|  |
| --- |
| МУНИЦИПАЛЬНОЕ АВТОНОМНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  ГОРОДА ХАБАРОВСКА «ЛИЦЕЙ ИННОВАЦИОННЫХ ТЕХНОЛОГИЙ» |
| **Н.Н. Гончаренко, Е.В. Редько**  **Графика на языке Python:**  **учебное пособие** |
|  |
| Хабаровск  2023 |

|  |
| --- |
| **Графика на языке Python**: учебное пособие / Н.Н. Гончаренко, Е.В. Редько. – Хабаровск: МАОУ «Лицей инновационных технологий». – 2023. – 47 с. |
| Настоящее пособие может быть использовано педагогами, реализующими дополнительные общеобразовательные программы естественнонаучной направленности, учащимися, учителями информатики, студентами–бакалаврами педагогических направлений подготовки.  Пособие состоит из двух частей. Первая часть содержит теоретический материал по базовым принципам создания графических примитивов на языке Python и способам анимации.  Вторая часть представляет собой сборник практических работ для школьников с примерами создания анимации на простых геометрических объектах и заданиями для самостоятельной работы. Выполнение практических работ предполагает владение базовыми навыками программирования на языке Python. Дополнительные средства рисования (инструменты библиотеки Tkinter, «черепашья» графика) описаны непосредственно в пособии в объеме, достаточном для выполнения заданий. |
| © МАОУ ЛИТ, 2023 |

# Содержание

[Содержание 3](#_Toc129654797)

[Введение 4](#_Toc129654798)

[1. Средства рисования в языке Python 5](#_Toc129654799)

[1.1 Графическая библиотека Tkinter 5](#_Toc129654800)

[1.2 Анимация в Tkinter 16](#_Toc129654801)

[1.3 Исполнитель «Черепашка» 22](#_Toc129654802)

[2. Практические работы 24](#_Toc129654803)

[Практическая работа № 1. Создание рисунка «Аквариум с рыбками» с использованием графических примитивов Tkinter 24](#_Toc129654804)

[Практическая работа № 2. Анимация для рыбок в аквариуме 27](#_Toc129654805)

[Практическая работа № 3. Анимация бабочки с изменением траектории движения 31](#_Toc129654806)

[Практическая работа № 4. Создание рисунка «Домик» с помощью черепашьей графики 35](#_Toc129654807)

[Задания для самостоятельной работы 41](#_Toc129654808)

[Список источников 46](#_Toc129654809)

# Введение

Цель обучения информатике в школе – подготовка учеников к работе в информационном обществе. Для этого необходимо хорошо владеть разными типами прикладных программ, уметь самостоятельно осваивать программные продукты, выбирать адекватный инструмент под задачу, развивать системный подход, алгоритмическое мышление. Интерпретируемый язык программирования Python идеально подходит для этих целей.

Язык Python – язык высокого уровня общего назначения, допускающий возможность создания настоящих, графически оформленных, работоспособных программ. Изучение Python развивает математическую интуицию и геометрические представления, формирует алгоритмический, структурный, логический и комбинаторный типы мышления, повышает творческую активность и самостоятельность школьников.

Обучение школьников приемам построения компьютерной графики имеет не только прикладное значение, но и способствует развитию алгоритмического мышления.

В пособии рассмотрены принципы рисования простых объектов с использованием графических примитивов языка Python, а также модуль Turtle, разработаны задания для проверки усвоения материала.

Учебное пособие «Графика на языке Python» актуально для использования учителями информатики в рамках проведения уроков по программам углубленного изучения информатики, а также при организации занятий внеурочной деятельности.

# 1. Средства рисования в языке Python

## 1.1 Графическая библиотека Tkinter

**Tkinter (Tk)** – графическая библиотека Python, которая предназначена для создания программ с оконным интерфейсом. Она кроссплатформенная, то есть с ее помощью можно писать приложения для Windows, Linux, macOS.

Кроссплатформенная графическая библиотека **Tk** содержит компоненты графического интерфейса пользователя (*graphical user interface* – GUI). Эта библиотека написана на языке программирования **Tcl** (**Tcl** – это скриптовый язык). Область применения **Tcl/Tk** – быстрое прототипирование. С его помощью создают графические интерфейсы, встраивают новые сценарии в программы, тестируют.

Чтобы работать с **Tcl/Tk** на Python, используется библиотека **Tkinter**. Таким образом, **Tkinter**– это интерфейс **Tcl/Tk**. На английском его обычно называют «тикль–ток», на русском – «так–тикль».

Удобство **Tkinter** в том, что она входит в стандартный набор Python. Если в системе установлен Python, то **Tkinter** тоже работает.

### 1.1.1 Общая схема создания окна

Листинг 1:

From tkinter import \* # Импортируем библиотеку

window = Tk() # Создаем новое окно

window.mainloop() # Запускаем бесконечный цикл окна

**Tk** является базовым классом любого **Tkinter** приложения. При создании объекта этого класса запускается интерпретатор **tcl/tk** и создаётся базовое окно приложения.

**Tkinter** является событийно–ориентированной библиотекой. В приложениях такого типа имеется главный цикл обработки событий. В **Tkinter** такой цикл запускается методом **mainloop()**. Окно будет ждать любого действия от пользователя до тех пор, пока пользователь его не закроет (рис. 1). Без **mainloop()** окно не отобразится.

Для явного выхода из интерпретатора и завершения цикла обработки событий используется метод **quit()**.

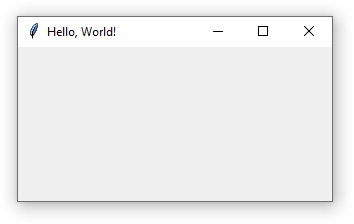


Рисунок 1 – Пустое окно (результат работы программы листинга 1)

Особенность **Tkinter** в том, что библиотека автоматически подстраивает внешний вид окна под стиль операционной системы. На Windows, Linux и macOS оно будет различным, соответствующим стилю системы (рис. 2).



Рисунок 2. Различие стилей базового окна, объекта класса Tk, в зависимости от операционной системы

### 1.1.2 Базовые виджеты

Виджеты – это основа библиотеки **Tkinter**. Через них пользователи взаимодействуют с программой.

Каждый виджет определен классом (см. таблицу 1).

Таблица 1 – Классы виджетов

|  |  |  |
| --- | --- | --- |
| **Класс виджета** | **Что делает** | **Пример использования** |
| **Frame** | Помогает организовать пользовательский интерфейс как визуально, так и на уровне кода. Отображается как простой прямоугольник. | Разделение интерфейса на блоки. |
| **Label** | Виджет для отображения текста или изображения. Пользователи могут его просматривать, но не могут взаимодействовать с ним. | Заголовки, подписи, иконки в интерфейсе. |
| **Button** | Это элемент интерфейса, с которым пользователи могут взаимодействовать. На кнопку нажимают, чтобы выполнить действие. **Button** может отображать текст и изображение, как и **Label**, но также может принимать дополнительные параметры для изменения поведения. | Кнопки в интерфейсе: для перехода, сохранения, выхода из программы. |
| **Checkbutton** | Разновидность кнопки, которая содержит двоичное значение. При нажатии переключатель переворачивается, затем происходит обратный вызов. | Включение и выключение опций с помощью галочек. |
| **Radiobutton** | Кнопка, которая позволяет выбрать один из нескольких взаимоисключающих вариантов. | Список с опциями, например, для выбора языка интерфейса. |
| **Entry** | Виджет для ввода одной строки текста. | Указание имени, пароля, города и других данных пользователей. |
| **Combobox** | Объединяет **Entry** со списком опций. Пользователи могут выбирать из предложенных вариантов или указывать свои. | Выпадающий список. |

Это лишь базовые виджеты. Посмотреть их подробное описание можно в официальной документации **Tk**.

