
turtle.tracer(1)
while True:
    arm1.right(45)
    arm2.left(45)
    arm1.setposition(-50,-190)
    arm2.setposition(50,-190)
    time.sleep(0.5)
    arm1.left(45)
    arm2.right(45)
    pupil3.hideturtle()
    pupil4.showturtle()
    hat.goto(0,200)
    time.sleep(0.1)
    pupil4.hideturtle()
    pupil3.showturtle()
    hat.goto(0,140)
    time.sleep(0.1)
#-----4

```



Строки, расположенные между цифрами 2 и 3 конкретизируют все параметры объектов для создания нашего клоуна. Объектами со цветами и положением на плоскости являются, как было сказано выше, квадрат, круг и треугольник. Строки программы между цифрами 3 и 4 задают анимацию собранного клоуна: клоун мигает глазами, его шляпа подпрыгивает, он разводит руки. Все движения периодически повторяются, создавая анимационный эффект.

# **Анимация изображений, построенных из полигонов**

Для того, чтобы осуществить анимацию (движение) фигуры, рассмотренной в предыдущем параграфе, построенной из нескольких объектов, необходимо применить команды движения к каждому из составляющих фигуру объекту. Сейчас мы разберем варианты построения изображений на основе простых геометрических фигур. Такой метод позволит сформировать изображение, состоящее из нескольких геометрических фигур как один объект и применить к нему существенно меньшее количество команд движения, упростив программу. Начнем с простейшей фигуры: прямоугольника.

## **Анимация прямоугольника**

Как мы уже говорили анимация работает как серия изображений, которые немного отличаются. Чтобы создать анимацию перемещения прямоугольника вправо, нам необходимо нарисовать прямоугольник, затем обозначить его как один объект. Сделав это, мы сможем применить к прямоугольнику как к единому объекту, все команды библиотеки turtle. В таблице 15 показана простая программа, реализующая нашу идею. Поясним подробно код таблицы. Код `t.begin_poly()` между линиями 1 и 2 дает команду на создание

полигона (в нашем случае прямоугольника). К нему (прямоугольнику) затем можно будет применять все команды библиотеки, так же, как мы применяем их к простым примитивам таким как turtle, квадрат круг, треугольник и т. д. Все коды, расположенные между строками обозначенными 1 и 2, создают полигон (в нашем случае прямоугольник). Длее, полученный прямоугольник добавляется к экрану(строка 4). Код линии 3 поворачивает полученный объект-прямоугольник на 90 градусов. Теперь мы получили новый turtle-объект (линия 5), к которому применимы все команды библиотеки. Обратим внимание на то, что в этом случае объект-прямоугольник можно поворачивать (как видно из изображений, показанных в правой части таблицы) а также изменять размер полученного изображения с использованием команды t.shapesize (размер изображения). Несколько последовательных положений прямоугольника объединенных в одном кадре с использованием команды t.stamp()), показаны на изображениях правой части таблицы.

Таблица 15

```

import turtle,time
t=turtle.Turtle()
wn= turtle.Screen()
wn.setup(1200,700)

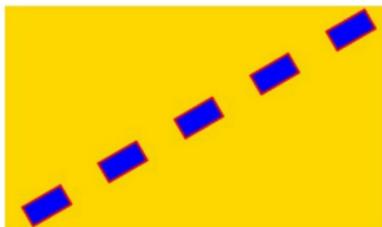
wn.bgcolor('gold')
t.penup()
t.hideturtle()
X=-600
Y=0
angle=0
#-----1
t.begin_poly()
for i in range(2):
    t.fd(100)
    t.left(90)
    t.fd(50)
    t.left(90)
t.end_poly()
m= t.get_poly() #-----2
t.tilt(90) #-----3
wn.addshape('rectangle',m) #-----4
t.shape('rectangle') #-----5
t.shapesize(outline=5)
t.color("red", "blue")
t.up()
t.hideturtle()
t.goto(X,Y)
t.left(angle)
t.showturtle()
for i in range(5):
    t.fd(200)
    t.stamp()
    time.sleep(0.01)

```

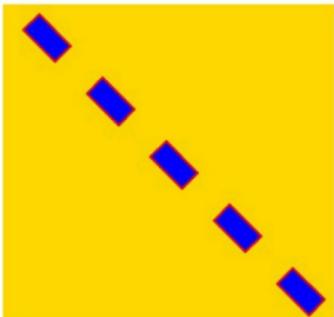
X=0 Y=0 angle=0



X=-600 Y=0 angle=30



X=-600 Y=400 angle=-45



## Пульсирующее сердце

Применим указанный прием для воспроизведения пульсирующего сердца. Соответствующий скрипт программы по-

казан в таблице 16.

## Таблица 16

```
import turtle,time
wn=turtle.Screen()
wn.setup(600,600,100,0)
turtle.tracer(1)
t=turtle.Turtle()
t.up()
t.color('red')
turtle.bgcolor('light blue')
#-----1
t.begin_poly()
t.setheading(140)
t.fd(111.65)
for i in range (200):
    t.rt(1)
    t.fd(1)
t.lt(120)
for i in range (200):
    t.rt(1)
    t.fd(1)
t.fd(111.65)
t.end_poly()
#-----2
t.hideturtle()
wn.addshape('heart',t.get_poly())
q=turtle.Turtle(shape='heart')
q.tilt(90)
q.color('red')
#-----3
while True:
    for i in range(5):
        q.shapesize(1-0.05*i)
        time.sleep(0.1)
    for i in range(5):
        q.shapesize(0.8+0.05*i)
        time.sleep(0.1)
#-----4
```



Скрипты между линиями 1 и 2 создают форму сердца из полигона с использованием основных команд черепашьей графики и базовых команд языка Python. Строки между линиями 2 и 3 определяют полученную геометрию формы сердца как объект черепашьей графики. Коды между линиями 3 и 4 определяют бесконечный цикл, в котором размер сердца периодически изменяется, то увеличиваясь, то уменьшаясь.

## Движущийся автомобиль

Используя указанный метод, можно построить более сложную геометрическую форму а затем применить к полученной форме все команды черепашьей графической библиотеки. Построим схематическое изображение формы автомобиля по заданным координатам вершин. Наш автомобиль будет состоять из следующих компонентов: кузов из двух полигонов (четырехугольник и трапеция), а также первое и второе колеса. Поясним коды таблицы 17. Команда, обозначенная линией с индексом \*, объявляет создание сложного объекта, состоящего из нескольких полигонов. Первый четырехугольник (часть кузова автомобиля) задается кодами между линиями 1 и 2. Вершины первого многоугольника обозначены переменными pos1, pos2, pos3, pos4, сам полигон определяется переменной poly1. Коды между линиями 1 и 2 Стока с индексом \*\* добавляет первый

полигон в геометрическую форму, которую мы хотим создать. Аналогично добавляется второй полигон с вершинами pos5, pos6, pos7 и pos8 (коды между линиями 2 и 3). Создав два полигона, формирующих кузов машины, добавляем первое и второе колесо с использованием кодов begin\_poly() и end\_poly() (коды между линиями 3 и 4). Создав автомобиль с использованием полигонов, определяем объект car=turtle.Turtle(shape='car') с помощью черепашьей графики (коды между линиями 4 и 5). Движение автомобиля определяется строками, обозначенными индексами #6, #7, #8, через функцию(линия \*\*\*) motion с параметром angle. Несколько статических кадров полученной анимации показаны на изображениях в правой части таблицы.

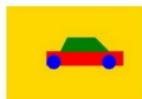
Таблица 17

```

import turtle
wn=turtle.Screen()
wn.setup(1000,700)
wn.bgcolor('gold')
import time
turtle.penup()
turtle.hideturtle()
s=turtle.Shape('compound') #-----*
#Вершины первого четырехугольника-----1
pos1=-50,0
pos2=60,0
pos3=60,20
pos4=-50,20
poly1=(pos1,pos2,pos3,pos4)
s.addcomponent(poly1,'red') #-----**
##Вершины второго четырехугольника-----2
pos5=-30,20
pos6=-20,40
pos7=20,40
pos8=40,20
poly2=(pos5,pos6,pos7,pos8)
s.addcomponent(poly2,'green')
#-----3
#Первое колесо
turtle.goto(40,-5)
turtle.begin_poly()
turtle.circle(10)
turtle.end_poly()
m1=turtle.get_poly()
s.addcomponent(m1,'blue')
#Второе колесо
turtle.goto(-40,-5)
turtle.begin_poly()
turtle.circle(10)
turtle.end_poly()
m2=turtle.get_poly()
s.addcomponent(m2,'blue')
#-----4
wn.addshape('car',s)
car=turtle.Turtle(shape='car')
car.penup()
car.tilt(90)
car.hideturtle()
car.goto(-400,200)
car.showturtle()
#-----5
def motion(angle): #----- ***
    car.setheading(angle)
    for i in range(200):
        car.fd(2)