### 1.1.3 Общие для всех виджетов свойства

Все виджеты в **Tkinter** обладают некоторыми общими свойствами. Опишем их, перед тем как перейти к рассмотрению конкретных виджетов. Виджеты создаются вызовом конструктора соответствующего класса.

Первый аргумент (как правило неименованный, но можно использовать имя **master**) это родительский виджет, в который будет упакован (помещен) создаваемый виджет. Родительский виджет можно не указывать, в таком случае будет использовано главное окно приложения.

Далее следуют именованные аргументы, конфигурирующие виджет. Это может быть используемый шрифт (**font**=...), цвет виджета (**bg**=...), команда, выполняющаяся при активации виджета (**command**=...) и так далее. Полный список всех аргументов можно посмотреть в **manoptions** и **man**–странице соответствующего виджета (например **manbutton**, см. разделы «STANDARD OPTIONS» и «WIDGET–SPECIFIC OPTIONS»).

Приведем пример кода, создающего окно с двумя кнопками: первая кнопка будет закрывать окно (завершать работу программы), вторая – печатать заданное слова в консоль Python (рис.3).

Листинг 2:

from tkinter import \*

def button\_clicked():

print ("Клик!")

root=Tk()

button1 = Button(root,bg="grey",

width=10,

font='Arial 14',

text="Close",

command=root.destroy)

button1.pack()

button2 = Button(

root,

bg="red",

text="Кликни меня!",

command=button\_clicked

)

button2.pack()

root.mainloop()

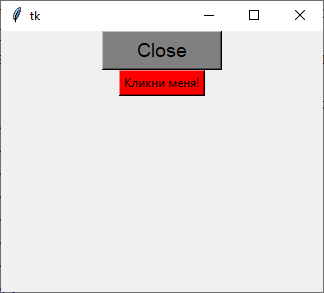


Рисунок 3 – Окно с двумя кнопками, созданными в листинге 2

В любом приложении виджеты не разбросаны по окну как попало, а организованы, интерфейс продуман до мелочей и обычно подчинен определенным стандартам. В примере листинга 2 элементы расположены друг под другом с помощью наиболее простого менеджера геометрии – **pack()**.

### 1.1.4 Виджет Canvas

Рисование в **Tkinter** реализовано при помощи виджета **Canvas**. Виджет **Canvas** имеет функционал высокого уровня, который позволяет создавать графику в **Tkinter**. Рисование можно использовать для создания графиков статистики, самодельных пользовательских виджетов и даже небольших игр.

Виджет Canvas как графическое полотно имеет систему координат с началом в левом верхнем углу и положительным направлением оси OY сверху вниз (рис. 4). Размеры канвы, задаваемые при создании, определяют максимальные возможные координаты.

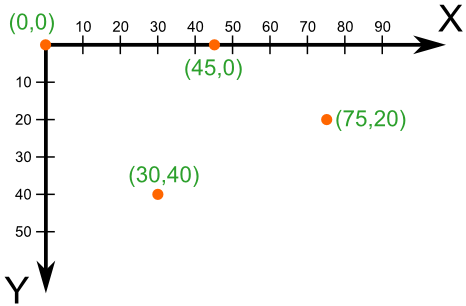


Рисунок 4. Система координат для компьютерной графики

Общая схема программы, использующей виджет Canvas (рис. 5), представлена в листинге 3.

Листинг 3:

from tkinter import \*

root=Tk()

canvas=Canvas(root, width=400, height=200)

canvas.create\_line(

60, 90, 160, 90, 110, 185, 60, 90,

fill="red",

width=4

)

canvas.pack()

button = Button(

root,

bg="grey",

text="Close",

command=root.destroy

)

button.pack()

root.mainloop()

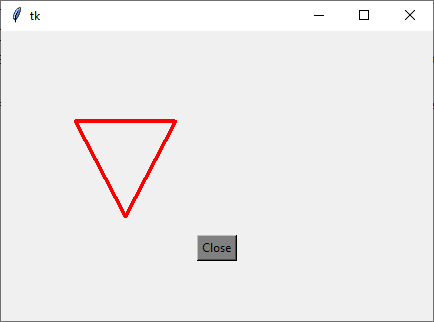


Рисунок 5. Пример окна, на котором размещен графический виджет Canvas

Рассмотрим методы класса **Canvas** для построения графических объектов (см. таблицу 2).

Таблица 2 – Методы класс Canvas

|  |  |  |
| --- | --- | --- |
| **Графический объект** | **Параметры** | **Примеры** |
| Линия (отрезок прямой)  **create\_line()** | **x1, y1, x2, y2** – координаты начальной и конечной точек;  **dash(a, b)** – пунктирная линия, где **a** – длина пунктирного тире в пикселях, **b** – пустой промежуток между тире;  **fill**– **цвет линии;**  **width** – толщина линии | **canvas.create\_line(20, 30, 220, 30)**  #горизонтальная сплошная линия  **canvas.create\_line(300, 35, 300, 150, dash=(3, 2))**  #вертикальная пунктирная линия |
| Ломаная линия  **create\_line()** | *n*–пар координат **x, y** точек, которые будут соединяться линией  **fill**– **цвет линии**  **width** – толщина линии | **canvas.create\_line(**  **60, 90, 160, 90,**  **110, 185, 60, 90,**  **fill="red",**  **width=4**  **)**  #замкнутая ломаная красного цвета (треугольник) с координатами вершин (60, 90), (160, 90), (110, 185) и толщиной линии 4  **canvas.create\_line(**  **250, 20, 380, 90,**  **300, 130, 160, 10,**  **fill="black",**  **width=3**  **)**  # ломаная черного цвета с четырьмя вершинами |
| Прямоугольник  **create\_rectangle()** | **x1, y1, x2, y2** – координаты левой верхней и правой нижней вершин;  **outline** – цвет контурной линии;  **fill** – закраска внутренней области прямоугольника;  **width** – толщина контура. | **canvas.create\_rectangle(**  **30, 10, 320, 180,**  **outline="#fb0", fill="#fb0"**  **)** |
| Многоугольник  **create\_polygon()** | *n*–пар координат **x, y** вершин многоугольника;  **outline** – цвет контурной линии;  **fill** – закраска внутренней области многоугольника;  **width** – толщина контура. | **points = [**  **150, 100, 200, 120,**  **240, 180, 210, 200,**  **150, 150, 100, 200**  **]**  **canvas.create\_polygon(**  **points, outline='maroon',**  **fill='yellow', width=4**  **)** |
| Круг (овал)  **create\_oval()** | Первые четыре параметра определяют ограничивающие координаты фигуры. Иными словами, это **x** и **y** координаты верхней левой и правой нижней точек квадрата (прямоугольника), в который вписан круг (овал);  **outline** – цвет контурной линии;  **fill** – закраска внутренней области фигуры;  **width** – толщина контура. | **canvas.create\_oval(**  **50, 50, 280, 180,**  **outline="green",**  **fill="yellow",**  **width=3**  **)** |
| Сектор  **create\_arc()** | Первые четыре параметра определяют ограничивающие координаты фигуры. Иными словами, это **x** и **y** координаты верхней левой и правой нижней точек квадрата (прямоугольника), в который помещен сектор;  **start** – начальный угол сектора;  **extent** – конечный угол сектора;  **outline** – цвет контурной линии;  **fill** – закраска внутренней области фигуры;  **width** – толщина контура. | **canvas.create\_arc(**  **100, 30, 290, 130,**  **start=0,**  **extent=270,**  **outline="#dbd",**  **fill="#0bb",**  **width=4**  **)** |
| Текст  **canvas.create\_text()** | Первые два параметра – это **x** и **y** координаты центральной точки текста;  Если мы закрепим текстовый объект по направлению запада **anchor=W**, текст будет начинаться в этой части окна;  **font** – шрифт текста;  **text** – строка, выводимая на область рисования Canvas. | **canvas.create\_text(**  **100, 100,**  **anchor=W,**  **font=("Courier",12,"bold"),**  **text="Ваш текст",**  **fill="blue",**  **width=250**  **)** |

## 1.2 Анимация в Tkinter

### 1.2.1 Первый способ реализации движения (анимации) объекта – метод move() из библиотеки Tkinter.

Метод move(), вызванный для Canvas принимает идентификатор объекта и целые числа – смещение по осям.