motion(0) #-----6
motion(-90)#-----7
motion(0)#-----8

```



## Движение автомобилей вдоль холмистой дороги

В этом примере рассмотрим как создать анимацию трех автомобилей с водителями вдоль холмистой дороги. При этом используем прием прошлого примера: автомобили с водителями построены с помощью полигонов-многоугольников. Этот метод позволяет использовать почти все команды черепашьей графики, в частности повороты изображения на заданный угол а также сжатие и растяжение объекта, конечно, в добавление к основным командам графики. Полный код и несколько статических кадров показаны в таблицах 18-1 ,18-2 и 18-3) В представленном коде выделены следующие блоки: функция для построения полигонов определяется строками между линиями 1 и2; функция для построения окружностей(колеса автомобиля и глаза водителя расположена между строками 2 и 3; коды обращения к функции построения кузова автомобиля размещены между линиями 3 и 4; колеса автомобиля представлены кодами строк между линиями 4 и 5; коды, создающие водителя представлены строками между линиями 5 и 6; строки между линиями 6 и7 определяют объект автомобиль с водителем; коды между линиями 7 и 8 задают еще два автомобиля с использованием стандартной функции `clone()` черепашьей графики; строки между линиями 8 и 9 задают бесконечную анимацию- движение автомобиля; блоки между линиями 8 и 10, 10 и 11, а также между 12 и 13 определяют движение автомобилей по хол-

мам. Изображение холмистой поверхности вставлено в окно экрана с использованием функции `wn.bgpic('Street7.gif')`, где файл 'Street7.gif' , определяющий изображение должен иметь расширение gif и р должен быть расположен в том-же фолдере где основной исполняющий файл. Изображение окна экрана и код может быть скачан по адресу:

<https://github.com/victenna/Cars-are-driven-over-the-Hill>

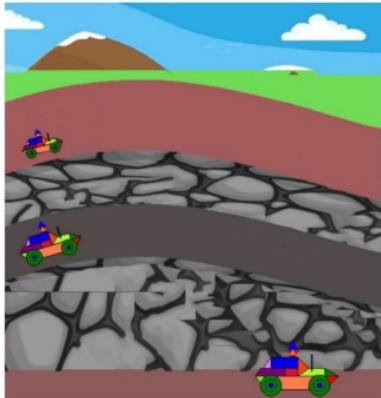
Остановимся подробнее на описании представленных функций. Функция для построения полигонов фактически строит четырехугольники и поэтому определяется четырьмя вершинами:  $x1,y1; x2,y2; x3,y3; x4,y4$ . Кроме этого функция имеет в качестве параметров цвет прямоугольника и цвет окаймляющей его линии (в нашем случае цвет окаймляющей линии для всех четырехугольников черный). Функция для построения окружностей имеет следующие параметры: координаты точки  $(x,y)$ , от которой начинаем рисовать круг, а также радиус круга  $r$ , и кроме этого цвет круга и окаймляющей его линии. Все обращения к этим функциям расположены между линиями 3 и 5. Кроме этого программа с помощью этих функций рисует и водителя.

Таблица 18-1

```

import turtle,time,random
wn=turtle.Screen()
wn.setup(1200,700,0,0)
wn.bgpic('Street7.gif') #-----*
turtle.tracer(20)
turtle.penup()
turtle.hideturtle()
s=turtle.Shape('compound')
#-----
# Функция для построения полигонов:
def polygon(x1,y1,x2,y2,x3,y3,\n
x4,y4,clr1,clr2):
    poly=((x1,y1),(x2,y2),\n
(x3,y3),(x4,y4))
    s.addcomponent(poly,clr1,clr2)
# -----
#Функция для построения окружностей:
def circles(x,y,r,clr1,clr2):
    turtle.up()
    turtle.goto(x,y-r)
    #turtle.down()
    turtle.begin_poly()
    turtle.circle(r)
    turtle.end_poly()
    m=turtle.get_poly()
    s.addcomponent(m,clr1,clr2)
#-----
# Автомобиль (кузов)
polygon(0,15,10,30,55,30,\n
55,15,'purple','black')
polygon(15,30,30,50,30,30,\n
30,30,'violet','black')
polygon(30,30,30,50,55,50,\n
55,30,'blue','black')
polygon(55,30,55,50,80,50,\n
80,30,'blue','black')
polygon(55,5,55,30,80,30,\n
80,15,'crimson','black')
polygon(80,15,80,50,100,\n
30,100,30,'orange red','black')
polygon(80,15,100,30,130,30,\n
130,15,'green yellow','black')
polygon(130,15,130,30,155,\n
15,155,15,'green','black')
polygon(130,0,130,15,155,15,\n
140,0,'red','black')
polygon(30,-10,30,15,130,15,\n
130,-10,'coral','black')
polygon(20,-10,0,15,20,15,\n
20,15,'lawn green','black')
polygon(105,25,105,55,107,55,\n
107,25,'black','black')
#-----4

```



## Таблица 18-2(продолжение Таблицы 18-1)

```

# Колеса автомобиля
#Первое колесо
circles(30,-2,20,'green','black')
circles(30,-2,10,'forest green','black')
circles(30,-2,5,'blue','black')

#Второе колесо
circles(120,-2,20,'green','black')
circles(120,-2,10,'forest green','black')
circles(120,-2,5,'blue','black')
#-----5

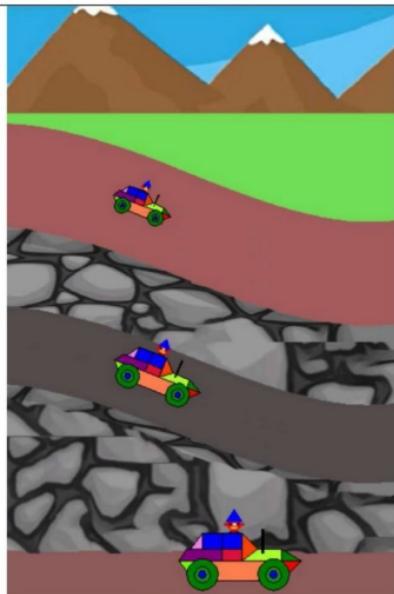
# Мальчик
polygon(62.5,50,67.5,53,72.5,\ 
53,77.5,50,'navy','black')
polygon(67.5,53,60,60,80,60,\ 
72.5,53,'red','red')
polygon(60,60,62.5,65,77.5,\ 
65,80,60,'red','red')
polygon(60,65,70,80,80,65,\ 
80,65,'blue','blue')

polygon(66,56,66,58,74,58,\ 
74,56,'gold','gold')
polygon(69,60,69,62,72,62,\ 
72,60,'light blue','light blue')
polygon(58,60,58,62,60,62,\ 
60,60,'brown','brown')
polygon(80,60,80,62,82,62,\ 
82,60,'brown','brown')

#Глаза
circles(65,62,1,'black','black')
circles(75,62,1,'black','black')
#-----6
wn.addshape('car')
car=turtle.Turtle(shape='car')
car.penup()
car.tilt(90)
car.hideturtle()
car.goto(-400,0) #Верхний Автомобиль
car.shapesize(0.4)
car.showturtle()
car.goto(-600,150)
#-----7
car1=car.clone() #Нижний Автомобиль
car1.shapesize(0.8)
car1.goto(-600,-270)

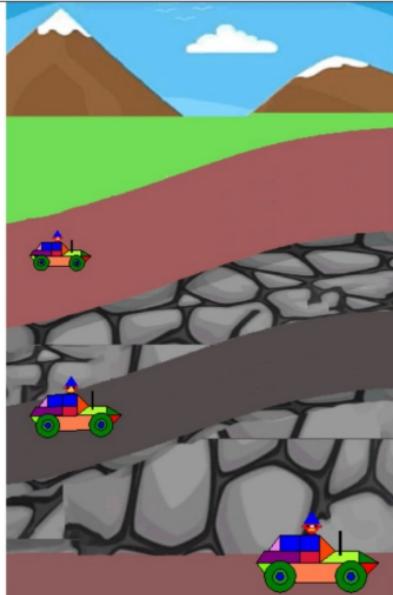
car2=car.clone() #Средний Автомобиль
car2.shapesize(0.6)
car2.goto(-600,-20)
#-----8
#Цикл задает бесконечное

```



## Таблица 18-3(продолжение Таблицы 18-2)

```
# движение автомобилей
step=1.5
#for m in range(1):
while True:
#-----9
for i in range(105):
    car.setheading(-20-0.02*i)
    car2.setheading(-20-0.02*i)
    car.fd(1)
    car2.fd(1)
    car1.fd(step)
    angle=car.heading()
    angle2=car2.heading()
#-----10
for i in range(585):
    car.setheading(angle+0.065*i)
    car.fd(1)
    car2.setheading(angle2+0.07*i)
    car2.fd(1)
    car1.fd(step)
    #car2.fd(1)
    angle=car.heading()
    angle2=car2.heading()
#-----11
for i in range(510):
    car.setheading(angle-0.08*i)
    car.fd(1)
    car2.setheading(angle2-0.09*i)
    car2.fd(1)
    car1.fd(step)
    #car2.fd(1)
    X=car.xcor()
#-----12
if X>500:
    car.goto(-600,150)
    car1.goto(-600,-270)
    car2.goto(-600,-20)
#-----13
```



## Движение автомобиля по окружности в вертикальной плоскости

Если все блоки, начиная с линии 6 и ниже заменить на те,

что представлены ниже в таблице 19, получим иной сценарий движения автомобиля (петля Нестерова или американские горки). В этом примере с помощью черепашьей графики программы Python создаем анимацию: движение автомобиля вдоль вертикальной петли. Сама петля с окружающим пейзажем введена в окно экрана файлом изображения, выбранным из интернета. Файл изображения окна следующий: Street9(4).gif. Этот файл можно скачать по адресу:

<https://github.com/victenna/Cars-are-driven-over-the-Hill>

Таблица 19