Чередование движения объектов выполняется через метод after(). Метод after() вызывает функцию, переданную вторым аргументом, через количество миллисекунд, указанных первым аргументом.

Отслеживать положение объекта на Canvas позволяет метод coords(). Выражение c.coords(d) является списком текущих координат объекта d. Третий элемент списка соответствует его второй координате x.

Рассмотрим пример программы, использующей вышеописанные методы для решения задачи: движение прямоугольника горизонтально от левой границы окна к правой (по достижении прямоугольником правой границы он продолжает движение слева).

Листинг 4:

from tkinter import \*

root = Tk()

c = Canvas(root, width=500, height=500)

c.pack()

def motion():

c.move(r,5,0)

if c.coords(r)[2] < 600:

root.after(5, motion)

if c.coords(r)[2] >= 600:

root.after(5, motionback)

root.after(5, motion)

def motionback():

c.move(r,–600,0)

r = c.create\_rectangle(0,250,50,270,\

fill="blue", outline="black")

motion()

В основной программе на Canvas создан прямоугольник с идентификатором r и вызвана функция motion(), которая и реализует движение прямоугольника.

Функция motion() делает сдвиг прямоугольника r с помощью метода move() на 5 пикселей в положительном направлении горизонтальной оси x и 0 пикселей в вертикальном направлении y.

После каждого такого сдвига при помощи метода coords() проверяется значение координаты x – если оно не превышает горизонтальные размеры окна (600 пикселей), то при помощи метода after() задается задержка в 5 миллисекунд и очередной вызов функции motion(). Если же прямоугольник вышел за границы окна, то с задержкой 5 миллисекунд методом after() вызывается функция motionback(), которая перемещает прямоугольник на стартовую позицию в левой части окна и снова запускается функция motion().

### 1.2.2 Второй способ реализации движения объекта – изменение его координат coords()

Другим способом анимацию можно реализовать с помощью изменения координат объекта.

Метод coords() для Canvas позволяет не только проверять значения координат, но и устанавливать новые.

Приведем пример программы для решения задачи: движение пузырька воздуха в закрытом сосуде.

Листинг 5:

from tkinter import\*

import math

import time

class bubble:

def \_\_init\_\_(self,c):

self.c=c

self.x=250

self.y=250

self.d=5

self.angle=3.14/3

self.size=50

self.dx=self.d\*math.cos(–self.angle)

self.dy=self.d\*math.sin(–self.angle)

self.id=c.create\_oval(self.x–self.size,self.y–self.size,self.x+self.size,self.y+self.size,outline='powderblue',fill='lightcyan',width=3)

def delete(self):

self.c.delete(self.id)

def move\_bubble(self):

self.c.coords(self.id,self.x–self.size,self.y–self.size,self.x+self.size,self.y+self.size)

self.c.after(20,self.move\_bubble)

if self.x>=500–self.size or self.x<=self.size:

self.angle=(3.14–self.angle)

self.dx=self.d\*math.cos(–self.angle)

self.dy=self.d\*math.sin(–self.angle)

if self.y>=500–self.size or self.y<=self.size:

self.angle\*=–1

self.dx=self.d\*math.cos(–self.angle)

self.dy=self.d\*math.sin(–self.angle)

self.x+=self.dx

self.y+=self.dy

def main():

root =Tk()

c = Canvas(root, width=500, height=500)

c.pack(fill=BOTH,expand=1)

bubble1=bubble(c)

root.update()

bubble1.move\_bubble()

root.update()

if \_\_name\_\_=='\_\_main\_\_':

main()

В основной программе к созданной Canvas применен метод pack() – менеджер геометрии, для которого параметр fill() установлен в значение BOTH, что значит, что канва будет заполнять все пространство окна. Второй параметр expand (расширение) установлен в значение 1, чтобы канва всегда располагалась в центре окна, если оно будет расширяться.

Для работы с пузырьком создадим класс bubble. Рассмотрим далее методы класса.

Метод инициализации \_\_init\_\_() является конструктором и принимает два параметра ***self*** (чтобы обращаться к объекту) и ***c*** типа Canvas, то есть канва, на которой будет существовать объект. В методе \_\_init\_\_() перечисляется список свойств класса и задаются их значения по умолчанию: «родительская» канва (с), начальные координаты (x, y), длина смещения (d), угол направления движения (angle), диаметр (size), значения смещения по каждой из координат (dx, dy), идентификатор графического объекта (id). Графический объект в данной задаче – это круг. Рассмотрим параметры круга – примитива oval (см. таблицу 2):

self.id=c.create\_oval(self.x–self.size,self.y–self.size,self.x+self.size,self.y+self.size,outline='powderblue',fill='lightcyan',width=3)

Первые четыре параметра определяют ограничивающие координаты фигуры. Иными словами, это *x* и *y* координаты верхней левой и правой нижней вершин квадрата (прямоугольника), в который помещен круг (овал). Зная координаты центра окружности пузырька (self.x, self.y) и его радиус (self.size), соответственно рассчитаем координаты левого верхнего угла как self.x–self.size, self.y–self.size, а правого нижнего, как self.x+self.size, self.y+self.size (рис. 6).

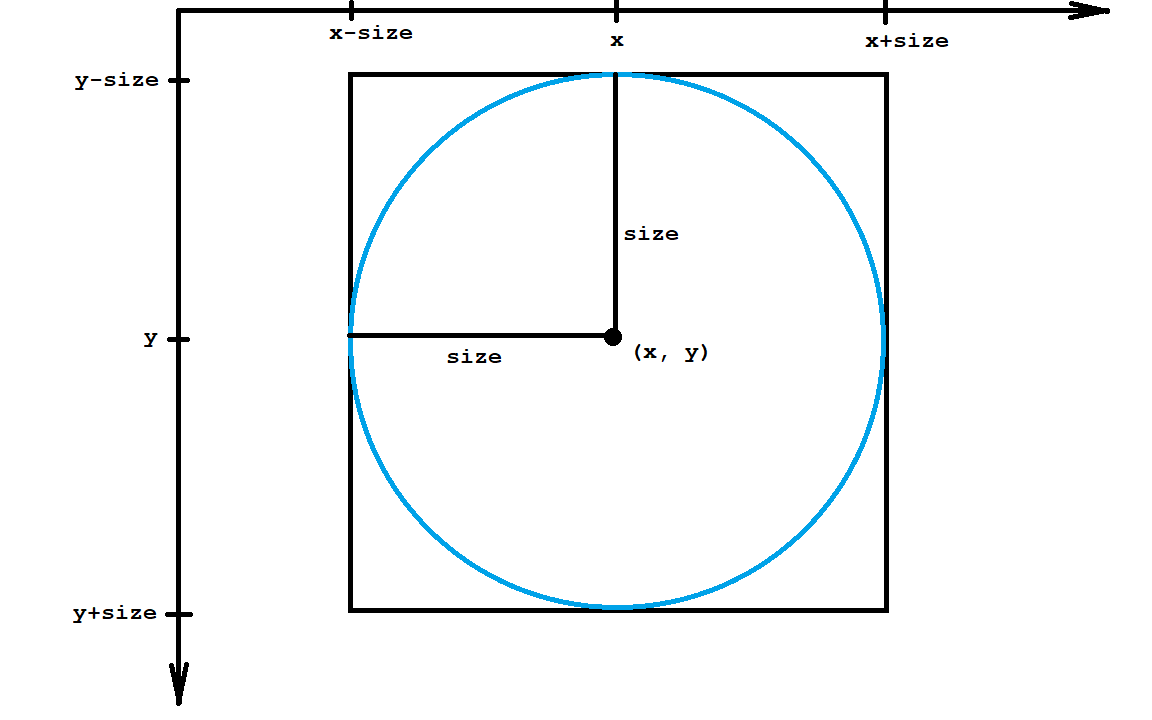


Рисунок 6. Расчет параметров для построения окружности на канве через заданные координаты центра и радиус

Цвет контурной линии outline зададим насыщенным голубым – powerblue, а закраскe внутренней области фигуры fill – светлоголубым lightcyan.

Толщина контура для пузырька width равна 3.

Метод delete() предназначен для того чтобы иметь возможность удалить объект с канвы.

Метод move\_bubble() отвечает за движение. Основной шаг движения задан при помощи coords(), который перерисовывает bubble в новых координатах, – изменение значений координат на величину смещения задано в конце данного метода move\_bubble():

self.x+=self.dx

self.y+=self.dy

Пересчет угла направления движения и величин смещения выполняется в тех случаях, когда пузырек достиг стенок сосуда (границ канвы): если пузырек отталкивается от вертикальных стенок (self.x>=500–self.size or self.x<=self.size), то новый угол – это π–старый угол. Если же пузырек отталкивается от горизонтальных стенок (self.y>=500–self.size or self.y<=self.size), то значение угла меняется на противоположное.

В основной программе создаем один объект класса bubble и вызываем для него метод move\_bubble().

## 1.3 Исполнитель «Черепашка»

Библиотека **turtle** – это расширение языка Python, позволяющее рисовать на экране несложные рисунки. Представьте себе, что по экрану компьютера ползает маленькая черепашка (**turtle**). Вы можете управлять движением черепашки, отдавая ей различные команды вида «Проползти вперед на 10 пикселей», «Повернуть направо», «Повернуть налево». После того, как вы отдадите ей команду «Начать рисовать», черепашка будет оставлять за собой след, пока не получит команду «Закончить рисовать». Управлять черепашкой можно при помощи инструкций языка Python.

Черепаха находится на координатной плоскости и может передвигаться по ней только вперёд, но может также поворачиваться на месте. При движении черепаха способна чертить линию карандашом, или двигаться, не оставляя за собой след. Есть возможность заменять карандаши, и тем самым управлять толщиной линии и её цветом.

Чтобы использовать исполнителя «черепашку», достаточно сделать импорт библиотеки **turtle**:

import turtle

Далее в программе необходимо создать экземпляр класса turtle:

t = turtle.Turtle()

Если сейчас запустить программу, то появится графическое окно и тут же пропадет. Чтобы этого не происходило, в конце добавим команду:

turtle.done()

Далее «черепашка» может ползти вперед (то есть, куда глаза глядят) с помощью команды **forward()** или назад с помощью команды **backward()**, поворачиваться влево, вправо с помощью команд **left()** и **right()**.

Начало координат в окне для графики модуля **turtle** находится в центре окна. Положительное направление оси X определяется слева направо, положительное направление оси Y определяется снизу вверх, чем больше X, тем правее черепашка, чем больше Y, тем выше черепашка.

Остальные часто используемые команды:

* begin\_fill() – начало рисования закрашенной области
* end\_fill() – завершение рисования закрашенной области
* color(color) – задать цвет курсора color ;
* penup() – поднять курсор, чтобы при перемещении курсора не происходило рисование линий;
* pendown() – опустить курсор, чтобы при перемещении курсора происходило рисование линий;
* goto(x, y) – перейти в точку с координатами x и y;
* circle(radius) – нарисовать круг с радиусом = radius;
* circle(r,n) – начертить дугу радиуса r, градусной мерой n против часовой стрелки, если r>0, по часовой стрелке, если r<0
* clear() – очистка холста
* done() – этим методом должны завершаться все программы использующие Turtle.

Работу с исполнителем «черепашка» продемонстрируем в практической работе № 4.

# 2. Практические работы

## Практическая работа № 1. Создание рисунка «Аквариум с рыбками» с использованием графических примитивов Tkinter

Сначала надо создать новое окно и холст:

from tkinter import \*

window = Tk()

window.title('Рыбка')

c = Canvas(window,width=600,height=600,bg="#ff9")

Теперь на холсте нарисуем два прямоугольника, один из которых будет аквариумом, а второй – его подставкой:

c.create\_rectangle(100,100,500,450,

fill="light blue", outline="black") #аквариум

c.create\_rectangle(75,450,525,500,

fill="green", outline="black") #подставка аквариума

На дне аквариума нарисуем 4 камня разных цветов и размеров. Координаты y правых нижних точек прямоугольников, в которые вписаны круги (камни) должны совпадать с координатой y правой нижней точки прямоугольника (аквариума), а координаты x должны быть больше координаты x левой верхней точки аквариума и меньше координаты x правой нижней точки аквариума:

c.create\_oval([125,400],[200,450],fill="light gray", outline="black") #серый овальный камень

c.create\_oval([200,420],[230,450],fill="light yellow", outline="black") #светло–желтый круглый камень

c.create\_oval([300,420],[330,450],fill="purple", outline="black") #фиолетовый круглый камень

c.create\_oval([330,390],[420,450],fill="orange", outline="black") #оранжевый овальный камень

c.create\_oval([450,420],[480,450],fill="blue", outline="black") #синий круглый камень

Теперь перейдем к рисованию первой рыбки. Сделаем её круглой с треугольными плавниками и хвостом, а также нарисуем два глаза:

c.create\_oval([325,150],[450,275],fill="yellow", outline="black") #тело желтой рыбки

c.create\_polygon([300,225],[355,205],[350,150],

fill="orange", outline="red") #левый плавник

c.create\_polygon([425,290],[405,225],[460,195],

fill="orange", outline="red") #правый плавник

c.create\_polygon([420,160],[400,115],[470,150],

fill="red", outline="black") #хвост

c.create\_oval([340,220],[360,240],fill="white", outline="black") #белок левого глаза

c.create\_oval([342,228],[352,238],fill="black", outline="black") #зрачок левого глаза

c.create\_oval([370,240],[390,260],fill="white", outline="black") #белок правого глаза

c.create\_oval([372,248],[382,258],fill="black", outline="black") #зрачок правого глаза

Вторую рыбку сделаем треугольной с плавником и хвостом – секторами, у нее будет только один глаз, так как она повернута боком:

c.create\_polygon([275,250],[275,375],[150,350],

fill="crimson", outline="black") #тело малиновой рыбки

c.create\_arc([130,210], [230,310], start=180,

extent=250, fill='blue', outline='black') #хвост

c.create\_arc([210,360],[250,400], start=0,extent=200, fill='purple') #плавник

c.create\_oval([260,350],[273,363], fill="white") #белок глаза

c.create\_oval([267,353],[272,358], fill="black") #зрачок глаза

В конце программы «упакуем» объекты и запустим цикл обработки событий:

c.pack()

window.mainloop()

Результат работы программы приведен на рисунке.