```
wn.addshape('car',s)
car=turtle.Turtle(shape='car')
car.penup()
car.tilt(90)
car.hideturtle()
car.shapesize(0.6)
car.goto(0,-250)
car.showturtle()
car.setheading(0)
turtle.tracer(50)
while True:
    for m in range(360):
        car.circle(250,1)
        time.sleep(0.008)
```



## Спиннер

В этом примере также, как и в предыдущем, строим полигоны для создания простого спиннера. Соответствующая программа показана в таблице 20. Строки между цифрами 1 и 2 определяют функцию для создания кругов, формирующих спиннер. В строках между цифрами 2 и 3 обращаемся к функции и задаем положение кругов и их цвета. ПолYGON (треугольник), являющийся основанием для спиннера, задан координатами в строках между цифрами 3 и 4. Наконец вра-

щение самого спиннера определяется командами цикла, расположеными между цифрами 4 и 5.

Таблица 20

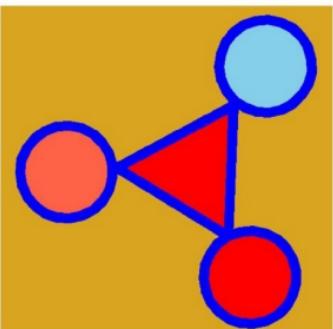
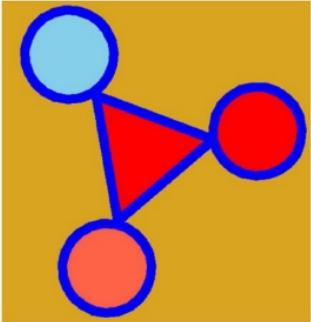
```

import turtle,time
turtle.bgcolor('goldenrod')
wn=turtle.Screen()
wn.setup(800,800,100,0)
turtle.tracer(2)
s=turtle.Shape('compound')

t1=turtle.Turtle()
t1.hideturtle()
t1.penup()
#-----1
def polygon(x,y,clr1,clr2):
    t1.goto(x,y)
    t1.begin_poly()
    t1.setheading(0)
    t1.circle(70)
    t1.end_poly()
    m1=t1.get_poly()
    s.addcomponent(m1,clr1,clr2)
#-----2
polygon(170,-170,'tomato','blue')
polygon(-170,-170,'sky blue','blue')
polygon(0,100,'red','blue')
#-----3
poly1=((0,100),(100,-70),(-100,-70))
s.addcomponent(poly1,'red','blue')

wn.addshape('spin',s)
q=turtle.Turtle(shape='spin')
q.shapesize(outline=15)
q.tilt(90)
#-----4
while True:
    q.lt(2)
    time.sleep(0.0015)
#-----5

```



## Спрайт анимация на основе библиотеки Python

## Turtle

В предыдущих разделах мы познакомились с принципами анимации, создаваемой с помощью стандартных примитивов, встроенных в библиотеку turtle. Теперь научимся создавать проекты с использованием объектов-изображений, называемых спрайтами, при этом существенно расширив возможности анимации. Объектами могут быть животные , люди, цветы, фрукты и т. д. Каждый спрайт представляет собой неподвижную картинку (изображение). Быстро сменяющая друг друга серия этих картинок составляет анимацию и называется спрайтовой анимацией. Создавая программу, мы заставляем спрайты перемещаться по экрану а также взаимодействовать друг с другом, например, направление движения одного объекта может изменяться при соприкосновении с другим объектом либо, характер движения или цвет объекта изменяется по команде от другого объекта. Действия, которые выполняет спрайт, полностью зависят от написанного кода. Один проект может иметь несколько спрайт(sprite)-изображений, каждое из которых должно быть введено в программу с использованием соответствующих файлов. Изображения можно выбрать из Интернета, либо нарисовать самому, либо воспользоваться спрайтами библиотеки популярного в настоящее время языка Scratch, предназначенного для обучения детей. Библиотека языка Scratch содержит много спрайт-изображений, каждое из которых имеет несколько костюмов. Каждый из костюмов фактиче-

ски является отдельным изображением, отличающимся от предыдущего таким образом, что используя все костюмы в совокупности с помощью кодов программы можно создавать анимацию изображений. Когда студент хорошо освоился с основами языка Scratch, в котором все команды представлены в виде блоков, появляется желание создать аналогичную программу с помощью текстового языка. Приходит на помощь язык программирования Python. Язык содержит множество различных библиотек, многие из которых используются профессиональными программистами и известными компаниями для создания серьзных программ. Одной из наиболее простых библиотек языка, приспособленной для школьников 12+ и встроенной в стандартную версию языка (ее не нужно дополнительно скачивать-инсталлировать), является библиотека Turtle. Большинство команд этой библиотеки идентичны блочным командам языка Scratch, и в этом ее преимущество перед другими библиотеками. Поэтому студенту, знакомому с языком Scratch нетрудно освоиться с основами языка Python. Однако, как мы видели в предыдущих разделах, библиотека Python- Turtle содержит лишь несколько стандартных изображений, которыми можно пользоваться для создания sprite-анимаций: квадрат, круг, стрелка, черепашка. Считается, что библиотека Turtle предназначена в основном для рисования геометрических фигур и анимаций с использованием shape-images этой библиотеки. Однако это не так. Простая и понятная для де-

тей 10+ библиотека Turtle имеет в своем составе команды, позволяющие детям создавать отличные анимационные проекты, наподобие тех, которые создаются с помощью языка Scratch. Нужно лишь научиться добывать нужные нам для проекта изображения (из интернета, либо рисовать самим), научиться вводить их в программу и контролировать движения этих изображений с помощью команд библиотеки. Итак приступим.

## **Скольжение одиночного изображения**

На первом этапе научимся передвигать единичный спрайт-изображение в заданном направлении. В качестве спрайт-изображения выберем тыквенного человечка, назовем его памкин (на английском pumpkin-тыква) . В качестве фона , на котором будет двигаться памкин, выберем пешеходный переход. Во-первых, нам надо найти и запомнить изображения памкина и фона в виде файлов в компьютере. Следует учесть, что все файлы изображений, которые используются в кодах библиотеки turtle должны иметь расширение gif. Обычно размер спрайт-изображения по горизонтали и вертикали не превышает 100-200 пикселей. Изображения фона, заполняющего экран, имеет размеры примерно 800-1200 пикселей по горизонтали и не более 750 пикселей по вертикали. Предположим, мы нашли нужное нам спрайт-изображение памкина, изображение фона и запомнили их в нужном формате(gif) в компьютере. Теперь, используя коды библиотеки turtle, создаем программу (скрипт)

которая передвигает памкина вдоль горизонта на фоне изображения-улицы.

В таблице 21 показан соответствующий скрипт.

Таблица 21

```
import turtle,time  
# Установка и выбор параметров экрана  
wn=turtle.Screen()  
wn.setup(1000,700)  
# Установка сцены-изображения экрана  
wn.bgpic('street.gif')  
# Добавление изображения спрайта к экрану  
wn.addshape('man.gif')  
# Объявление объекта man  
man=turtle.Turtle('man.gif')  
man.up()  
man.goto(-80,-290)  
#Движение объекта в горизонтальном  
#направлении слева направо  
for i in range(100):  
    man.fd(2)  
    time.sleep(0.01)
```

Image for file name: man.gif



Image for file name: street.gif



Поясним работу представленной программы. Первая строка вызывает библиотеки языка Python turtle и time, т. е. Теперь мы можем пользоваться любой командой этих библиотек. Вторая строка определяет объект Screen (обозначили его с помощью переменной wn), далее установили размер экрана (по горизонтали 1000 пикселей, по вертикали 700 пикселей) , ввели изображение фона-улицы, а

затем добавили спрайт-изображение памкина в окно экрана: файл man.gif. Затем определили объект с именем man: (man=turtle.Turtle('man.gif')). Теперь можем применять к введенному объекту man все команды библиотеки графической библиотеки turtle. Запустив программу, вы увидите как наш объект-изображение скользит по экрану.

Несколько существенных замечаний, касающихся указанной выше программы:

Файлы изображений должны располагаться в той же директории (folder), что и основной script(код) программы.

Сами изображения должны быть расположены на прозрачном фоне, иначе фон изображения наложится на основной фон экрана и исказит его.

Для того, чтобы обеспечить все приведенные требования к правильной настройке изображения, воспользуемся следующими рекомендациями: находим интересующее нас изображение в интернете, копируем его с помощью команды Ctrl+c, открываем один из редакторов, например, редактор Paint.net (предварительно бесплатно скачав-установив его на свой компьютер) и, используя команду Ctrl+v, получаем выбранное нами изображение на экране редактора. Для того, чтобы изменить размеры изображения нужно использовать опцию Image->Resize для того, чтобы сделать прозрачный фон. Затем следует запомнить полученное изображение в виде файла с расширением gif и поместить его в директорию, где расположен основной файл с кодами языка. Обра-

тите внимание на то чтобы имя файла изображения в директории совпадало с именем файла изображения в коде исполнительного файла. После запуска программы в Python редакторе получаем анимацию-движение, несколько кадров которой показано ниже.





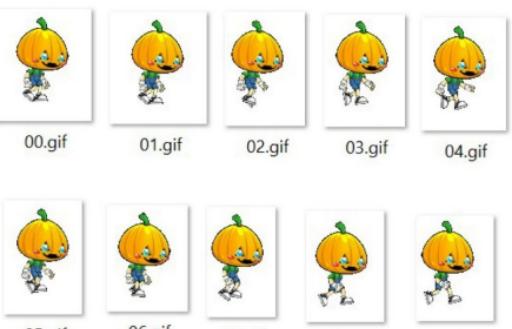


На показанных кадрах видно, что памкин-человечек скользит вдоль улицы, но не идет. Это понятно поскольку мы ввели лишь одно изображение человечка, назовем его памкин. Для того, чтобы получить настоящую анимацию в процессе движения, нам необходимо использовать несколько сменяющих друг друга изображений памкина, каждое из которых отличается от предыдущего положением ног. Если мы научимся менять эти изображения в каждом новом кадре (процессе движения), получится имитация ходьбы.

# Анимация ходьбы с несколькими изображениями

В таблице 22 представлен код, который демонстрирует как памкин переходит дорогу перебирая ногами. Код использует 10 спрайтов-изображений памкина с разным положением ног.

Таблица 22

<pre>import turtle,time wn=turtle.Screen() wn.bgpic('street.gif') image=['00.gif','01.gif','02.gif',        '03.gif','04.gif','05.gif',        '06.gif','07.gif','08.gif',        '09.gif'] man=turtle.Turtle() man.up() man.goto(-40,-275) for i in range(10):     wn.addshape(image[i]) for m in range(60):     m1=m%10     man.shape(image[m1])     man.fd(10)     time.sleep(0.2)</pre>	<p>Несколько изображений памкина:</p>  <p>00.gif      01.gif      02.gif      03.gif      04.gif 05.gif      06.gif      07.gif      08.gif      09.gif</p>
---	--

В приведенной программе 10 изображений человечка-памкина введены в программу 10-ю файлами в виде Python-list (`image=[...]`) и затем добавлены на экран с использованием команды `wn.addshape(image[i])`, где `i` пробегает значения от 0 до 9 включительно. Движение памкина осу-

ществляется с помощью цикла (for m in range(60):). Целое число m1 пробегает только 10 значений ( 0,1,2...9), в то время как величина i изменяется от 0 до 60 включительно. При этом памкин проходит весь путь от левой части экрана до конца правой. Файлы с изображениями памкина, изображение улицы а также файл-программу можно скачать по адресу:

<https://github.com/victenna/Pumpkin-man>

Если вы хотите, чтобы после первого прохода по перекрестку памкин возвращался обратно влево и вновь повторял тот же маршрут и так бесконечное количество раз, программа может быть изменена следующим образом, как показано в таблице 23 (Код показан в левой колонке):

Таблица 23

```

import turtle,time
wn=turtle.Screen()
wn.bgpic('street.gif')
image=['00.gif','01.gif','02.gif',\
       '03.gif','04.gif','05.gif',\
       '06.gif','07.gif','08.gif',\
       '09.gif']
man=turtle.Turtle()
man.up()
man.goto(-40,-275)
for i in range(10):
    wn.addshape(image[i])
m=-1
while True:
    for m in range(60):
        m1=m%10
        man.shape(image[m1])
        man.fd(10)
        time.sleep(0.2)
    if m==59:
        man.hideturtle()
        man.goto(-40,-275)
        man.showturtle()

```

```

import turtle,time
wn=turtle.Screen()
wn.bgpic('street.gif')
image=['00.gif','01.gif','02.gif',\
       '03.gif','04.gif','05.gif',\
       '06.gif','07.gif','08.gif',\
       '09.gif']
image1=['00r.gif','01r.gif','02r.gif',\
       '03r.gif','04r.gif','05r.gif',\
       '06r.gif','07r.gif','08r.gif','09r.gif']
man=turtle.Turtle()
man.up()
man.goto(-40,-275)
for i in range(10):
    wn.addshape(image[i])
    wn.addshape(image1[i])
while True:
    for m in range(60):
        m1=m%10
        man.shape(image[m1])
        man.fd(10)
        time.sleep(0.2)
    for m in range(60):
        m1=m%10
        man.shape(image1[m1])
        man.fd(-10)
        time.sleep(0.2)

```

Для того,чтобы реализовать такой алгоритм движения вводим в программу еще один цикл (while True:). Таким образом программа содержит два цикла: внутренний, который обеспечивает движение памкина слева направо и второй внешний бесконечный цикл обеспечивает повторение движение, которое обеспечивается внутренним циклом.

Дальнейшим шагом является код программы (правая колонка в таблице), которая реализует ходьбу памкина слева

направо, затем справа налево и опять слева направо и справа налево. Для того, чтобы реализовать этот код пришлось дополнительно ввести в программу 10 изображений памкина зеркально отраженных по сравнению с изображениями левой таблицы. Эти изображения «работают», когда памкин идет справа налево.

## **Особенности исполнения кодов черепахи при работе с изображениями в библиотеке**

### **turtle**

Применим к встроенной в библиотеку изображению черепашки команду `left(45)`. При этом голова черепахи повернется на 45 градусов, и последующая за ней команда `fd(100)` сдвинет черепаху на расстояние 100 пикселей под углом 45 к горизонту. В том случае, когда мы используем в программе изображение, добавленное командой `wn.addshape(image)`, повернуть изображение вокруг своей оси нельзя. Но последующая за командой `left(45)` команда `fd(100)` сдвинет изображение на 100 пикселей в направлении под углом 45 градусов к горизонту также как и в случае со стандартным изображением черепахи. Аналогичное правило действует в отношении поворота вправо. На нижнем рисунке видно: слева голова черепашки повернулась при использовании команды `left(45)`, на правом изображении памкин не повернулся на 45 градусов, однако и черепашка и памкин прошли 200 пикселей в направлении под 45 градусов к горизонту.

```
import turtle  
wn=turtle.Screen()  
t=turtle.Turtle('turtle')  
wn.bgcolor('gold')  
t.shapesize(3)  
t.color('grey')  
t.left(45)  
t.fd(200)
```



```
import turtle  
wn=turtle.Screen()  
t=turtle.Turtle('turtle')  
wn.bgcolor('gold')  
wn.addshape('man.gif')  
t.shape('man.gif')  
t.left(45)  
t.fd(200)
```



Отметим также, что команда `shapesize(...)` в случае с изображениями, взятыми, например, из интернета, не работают. Если необходимо увеличить размер изображения или повернуть его на какой-либо угол, необходимо вначале создать необходимое вам изображение с использованием, например библиотеки PIL языка Python а затем ввести это изображение в программу.

Видео, демонстрирующее пешехода памкина при использовании кода в правой части показанной выше таблицы, расположено по адресу:

<https://youtu.be/K3OLvrmDTHI>

## Астронавт в открытом космосе

Сценарий этого анимационного проекта следующий: астронавт летит в открытом космосе вблизи космического корабля. Код и пояснения приведены в таблице 24

## Таблица 24

```
import turtle  
import random  
turtle.tracer(4)  
t=turtle.Turtle()  
wn=turtle.Screen()  
wn.setup(900,700)  
import time  
#Moonsky  
wn.bgpic('moonsky.gif')  
#Astronaut  
wn.addshape('astr.gif')  
Astro=turtle.Turtle('astr.gif')  
Astro.up()  
#Spaseshuttle  
wn.addshape('shuttle.gif')  
Shuttle=turtle.Turtle('shuttle.gif')  
Shuttle.penup()  
Shuttle.goto(-600,-600)  
Shuttle.setheading(40)
```