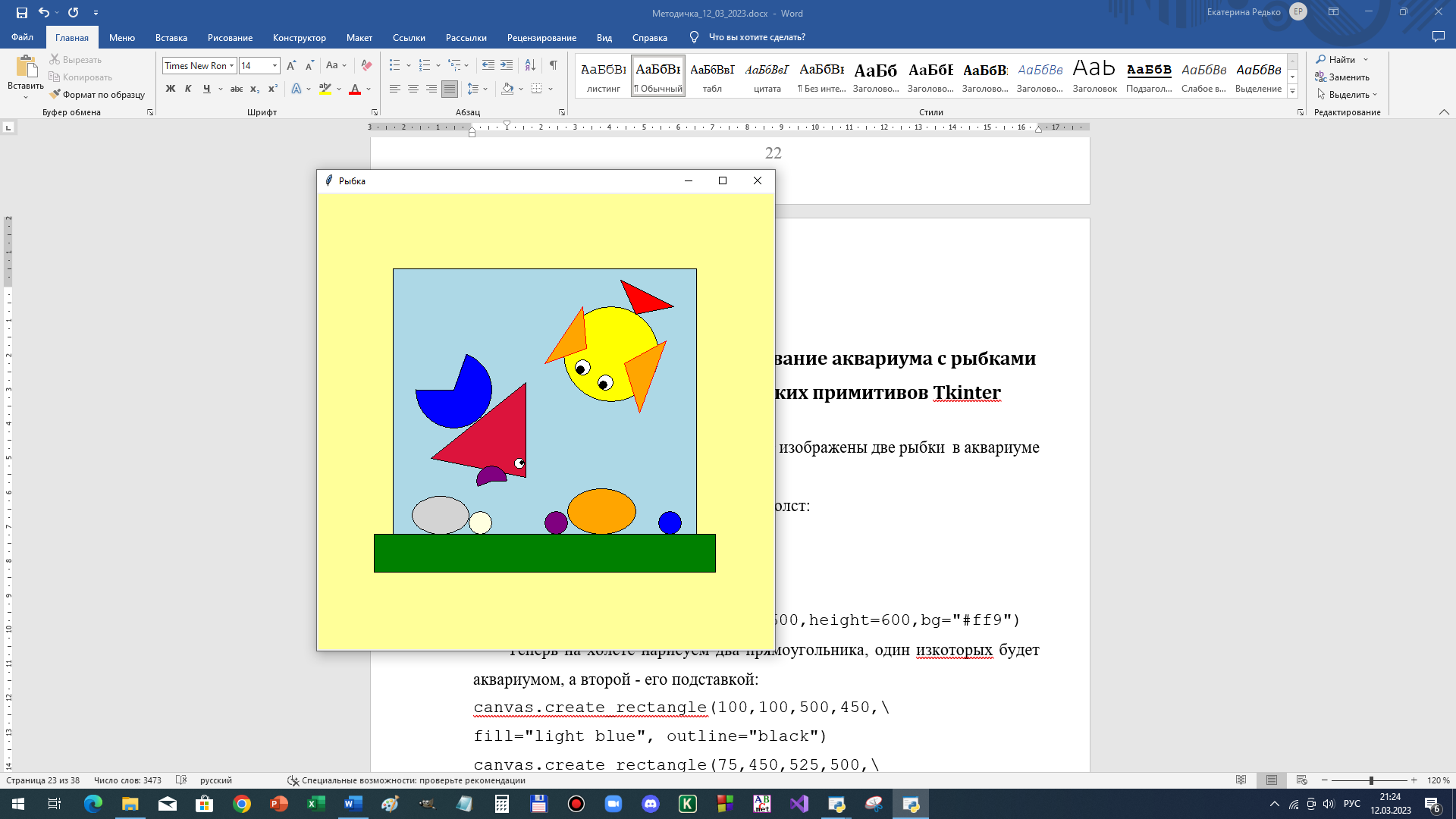


Рисунок 7. Демонстрация работы программы рисования аквариума с рыбками

## Практическая работа № 2. Анимация для рыбок в аквариуме

Для создания анимации используем программу, которая у нас получилась в результате выполнения Практической работы №1.

Чтобы передвигать рыбок, нужно дать названия всем геометрическим фигурам, из которых они состоят:

ryba1 = c.create\_polygon([275,250],[275,375],[150,350],

fill="crimson", outline="black")

ryba2 = c.create\_arc([130,210], [230, 310], start=180,

extent=250, fill='blue', outline='black')

ryba3 = c.create\_arc([210,360],[250,400], start=0,

extent=200, fill='purple')

ryba4 = c.create\_oval([260,350],[273,363], fill="white")

ryba5 = c.create\_oval([267,353],[272,358], fill="black")

rybka1 = c.create\_oval([325, 150],[450, 275],

fill="yellow", outline="black")

rybka2 = c.create\_polygon([300,225],[355, 205],

[350,150],fill="orange", outline="red")

rybka3 = c.create\_polygon([425, 290],[405, 225],

[460,195],fill="orange", outline="red")

rybka4 = c.create\_polygon([420, 160],[400, 115],

[470,150],fill="red", outline="black")

rybka5 = c.create\_oval([340, 220],[360, 240],

fill="white", outline="black")

rybka6 = c.create\_oval([342, 228],[352, 238],

fill="black", outline="black")

rybka7 = c.create\_oval([370, 240],[390, 260],

fill="white", outline="black")

rybka8 = c.create\_oval([372, 248],[382, 258],

fill="black", outline="black")

Теперь напишем функцию motion1(), с помощью которой будем передвигать первую рыбу вправо:

def motion1():

c.move(ryba1, 1, 0)

c.move(ryba2, 1, 0)

c.move(ryba3, 1, 0)

c.move(ryba4, 1, 0)

c.move(ryba5, 1, 0)

if c.coords(ryba1)[2] < 450:

root.after(10, motion1)

В функции motion() с помощью функции move() каждый элемент рыбки сдвигается на один пиксель вправо по x, пока рыба не достигнет правой границы аквариума. Осталось написать такую же функцию для второй рыбки, которая будет передвигаться влево.

Приведем полный листинг программы.

Листинг 6:

from tkinter import \*

root = Tk()

c = Canvas(root,width=600,height=600,bg="#ff9")

c.pack()

c.create\_rectangle(100,100,500,500,fill="light blue", outline="black")

c.create\_rectangle(75,450,525,500,fill="green", outline="black")

c.create\_oval([125,400],[200,450],fill="light gray", outline="black")

c.create\_oval([200,420],[230,450],fill="light yellow", outline="black")

c.create\_oval([300,420],[330,450],fill="purple", outline="black")

c.create\_oval([330,390],[420,450],fill="orange", outline="black")

c.create\_oval([450,420],[480,450],fill="blue", outline="black")

ryba1 = c.create\_polygon([275,250],[275,375],[150,350],

fill="crimson", outline="black")

ryba2 = c.create\_arc([130,210], [230,310], start=180,

extent=250, fill='blue', outline='black')

ryba3 = c.create\_arc([210,360],[250,400], start=0,

extent=200, fill='purple')

ryba4 = c.create\_oval([260,350],[273,363], fill="white")

ryba5 = c.create\_oval([267,353],[272,358], fill="black")

rybka1 = c.create\_oval([325,150],[450,275],

fill="yellow", outline="black")

rybka2 = c.create\_polygon([300,225],[355,205],

[350,150], fill="orange", outline="red")

rybka3 = c.create\_polygon([425,290],[405,225],

[460,195], fill="orange", outline="red")

rybka4 = c.create\_polygon([420,160],[400,115],

[470,150], fill="red", outline="black")

rybka5 = c.create\_oval([340, 220],[360, 240],

fill="white", outline="black")

rybka6 = c.create\_oval([342, 228],[352, 238],

fill="black", outline="black")

rybka7 = c.create\_oval([370, 240],[390, 260],

fill="white", outline="black")

rybka8 = c.create\_oval([372, 248],[382, 258],

fill="black", outline="black")

def motion1():

c.move(ryba1, 1, 0)

c.move(ryba2, 1, 0)

c.move(ryba3, 1, 0)

c.move(ryba4, 1, 0)

c.move(ryba5, 1, 0)

if c.coords(ryba1)[2] < 450:

root.after(10, motion1)

def motion2():

c.move(rybka1, –1, 0)

c.move(rybka2, –1, 0)

c.move(rybka3, –1, 0)

c.move(rybka4, –1, 0)

c.move(rybka5, –1, 0)

c.move(rybka6, –1, 0)

c.move(rybka7, –1, 0)

c.move(rybka8, –1, 0)

if c.coords(rybka1)[2] > 250:

root.after(10, motion2)

motion1()

motion2()

root.mainloop()

## Практическая работа № 3. Анимация бабочки с изменением траектории движения

Для создания анимации нарисуем бабочку и цветок простыми геометрическими фигурами, зададим имена элементам бабочки, чтобы далее их двигать.