```
#-----1  
while True:  
    Shuttle.fd(5)  
    X,Y=Shuttle.position()  
    Astro.setposition(X+15,Y+115)  
    Astro.fd(30)  
    Astro.right(5)  
    time.sleep(0.03)  
    if abs(X-600)<10 or abs(Y-600) <10:  
        Shuttle.goto(-600,-600)  
        Astro.goto(-600,-400)  
#-----2  
Shuttle.setheading(random.randint(40,80))
```



В показанном примере изображение окна экрана (звездного неба) а также астронавта и космического корабля вводится в программу с помощью модуля turtle.

дятся в программу с помощью трех файлов: moonsky.gif, astr.gif и shuttle.gif. Движение корабля и космонавта определяются кодами между линиями 1 и 2, каждое новое вхождение космического корабля в зону окна задается случайным углом наклона (последняя строка кода). Для того, чтобы астронавт не удалялся от корабля, в процессе работы программы определяются координаты корабля X, Y, а затем позиция космонавта определяется строкой кода Astro.setposition(X +15,Y+115). Файлы с необходимыми изображениями а также файл с программой можно скачать по адресу:

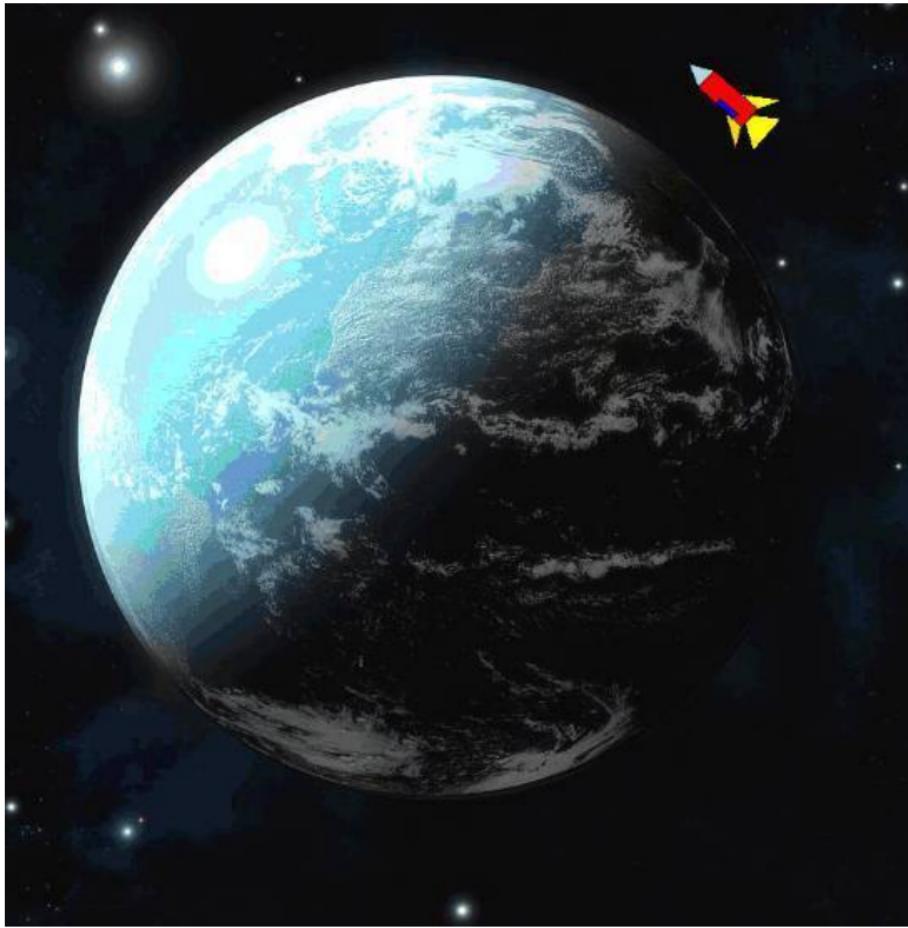
<https://github.com/victenna/Astronaut-in-free-space>

Видео файл с полученной анимацией можно посмотреть по адресу:

<https://youtu.be/vbO7jR3zMho>

## **Ракета вращается вокруг Земли**

Следующий проект: ракетоплан вращается вокруг Земли. Особенностью создания программы является то, что ракетоплан строим сами, используя метод полигонов, изображение Землю выбираем из интернета. Ниже на рисунке показана статическая картина того, что мы хотим реализовать с помощью команд черепашьей графики и использованием стандартных команд программы Питон.



В таблице 25 показан скрипт программы:  
Таблица 25

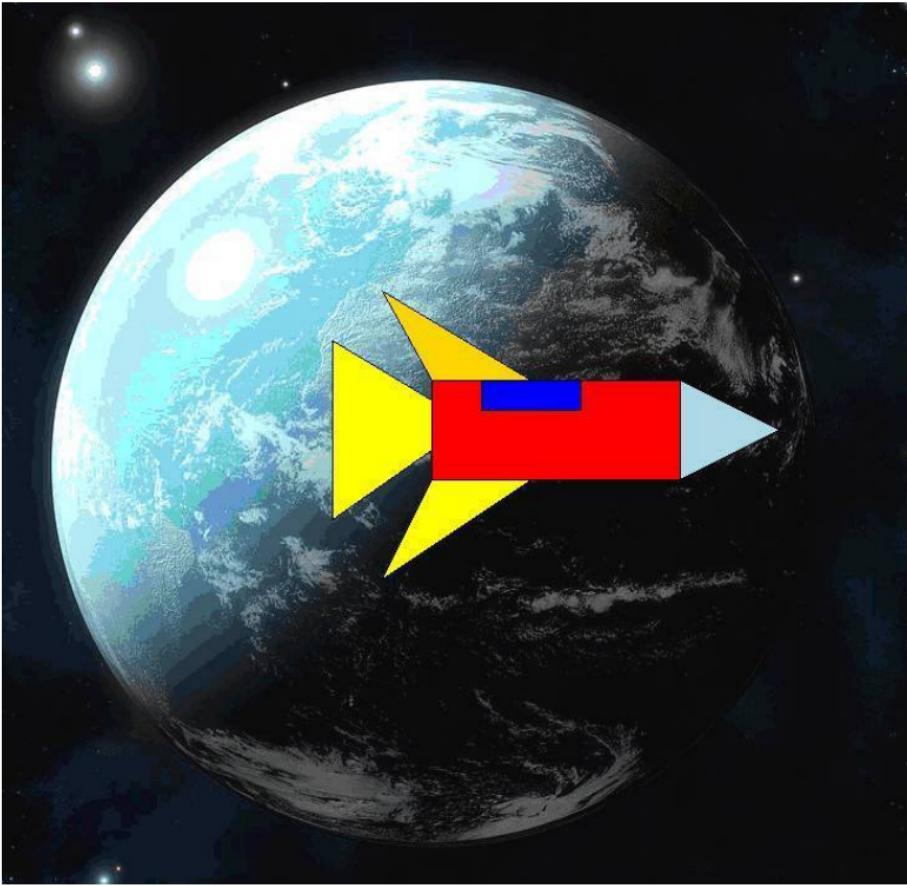
```
import turtle,time
wn=turtle.Screen()
wn.bgcolor('navy')
wn.setup(900,900)
image1='Earth.gif'
wn.addshape(image1)
earth=turtle.Turtle()
earth.shape(image1)
earth.goto(0,0)
turtle.tracer(2)
s1=turtle.Shape("compound")
poly1=((0,0),(25,0),(25,10),(0,10))----#1
poly2=((25,0),(35,5),(25,10))----#2
poly3=((0,10),(-5,19),(10,10))----#3
poly4=((0,0),(-5,-10),(10,0))----#4
poly5=((5,7),(15,7),(15,10),(5,10))----#5
poly6=((0,3),(-10,-4),(-10,14),(0,8))----#6
clr=['red','light blue','gold','yellow','blue']
s1.addcomponent(poly1,clr[0],'black')
s1.addcomponent(poly2,clr[1],'black')
s1.addcomponent(poly3,clr[2],'black')
s1.addcomponent(poly4,clr[3],'black')
s1.addcomponent(poly5,clr[4],'black')
s1.addcomponent(poly6,clr[3],'black')
wn.addshape('geometry',s1)
```

```
Rocket=turtle.Turtle(shape='geometry')
Rocket.shapesize(2)
Rocket.tilt(90)
Rocket.showturtle()
Rocket.up()
Rocket.goto(0,-400)
Rocket.rt(0)
turtle.tracer(2)
while True:
    Rocket.circle(400,1)
    time.sleep(0.01)
```



В отличие от предыдущего примера в представленном коде ракетоплан создается с использованием 6 полигонов с вершинами, обозначенными в строках #1-#6. Каждый из по-

лиголов окрашен в свой цвет, а все вместе образуют ракетоплан, который вращается вокруг земли и подчиняется всем командам черепашьей графики. Применяя к построенному ракетоплану функции поворота(left(),right()), используемые в черепашьей графике, создаем эффект его вращения вокруг Земли. Ниже на фоне Земли показан ракетоплан, построенный с помощью полигонов кода.



Файл с изображением Земли, а также файл с программой можно скачать по адресу:

<https://github.com/victenna/Rockets-around-Earth>

### **Планеты солнечной системы**

Представленная в таблице 26 программа реализует ани-

мацию солнечной системы с использованием планет-изображений-спрайтов, скачанных из интернета, вращающихся вокруг солнца. Коротко об отдельных блоках программы. Коды с файлами между линиями 1 и 2 вводят в программу изображения планет солнечной системы. Все файлы имеют расширение gif. Между линиями 2 и 3 расположены коды, заполняющие список, в который входят указанные файлы планет. В этом же блоке коды расставляют планеты в начальное координатное положение. Наконец, коды между линиями 3 и 4 задают вращение планет вокруг солнца а также луны вокруг земли.

## Таблица 26

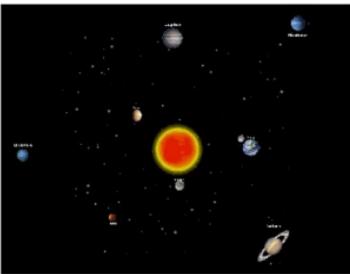
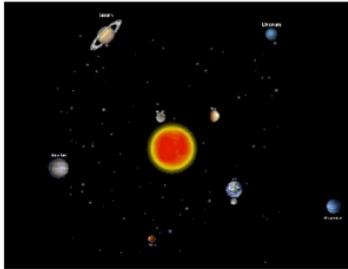
```

#Solar Planet Motion
import turtle
wn=turtle.Screen()
wn.setup(1300,1300,400,0)
wn.bgpic('sky.gif')
wn.bgcolor('black')
wn.tracer(20)
#-----1
Img=['Sun6.gif','mercury.gif',\
'venus1.gif','earth4.gif',\
'Mars2.gif','Jupiter2.gif',\
'Saturn4.gif','Urans2.gif',\
'Neptune2.gif','Moon.gif']
#-----2
planet=[]
turtle.hideturtle()

for n in range(10):
    wn.addshape(Img[n])
    planet.append(turtle.Turtle(Img[n]))
    planet[n].up()
    if n==0:
        planet[0].setposition(0,0)
    elif n==9:
        planet[n].goto(0,-310)
    else:
        planet[n].goto(0,-120-70*(n-1))
#-----3
while True:
    for m in range (1,10):
        planet[m].circle(120+70*(m-1),0.8-
0.1*(m-1))
        planet[9].hideturtle()

    planet[9].goto(planet[3].xcor(),planet[3].ycor())
    planet[9].fd(45)
    planet[9].showturtle()
    planet[9].left(0.1)
#-----4

```



## Полет воздушных шариков

Программа реализует следующий анимационный сцена-

рий. Мальчик, пританцовывая, выпускает из рук шарик, который поднимается вверх, постепенно увеличиваясь в объеме, и долетев до верхней части экрана лопается. Затем все действие повторяется вновь и вновь, создавая анимационное видео. Несколько кадров анимации представлено в правой части таблиц 27-1 и 27-2.

Таблица 27-1

```

import turtle,time,random
wn=turtle.Screen()
turtle.tracer(3)
wn.setup(1200,800)
wn.bgcolor('light blue')
wn.bgpic('background2.gif')#-----1
#-----2
clr=['red','blue','yellow','lime','gold']
clr1=['shredred.gif','shredblue.gif',\
      'shredyellow.gif',\
      'shredlime.gif','shredgold.gif']
#-----3
#Пять изображений мальчика
boyimage=['boys1.gif','boys2.gif','boys3.gif',\
          'boys4.gif','boys5.gif']
for m in range(5):
    wn.addshape(boyimage[m])
#-----4
for i in range(5):
    wn.addshape(clr1[i])
    wn.addshape('rope1.gif')
#-----5
def object(img,X0,Y0):
    t=turtle.Turtle()
    t.hideturtle()
    t.shape(img)
    t.up()
    t.goto(X0,Y0)
    t.showturtle()#!!!!!!!!!!!
    return t
#-----6
boy=object(boyimage[0],-40,-200)
balloon=object('circle',-11,-120)
cord=object('rope1.gif',-10,-155)
shred=object('shredred.gif',-40,200)
#-----7
balloon.shapesize(2.2,3)
balloon.color(clr[0])
balloon.setheading(90)
#-----8
def ball_size(ball,width,height,X,Y):
    ball.shapesize(width,height)
    ball.goto(X,Y)
#-----8

```



## Таблица 27-2(продолжение таблицы 27-1)

```
q=-1
while True:
    balloon.showturtle()#!!!!!!!
    ball_size(balloon,2.2,3,-11,-120)
    q=q+1
    q1=q%5
    balloon.color(clr[q1])
    shred.hideturtle()#-----ОСКОЛКИ
    cord.hideturtle()
    cord.goto(-10,-160)#веревка
    cord.showturtle()
#-----
for i in range(25):
    X=balloon.xcor()
    Y=balloon.ycor()
    cord.setposition(X,Y-20-2.5*i)
    balloon.fd(14)

balloon.shapesize(2.2*(1+0.06*i),3*(1+0.06*i))
time.sleep(0.05)
i1=i%5
i1=round(i/5)
i2=i1%5
boy.shape(boyimage[i2])#-----
if i==24:
    cord.hideturtle()
    balloon.hideturtle()
    shred.showturtle()
    shred.shape(clr1[q1])
    time.sleep(0.5)
#-----
```



Поясним основные ключевые блоки программы. Картинка на экране, на фоне которой мальчик запускает вверх шарик, определяется файлом строки с номером 1. Между стро-

ками 2 и 3 расположены коды, задающие списки с цветами шариков а также изображения лоскутов лопнувших шариков. Между линиями 3 и 4 расположены коды, определяющие, танцующего мальчика. Коды между линиями 4 и 5 вводят в программу 5 лоскутов лопнувших шариков и изображение нитки, на которой држится шарик. Очень важными являются коды между линиями 5 и 6. Эти коды служат для определения функции, которая вводит следующие объекты: мальчик, шарик, лоскуты лопнувшего шарика и нить, на которой держится шарик. Все 5 объектов заданы кодами между линиями 6 и 7. Наконец, коды между линиями 8 и 9 определяют движение шарика вверх а также анимацию движений мальчика с помощью соответствующих спрайтов. Файлы изображений можно скачать по адресу:

<https://github.com/victenna/Balloon>

## **Уличное движение**

Пример демонстрирует использование нескольких спрайтов –изображений для создания анимации движения автомобилей вдоль улицы. При этом окно экрана-сцена также является движущимся изображением, которое усиливает анимационный эффект. В таблице 28 приведен соответствующий код программы

Таблица 28

```

import turtle
import time
wn=turtle.Screen()
wn.setup(1000,500)
turtle.tracer(4)
#-----1
def object(img,X,Y):
    wn.addshape(img)
    t=turtle.Turtle(img)
    t.up()
    t.goto(X,Y)
    return t
#-----2
street=object('street.gif',-500,0)
car1=object('car_1.gif',600,-170)
car2=object('car_2.gif',600,-55)
car3=object('car_2.gif',600,-98)
#-----3
while True:
    wn.update()
    street.fd(2)
    car1.fd(-2)
    car2.fd(-1)
    car3.fd(-1.5)

    if car1.xcor()<-500:
        car1.goto(600,-170)
    if car2.xcor()<-510:
        car2.goto(600,-55)
    if car3.xcor()<-510:
        car3.goto(600,-98)
    if street.xcor()>490:
        street.goto(-490,0)
    time.sleep(0.01)
#-----4

```



Поясним работу программы. Скрипты функции `object`, расположенные между линиями 1 и 2, описывают изображения, которые затем вводятся в программу с помощью кодов между линиями 2 и 3. Функция `object` имеет следую-

щие параметры: файл изображения и координаты изображения на координатной плоскости экрана. В качестве объектов используются: изображение улицы (файл с изображением улицы 'street.gif', -500,0; файл с изображением первой машины 'car\_1.gif'; файл с изображением второй машины 'car\_2.gif'; файл с изображением третьей машины 'car\_2.gif'). Все изображения имеют свои начальные координаты, которые задаются параметрами функции X, Y. Движение автомобилей и перемещение улицы определяется скриптами, расположенными между строками 3 и 4. С правой стороны в таблице показаны несколько кадров полученной анимации. Файлы изображений, используемые в программе могут быть скачаны с сайта:

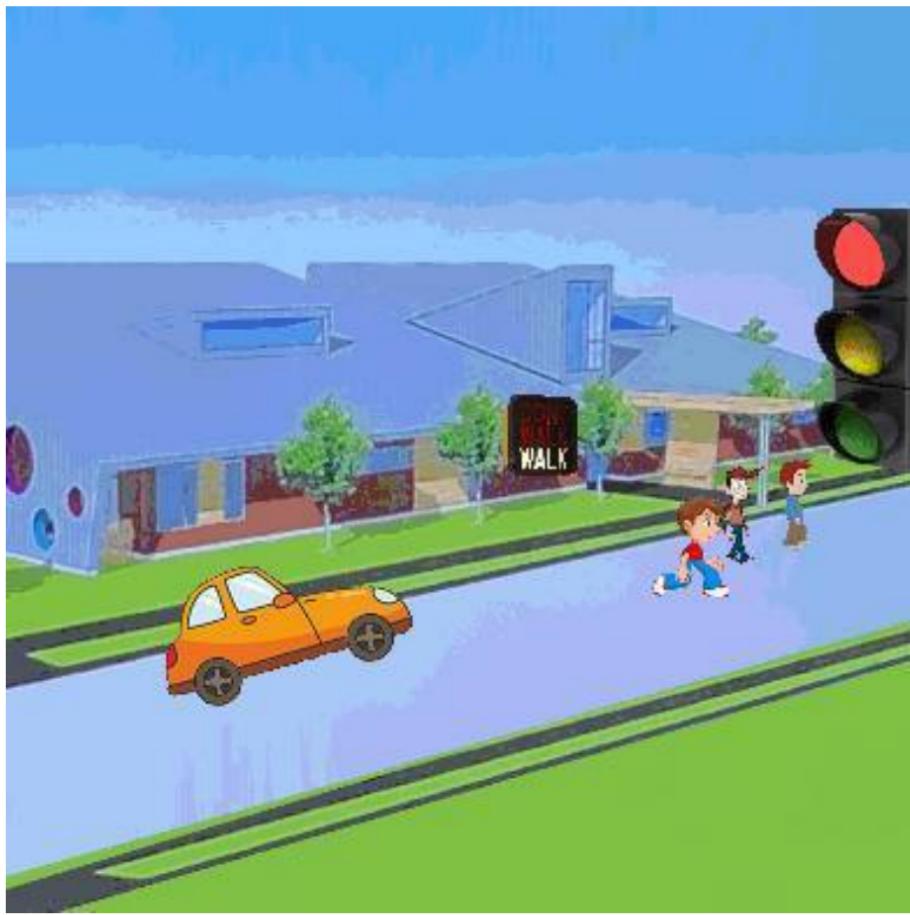
<https://github.com/victenna/Cars-on-a-road>

## **Пешеходный переход**

В демонстрируемом проекте участвуют несколько объектов: три пешехода (каждый пешеход имеет 6 спрайт-изображений, отличающихся положением ног, для имитации движения); автомобиль(один спрайт); светофор с тремя спрайтами для создания переключающегося света (красный, желтый и зеленый); знак, обозначающий для пешехода необходимость переходить улицу либо стоять, когда автомобиль движется по улице (два спрайта).

Несколько кадров сценария, реализованного с помощью этой программы, приведено ниже





Полный скрипт программы показан ниже в таблицах 29-1 и 29-2

Таблица 29-1