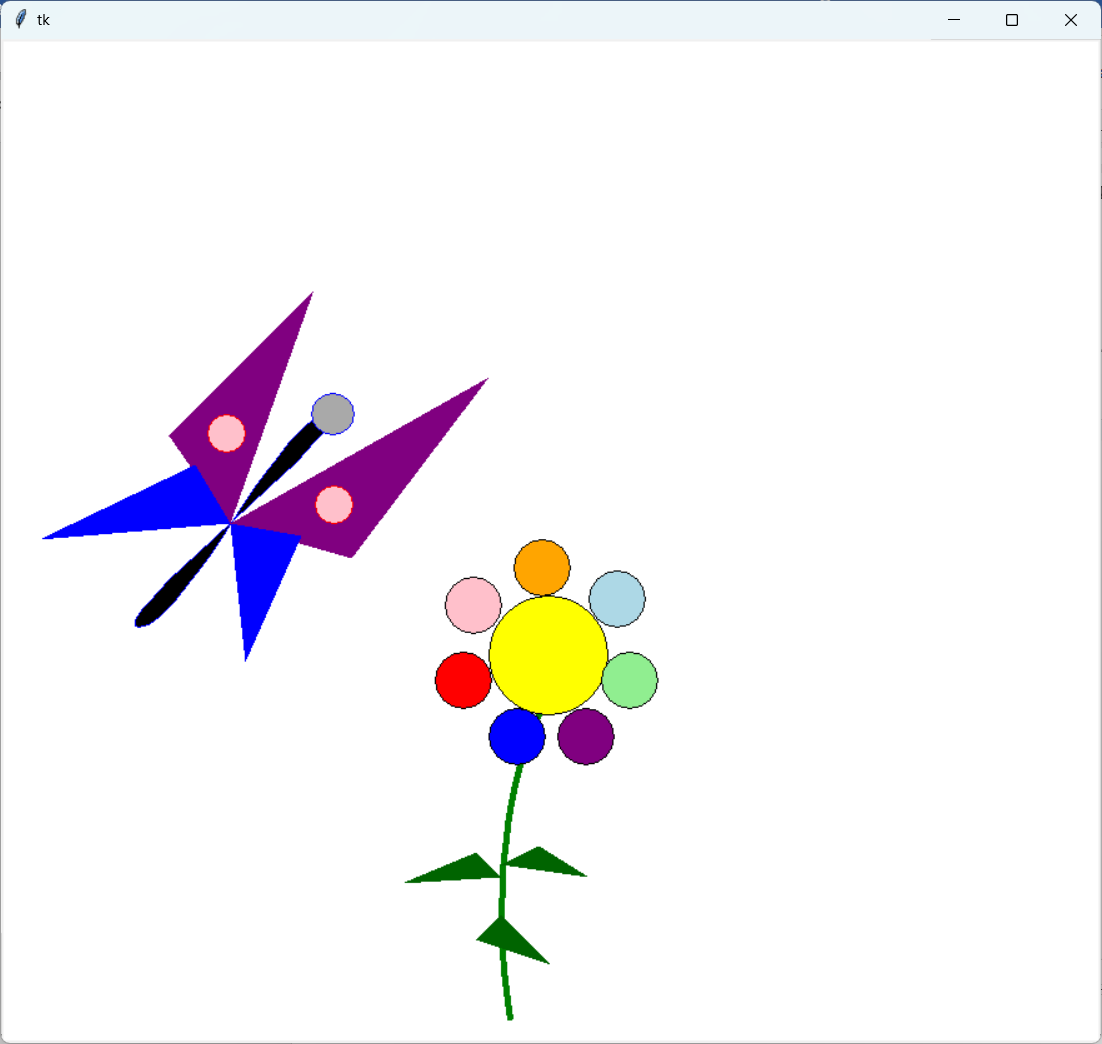


Рисунок . Начальный вид и положение бабочки

Создадим переменную orientation и зададим ей значение "вниз". Далее напишем функцию butterfly(), в которой объявим переменную orientation глобальной:

global orientation

В функции будем проверять значение переменной orientation и, если оно равно "вниз", сдвигать элементы бабочки по диагонали вправо вниз, пока её тело не достигнет границы сердцевины цветка. После этого поменяем значение переменной orientation на "вверх", в данной ситуации бабочка будем двигаться по диагонали вправо вверх (рис. 9).

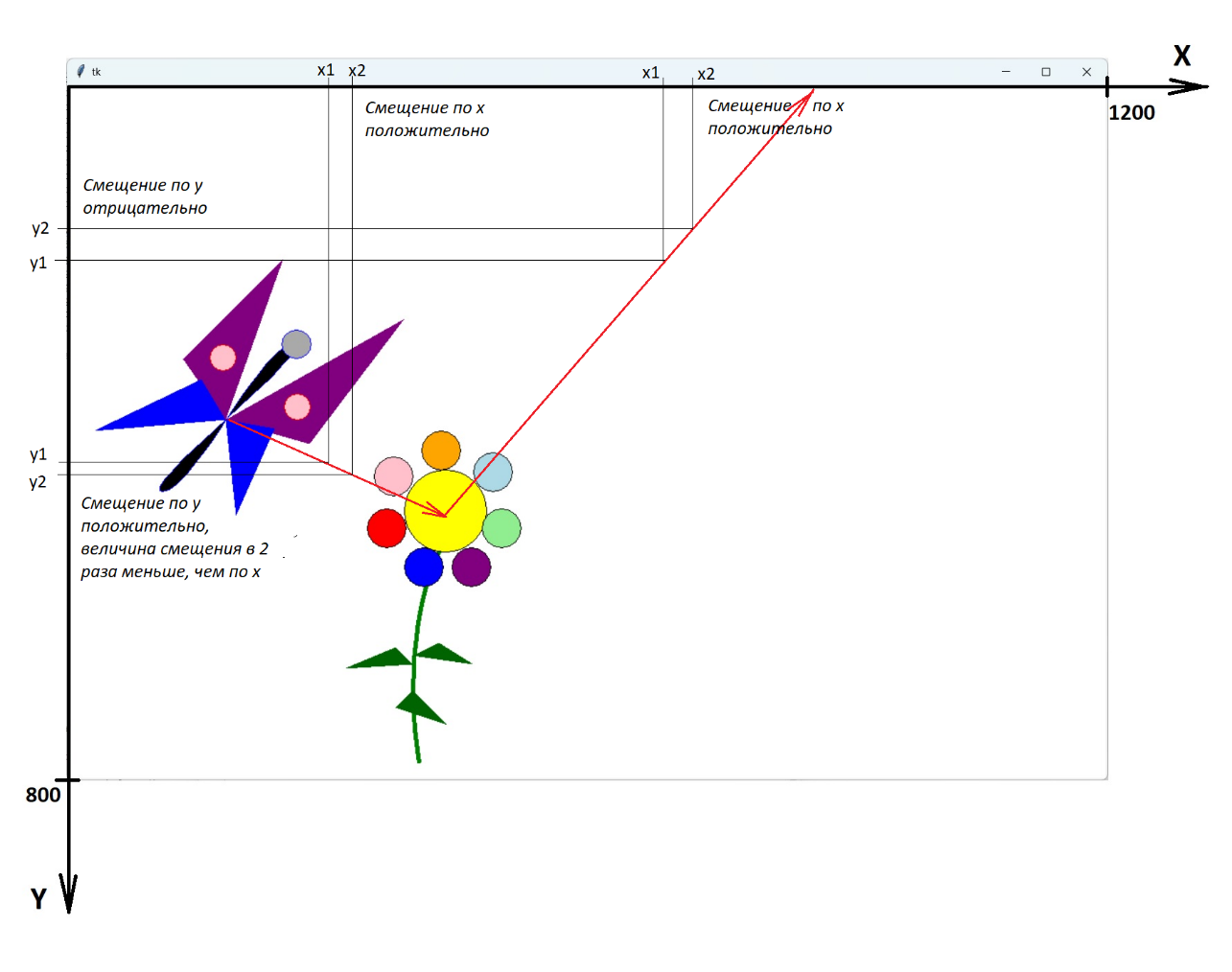


Рисунок . Траектория «полета» бабочки

Приведем пример готовой программы – **листинг 7**:

from tkinter import \*

window = Tk()

c = Canvas(window,width=1200,height=800,bg="white")

c.create\_arc([400,500],[550,900], start=75, extent=130, style=ARC, outline="green", width=5) #стебель

c.create\_polygon([440,740],[380,720],[400,700], fill="darkgreen") #листик 1

c.create\_polygon([400,670],[380,650],[320,675], fill="darkgreen") #листик 2

c.create\_polygon([400,660],[430,645],[470,670], fill="darkgreen") #листик 3

c.create\_oval([390,445],[485,540],fill="yellow") #сердцевина цветка

c.create\_oval([355,430],[400,475],fill="pink") #розовый лепесток

c.create\_oval([410,400],[455,445],fill="orange") #оранжевый лепесток

c.create\_oval([470,425],[515,470],fill="lightblue") #голубой лепесток

c.create\_oval([480,490],[525,535], fill="lightgreen")

#зеленый лепесток

c.create\_oval([445,535],[490,580],fill="purple") #фиолетовый лепесток

c.create\_oval([390,535],[435,580], fill="blue") #синий лепесток

c.create\_oval([347,490],[392,535], fill="red") #красный лепесток

body = c.create\_polygon([280, 300], [240, 300], [130, 470], [90,470], smooth=1, outline="blue") #тело бабочки

head = c.create\_oval([248, 283], [282, 316], outline="blue", fill="darkgray") #голова бабочки

wing1 = c.create\_polygon([250, 200], [183, 387], [134, 317], fill="purple") #крыло 1

wing2 = c.create\_polygon([183, 387], [30, 400], [155, 340],fill="blue") #крыло 2

wing3 = c.create\_polygon([183, 387], [390, 270], [280, 415], fill="purple") #крыло 3

wing4 = c.create\_polygon([183, 387], [195, 498], [240, 397], fill="blue") #крыло 4

pattern1 = c.create\_oval([165, 300], [195, 330], fill="pink", outline="red") #пятнышко на левом крыле

pattern2 = c.create\_oval([251, 357], [281, 387], fill="pink", outline="red") #пятнышко на правом крыле

orientation = "вниз"

def butterfly():

global orientation

if orientation == "вниз":

c.move(body, 2, 1)

c.move(head, 2, 1)

c.move(wing1, 2, 1)

c.move(wing2, 2, 1)

c.move(wing3, 2, 1)

c.move(wing4, 2, 1)

c.move(pattern1, 2, 1)

c.move(pattern2, 2, 1)

if c.coords(body)[2] > 485:

orientation = "вверх"

else:

c.move(body, 4, –5)

c.move(head, 4, –5)

c.move(wing1, 4, –5)

c.move(wing2, 4, –5)

c.move(wing3, 4, –5)

c.move(wing4, 4, –5)

c.move(pattern1, 4, –5)

c.move(pattern2, 4, –5)

window.after(10, butterfly)

c.pack()

butterfly()

window.mainloop()

## Практическая работа № 4. Создание рисунка «Домик» с помощью черепашьей графики

В данной практической работе построим изображение домика (рис. 8).

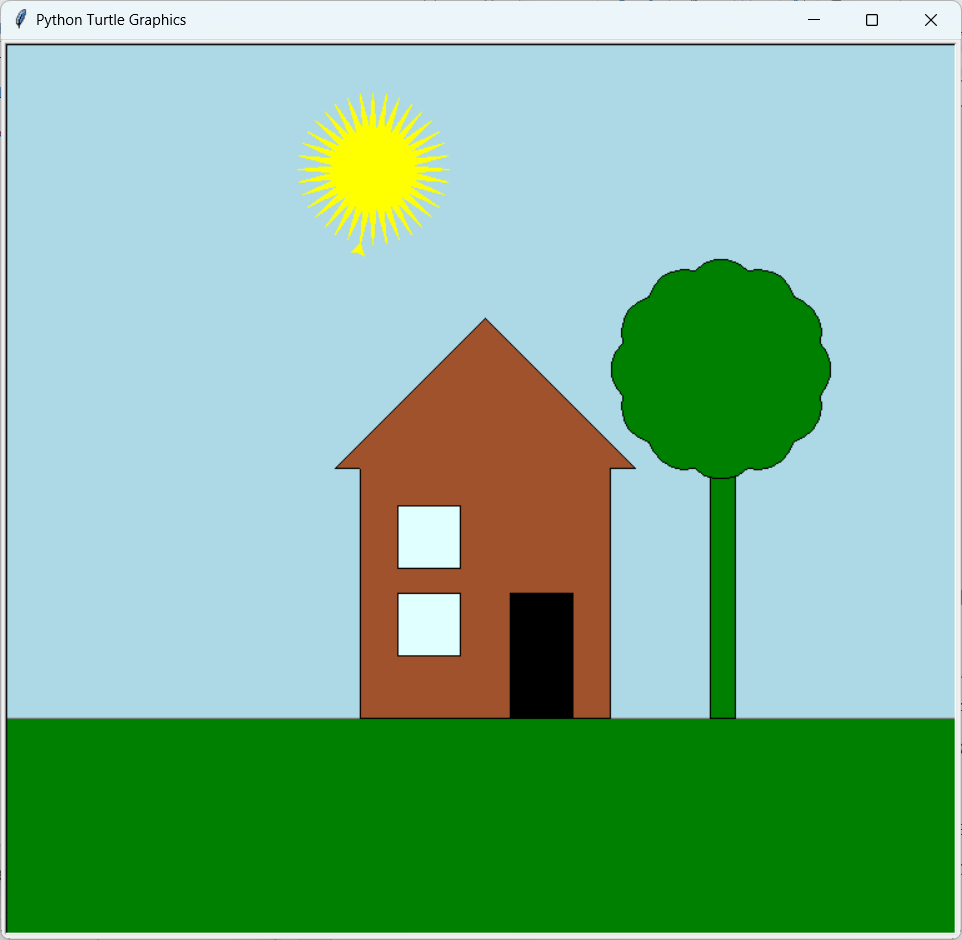


Рисунок 10. Рисунок «Домик», построенный с помощью черепашьей графики

Импортируем библиотеку turtle для работы с черепашьей графикой:

import turtle as t

Для расчета координат нам понадобятся значения ширины и высоты холста, чтобы их вычислить воспользуемся методами window\_height() и window\_width(). Полученные значения сохраним соответственно в переменные h и w, предварительно разделив значения на 2. Зададим черепахе скорость 5.

Сначала нарисуем небо. Для этого подымем перо командой penup(), далее воспользуемся функцией goto(), которая переносит черепаху в точку с заданными координатами. Функции зададим координаты левой нижней точки неба (–w,–h/2). После этого опустим перо командой pendown(), с помощью функции color() зададим перу светло–голубой цвет и черный контур, а затем начнем заливку командой begin\_fill(). Далее запустим цикл, в котором черепашка будет двигаться вперед и поворачивать влево на 90 градусов, в результате чего нарисует прямоугольник – небо.