```

import turtle,time
wn=turtle.Screen()
wn.setup(800,800)
turtle.bgcolor('light blue')
wn.bgpic('street.gif')
turtle.tracer(2)
#-----1
cars=['car2.gif','car3.gif','car4.gif','car2.gif','car3.gif','car4.gif']
image1=['man1.gif','man2.gif','man3.gif','man4.gif','man5.gif','man6.gif']
image2=['boy21.gif','boy22.gif','boy23.gif','boy24.gif','boy25.gif','boy26.gif']
image3=['boy41.gif','boy42.gif','boy43.gif','boy44.gif','boy45.gif','boy46.gif']
sign=['w_no.gif','w_yes.gif']
light=['red_light.gif','yellow_light.gif','green_light.gif']
#-----2
for i in range(6):
    wn.addshape(image1[i])
    wn.addshape(image2[i])
    wn.addshape(image3[i])
    wn.addshape(cars[i])
    if i<2:
        wn.addshape(sign[i])
    if i<3:
        wn.addshape(light[i])
#-----3
def object(X,Y):
    t=turtle.Turtle()
    t.up()
    t.goto(X,Y)
    return t
#-----4
man=object(180,-20)
man.shape(image1[0])
boy1=object(230,-10)
boy1.shape(image2[0])
boy2=object(130,-40)
boy2.shape(image3[0])
car=object(-400,-210)
car.shape(cars[0])
car.setheading(12)
turtle.tracer(1)
lightt=object(350,100)
lightt.shape(light[2])
signn=object(70,20)
signn.shape(sign[1])
#-----5
deltaX=5 #determines walking direction
deltaY=-2.5 #determines walking direction
q=-1
move=1
r=0
while True:#-----6
    q=q+1
    q0=q%3

```

## Таблица 29-2(продолжение Таблицы 29-1)

```

time.sleep(0.03)
if car.xcor()<-151:
    car.shape(cars[q0])
    move=1
    car.fd(10*move)
    lightt.shape(light[2])
    signn.shape(sign[0])
if car.xcor== -152:
    car.shape(car[0])
if car.xcor()>-152:
    if r<30:
        car.shape(cars[0])
        r=r+1
    lightt.shape(light[1])
    signn.shape(sign[0])
if r>=29 and r<81:
    r=r+1
    q1=q%6
    lightt.shape(light[0])
    signn.shape(sign[1])
    r1=r-29
    man.shape(image1[q1])
    man.goto(180+r1*deltaX,-20+r1*deltaY)
    boy1.shape(image2[q1])
    boy1.goto(230+r1*deltaX,-10+r1*deltaY)
    boy2.shape(image3[q1])
    boy2.goto(130+r1*1.2*deltaX,-40+r1*1.2*deltaY)
    time.sleep(0.05)
if r>=81:
    car.shape(cars[q0])
    car.fd(10)
    lightt.shape(light[2])
    signn.shape(sign[0])
if car.xcor()>500: #-----7
    q=-1
    man.hideturtle()
    man.shape(image1[0])
    man.goto(180,-20)
    man.showturtle()
    boy1.shape(image2[0])
    boy1.hideturtle()
    boy1.goto(230,-10)
    boy1.showturtle()
    boy2.shape(image3[0])
    boy2.hideturtle()
    boy2.goto(130,-40)
    boy2.showturtle()
    r=0
    car.hideturtle()
    car.setposition((-400,-210))
    car.showturtle() #-----8

```

Работа блоков, входящих в состав программы, следующая:

Коды между блоками 1 и 2 определяют списки изображений, которые используются в программе;

Коды между блоками 2 и 3 добавляют к экрану указанные изображения;

Коды между линиями 3 и 4, а также между линиями 4 и 5 определяют (с использованием библиотеки черепашьей графики) необходимые объекты, которые затем используются в программе;

Коды между линиями 6 и 7 задают цикл, с помощью которого осуществляется движение автомобиля, переключение светофора, движение пешеходов, а также переключение знака, разрешающего пешеходам двигаться. Все движения объектов подчиняются логике правильного движения на пешеходном перекрестке. Коды между линиями 7 и 8 задают логику повтора движения пешеходов при условии, что они(пешеходы) уходят за правый край экрана. При этом сценарий анимации движения повторяется.





Файлы изображений, используемые в этой программе могут быть скачаны с сайта:

<https://github.com/victenna/Pedestrian-Crossing>

Вideo файл с полученной анимацией можно посмотреть по адресу:

<https://youtu.be/zvTHDusc-H0>

# Особенности анимации с большим количеством спрайт-файлов

В примерах, которые показаны ниже, используется большое количество спрайтов изображений. Все статические спрайты получены из анимационных файлов с расширением gif. Каждый такой файл может содержать десятки а иногда и сотни статических кадров, и конечно вводить в программу каждый статический файл крайне неудобно. Для ускорения ввода таких файлов в программу Python можно воспользоваться специальной программой, преобразующей gif анимационный файл в набор статических перенумерованных в определенном порядке файлов, каждый из которых имеет такое же расширение gif , пригодное для использования в основной программе. Две возможные программы для ковертирования анимационного изображения в набор статических файлов показаны в таблице 30. Для того, чтобы воспользоваться показанными программами необходимо проинсталлировать на компьютер в среду Python библиотеку PIL, которая имеется в свободном интернете. С левой стороны показана программа, в которой не нужно задавать количество кадров gif файла, с правой стороны – программа, в которой необходимо задать количество кадров образуемого файла. Т.е., в первом случае количество кадров-изобра-

жений(frames) определяется исходным файлом, во втором случае этим количеством можно варьировать. Как в левой так и в правой программе строка с пометкой ! задает исходный анимационный файл с расширением gif, строки с пометкой !! определяют статические gif файлы.

Таблица 30

from PIL import Image, ImageSequence  im = Image.open('nice swimming.gif')#----! index = 1 for frame in ImageSequence.Iterator(im): frame.save("swim%d.gif" % index)#----!! index += 1	from PIL import Image  num_key_frames = 13  with Image.open('shuttle1.gif') as im:#-----! for i in range(num_key_frames): im.seek(im.n_frames // num_key_frames * i) im.save('{}.gif'.format(i))#-----!!
--	---

Переидем теперь к рассмотрению конкретных примеров.

## Летающий орел

Сценарий представленной ниже анимации заключается в том, что орел взмахивая крыльями пролетает через весь экран справа налево и исчезая появляется вновь с правой стороны. Первоначально в Интернете был найден анимационный файл орла, который, как оказалось имеет 21 статический кадр. Кадры отличаются друг от друга положением крыльев орла. Используя показанную выше программу(слева), мы преобразовали указанный файл в статические перенумерованные файлы по количеству, равному 21 и разместили

их в фолдере(папке) вместе с основным файлом Python программы. В таблице 31 показаны коды полученной программы. Коды между строками 1 и 2 вводят в программу с помощью списка статические изображения орла. Коды между линиями 3 и 5 задают основной цикл движения орла, коды между линиями 4 и 5 дают возможность перебирать статические изображения орла, создавая эффект анимации со взмахами крыльев. В правой части таблицы показаны два статических кадра полученной анимации.

Таблица 31

```

import turtle,time,random
wn=turtle.Screen()
wn.setup(1200,700)
wn.bgpic('orel_background.gif')
turtle.tracer(5)
#-----1
eagles=[0]
for i in range(1,22):
    i1=str(i)
    eagles.append('eagle'+i1+'.gif')
#-----2
orel=turtle.Turtle()
orel.up()
orel.goto(600,0)
#-----3
while True:
    orel.setheading(random.randint(-15,15))
    if orel.xcor()<-600:
        orel.goto(600,0)
#-----4
for i in range(1,22):
    wn.addshape(eagles[i])
    orel.shape(eagles[i])
    time.sleep(0.1)
    orel.fd(-10)
#-----5

```



Файлы изображений, используемые в этой программе могут быть скачаны с сайта:

<https://github.com/victenna/Eagle-Motion>

## Aquarium

Еще один пример анимации с большим количеством статических спрайт-кадров показан в таблице 32: аквариум с водой и рыбками. Вода в аквариуме во время анимации находится в постоянном движении. Для создания этой анимации

ции было использовано 74 статических кадров воды, которые получены из gif файла, с последующей раскадровкой с использованием программы превращения gif анимационного файла.

Коротко о структуре программы. Статические файлы с изображениями воды записаны в список AQ[]. Этот список первоначально пуст, т. е. в нем нет ни одного файла. Элементами списка становятся файлы с расширением gif, которыми пополняется список с помощью кодов, расположенных между линиями 1 и 2. Названиями файлов являются цифры, начиная от 0 и кончая цифрой 73, всего 74 файла. Все файлы, как было указано, имеют расширение gif. Файл с движущейся водой первоначально был найден в Интернете в виде анимационного gif файла. Затем с помощью программы, представленной в Таблице 30, он был преобразован в набор из 74 файлов, каждый из которых также имеет расширение gif. Между линиями 2 и 3 расположены файлы с изображением рыбы, плавающей в аквариуме. Основные коды, определяющие движение воды и рыбы расположены между линиями 4 и 5. Файлы изображений, используемые в этой программе могут быть скачаны с сайта:

<https://github.com/victenna/Aquarium-with-water-waves>

Видео с результатами программы можно посмотреть по адресу

<https://youtu.be/XeTwY9bwFLk>

Таблица 32

```

import turtle,time,random
wn=turtle.Screen()
wn.setup(850,600)
AQ=[]           #-----1
for i in range(74):
    i1=str(i)
    AQ.append(i1+'.gif')      #-----2
image11 = ("fi11.gif")
image12 = ("fi12.gif")
image31 = ("fi31.gif")
image32 = ("fi32.gif")
wn.addshape(image11)
wn.addshape(image12)
wn.addshape(image31)
wn.addshape(image32)
t=turtle.Turtle()    #-----3
t.showturtle()
t.speed(10)
t.penup()
X,Y=90,-90
dx,dy=2,4
k,r=0,0            #-----4
while True:
    r=r+1
    k=k+1
    k1=k%74
    r1=r%100
    wn.bgpic(AQ[k1])
    t.setposition(X+dx,Y+dy)
    X,Y=t.position()
    if dx>0 and r1<50:
        t.shape(image11)
    if dx>0 and r1>50:
        t.shape(image12)
    if dx<0 and r1<50:
        t.shape(image31)
    if dx<0 and r1>50:
        t.shape(image32)
    if X>250 or X<-260:
        dx=-dx
    if Y<-205 or Y>210:
        dy=-dy      #-----5

```



# **Анимация с управлением от кнопки компьютерной мыши**

## **Порхающая бабочка с машущими крыльями**

Код и несколько статических кадров предлагаемой анимации представлены в Таблице 33.

Таблица 33

```

import turtle
wn=turtle.Screen()
import time
wn.setup(900,700)
wn.bgcolor('pink')
wn.bgpic('grass.gif')
#-----1
flowers=['flo0.gif','flo1.gif',\
         'flo2.gif','flo3.gif',\
         'flo4.gif','flo5.gif']
#-----2
flower=[]
for n in range (6):
    wn.addshape(flowers[n])
    flower.append(turtle.Turtle(flowers[n]))
    flower[n].up()

flower[0].goto(-100,-230)
flower[1].goto(-300,-120)
flower[2].goto(300,-170)
flower[3].goto(0,235)
flower[4].goto(120,50)
flower[5].goto(-250,150)
#-----3
butterfly=[]
for i in range (18):
    i1=str(i)
    butterfly.append(i1+'.gif')
#-----4
t=turtle.Turtle()
t.up()
#-----5
while True:
    for i in range (18):
        wn.addshape(butterfly[i])
        t.shape(butterfly[i])
        time.sleep(0.1)
    def fly(x, y):
        t.goto(x, y)
        wn.onclick(fly)
#-----6

```



Для создания этой анимации мы используем управляющие кнопки компьютерной мыши. В нашем сценарии бабочка летает от одного места экрана к другому по клику левой кнопки компьютерной мышки. Наведя указатель мышки на желаемую точку экрана и кликнув кнопкой мышки, мы увидим как бабочка, помахивая крыльями перелетает в указанную нами точку. Управление анимацией с помощью кликов кнопки мышки возможно при использовании функции onclick() черепашьей библиотеки языка Питон. Рассмотрим более подробно коды программы. Строки между номерами 1 и 2 задают список файлов изображений 6-и цветков, которые будут размещены в окне экрана компьютера. Между цифрами 2 и 3 находятся коды, которые вводят изображения цветков в программу и устанавливают их в определенные координатные позиции на экране. Коды между номерами 3 и 4 вводят в программу с помощью списка файлы с изображением бабочки с порхающими крыльями. Таких файлов 18. Они отличаются друг от друга положением крыльев, создавая имитацию полета со взмахиванием крыльев. Объект бабочки названа именем t и в бесконечном цикле while True по команде onclick() перелетает в желаемую точку. Желаемая точка на экране выбирается указателем мышки с последующим кликом левой кнопкой компьютерной мышки. 18 файлов бабочки с машущими крыльями было выбрано тем же методом, как и в примере, описанном в Таблице 32.

Файлы изображений, используемые в этой программе мо-

гут быть скачаны с сайта:

<https://github.com/victenna/Flowers-and-Butterfly>

Вideo файл с полученной анимацией можно посмотреть  
по адресу:

[https://youtu.be/tDESlsOs\\_0Y](https://youtu.be/tDESlsOs_0Y)

## **28. Вертолет с парашютистами**

Следующей анимацией, которую мы здесь покажем является полет вертолета с парашютистами. Во время полета по щелчку левой кнопки мышки с вертолета спускается парашютист. Программа также, как и предыдущая, использует функцию черепашьей библиотеки для управления кнопками компьютерной мышки onclick(). В таблицах 34-1 и 34-2 показан скрипт программы, с правой стороны показано несколько отдельных кадров анимации.

Таблица 34-1

```

import turtle,time
wn=turtle.Screen()
wn.setup(1300,700)
wn.bgpic('road.gif')
t=turtle.Turtle()
t.up()
turtle.tracer(2)
#-----1
image1=['helicopter11.gif','helicopter12.gif']
image2=['helicopter21.gif','helicopter22.gif']
for i in range(2):
    wn.addshape(image1[i])
    wn.addshape(image2[i])
driver=[]
image3=['boy.gif','boy1.gif','boy2.gif',
'boy3.gif','girl.gif']
#-----2
def condition(object):
    object.setheading(-90)
    if object.ycor()>10:
        object.fd(5)
    if object.ycor()<-10:
        object.fd(0)
#-----3
for i in range(len(image3)):
    wn.addshape(image3[i])
    driver.append(turtle.Turtle())
    driver[i].shape(image3[i])
    driver[i].up()
    driver[i].hideturtle()
    driver[i].setheading(-90)
#-----4
t.setheading(90)
def up_down(step):
    for j in range (100):
        j1=j%2
        t.shape(image1[j1])
        t.fd(step)
        time.sleep(0.015)
#-----5
t.up()
t.hideturtle()
t.goto(-500,0)

```



## Таблица 34-2(продолжение Таблицы 34-1)

```
t.showturtle()
up_down(2.5)
t.setheading(0)

q=0
q1=1
q2=0
#-----6
while True:
    i=i+1
    i1=i%2
    if q1==1:
        t.shape(image1[i1])
    if q1==-1:
        t.shape(image2[i1])
    t.fd(q1*5)
    X=t.xcor()
    time.sleep(0.02)
    if X>530:
        q1=q1*-1
    if X<-530:
        q1=q1*-1

    condition(driver[q2])
#-----7
def h(x,y):
    global q
    global q2
    global X

    q=q+1
    q2=q%len(image3)
    driver[q2].goto(X,200)
    driver[q2].showturtle()

    wn.onclick(h)
#-----8
```



Коротко обсудим основные блоки представленной программы. Коды, расположенные между линиями 1 и 2 вводят в программу файлы изображений вертолета и 5-и парашютистов. Функция между линиями 2 и 3 определяет движение парашютистов от вертолета к земле. Обращение к функции производится в основном цикле, коды которого расположены между линиями 6 и 8. Между линиями 3 и 4 расположены коды, которые заполняют список файлами спрайтов парашютистов. Между линиями 4 и 5 находятся коды, которые, с использованием функции `<up_down(step):>` определяют подъем вертолета. Основной цикл, задающий движение вертолета вдоль горизонта, определяется кодами между линиями 6 и 8. Внутри этого цикла имеется функция `< def h(x,y): >`, которая по щелчку левой кнопки мышки показывает анимацию парашютиста спускающегося на землю.

Файлы изображений, используемые в этой программе могут быть скачаны с сайта:

<https://github.com/victenna/Helicopter>

Видео файл с полученной анимацией можно посмотреть по адресу:

[https://youtu.be/5P104dh\\_1cU](https://youtu.be/5P104dh_1cU)

# **Заключение**

Мы надеемся, что вы получили удовольствие, прочитав эту книгу и выполнив большинство из предложенных нами проектов. Основные базовые функции и команды, рассмотренные в наших примерах, характерны для любой библиотеки языка Python, не только черепашьей библиотеки. Освоив логику построения предложенных нами команд, вы сможете быстрее освоить более сложные библиотеки многих современных языков а также более эффективно освоить школьную математику.