По такому же принципу построим остальные объекты. Начальная точка земли будет иметь координаты (–w,–h) – левая нижняя точка холста. Домик будем строить по середине холста на поверхности земли, поэтому его начальная точка будет иметь координаты (home\_width//2, –h//2). На домике построим треугольную крышу, а также нарисуем окна и дверь. Справа от дома нарисуем дерево, а на небе – солнышко.

Приведем пример готовой программы.

Листинг 8:

import turtle as t

h = t.window\_height()//2

w = t.window\_width()//2

t.speed(5)

# небо

t.penup()

t.goto(–w, –h//2)

t.pendown()

t.color('gray', 'light blue')

t.begin\_fill()

for i in (w\*2, h\*2–h//2, w\*2, h\*2–h//2):

t.forward(i)

t.left(90)

t.end\_fill()

# земля

t.penup()

t.goto(–w,–h)

t.pendown()

t.color('gray', 'green')

t.begin\_fill()

for i in (w\*2, h//2, w\*2, h//2):

t.forward(i)

t.left(90)

t.end\_fill()

# домик

home\_width = 200 #ширина домика

home\_height = 200 #высота домика

t.color('black', 'sienna')

t.penup()

t.goto(home\_width//2, –h//2)

t.pendown()

t.begin\_fill()

for i in (home\_width, home\_height, home\_width, home\_height):

t.left(90)

t.forward(i)

t.end\_fill()

# крыша домика

t.left(90)

t.begin\_fill()

for i in [(90, home\_width), (90, 20)]:

t.forward(i[1])

t.right(i[0])

for i in [(225, 170), (90, 170), (135, 20)]:

t.left(i[0])

t.forward(i[1])

t.end\_fill()

#верхнее окно

window\_size = 50 #сторона окна

t.penup()

t.goto(–home\_width//2+30, –h//2+120)

t.pendown()

t.color('black', 'lightcyan')

t.right(90)

t.begin\_fill()

for i in (window\_size, window\_size, window\_size, window\_size):

t.left(90)

t.forward(i)

t.end\_fill()

#нижнее окно

t.penup()

t.goto(–home\_width//2+30, –h//2+50)

t.pendown()

t.color('black', 'lightcyan')

t.begin\_fill()

for i in (window\_size, window\_size, window\_size, window\_size):

t.left(90)

t.forward(i)

t.end\_fill()

#дверь

door\_width = 50

door\_height = 100

t.penup()

t.goto(home\_width//2–door\_width–30, –h//2)

t.pendown()

t.color('black', 'black')

t.begin\_fill()

for i in (door\_width, door\_height, door\_width, door\_height):

t.left(90)

t.forward(i)

t.end\_fill()

# дерево

tree\_height = 200

tree\_width = 20

t.color('black', 'green')

t.penup()

t.goto(200, –h//2)

t.pendown()

t.left(90)

t.begin\_fill()

for i in (tree\_height, tree\_width, tree\_height, tree\_width, tree\_height, tree\_width):

t.left(90)

t.forward(i)

t.end\_fill()

t.penup()

t.begin\_fill()

t.goto(200+tree\_width//2, –h//2+tree\_height)

t.pendown()

t.right(135)

for i in range(13):

t.right(60)

t.circle(30, 90)

t.end\_fill()

# солнце

X = –100

Y = 200

t.penup()

t.goto(X, Y)

t.pendown()

t.color('yellow')

t.begin\_fill()

for i in range(36):

t.forward(120)

t.left(170)

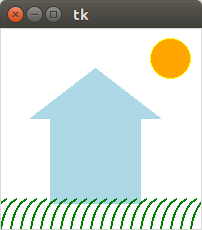
t.end\_fill()

t.done()

## Задания для самостоятельной работы

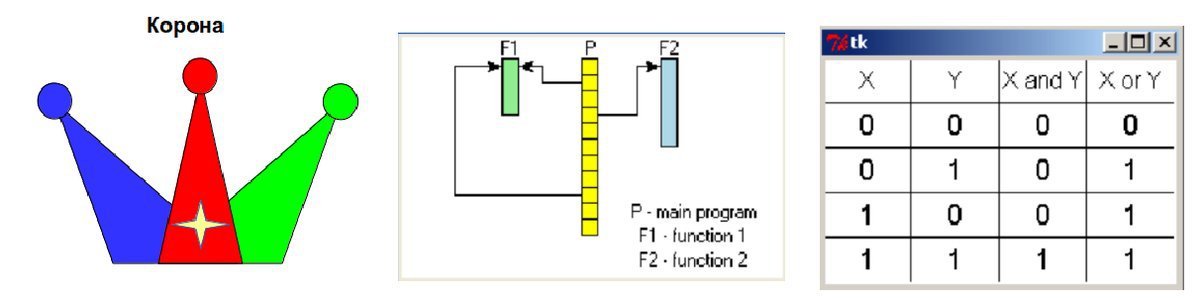
### Создание изображения на холсте в Tkinter

1. Нарисовать окружность с центром в точке (300, 200), и радиусом 50.
2. Нарисовать закрашенную окружность в точке (300, 200), и радиусом 50.
3. Нарисовать линию – отрезок, соединяющий точки (350, 30) и (120, 190).
4. Нарисовать красную линию между точками (20, 300) и (420, 90).
5. Нарисовать красную линию толщиной 3 из точки (10,300) в точку (300,300).
6. Создать на холсте подобное изображение:
   1. «Домик»

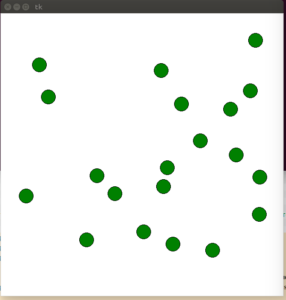


*Примечание: для создания травы используется цикл.*

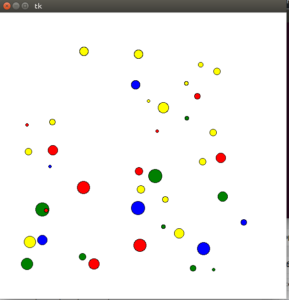
* 1. «Корона»



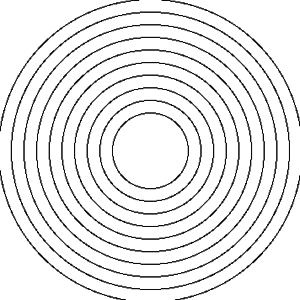
* 1. «Сосуд с молекулами газа» – 20 кругов в случайных местах

[](http://progras.ru/wp-content/uploads/2016/07/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA-%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0-%D0%BE%D1%82-2016-10-09-22-18-50.png)

* 1. «Конфетти» – 40 кругов в случайных местах случайного размера, случайного цвета

[](http://progras.ru/wp-content/uploads/2016/07/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA-%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0-%D0%BE%D1%82-2016-10-09-22-21-20.png)

* 1. «Жесткий диск» – 10 концентрических окружностей

[](http://progras.ru/wp-content/uploads/2016/07/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA-%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0-%D0%BE%D1%82-2016-10-09-22-31-41.png)

* 1. «Улица» – несколько домов в ряд

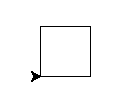
[](http://progras.ru/wp-content/uploads/2016/07/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA-%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0-%D0%BE%D1%82-2016-10-25-14-56-26.png)

* 1. Нарисовать таблицу со случайно выбранными числами из последовательности: 2, 4, 8, 16, 32, 64, 128, 256, 512. Каждому числу должен соответствовать свой цвет.

[](http://progras.ru/wp-content/uploads/2016/07/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA-%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0-%D0%BE%D1%82-2016-10-17-02-05-51.png)

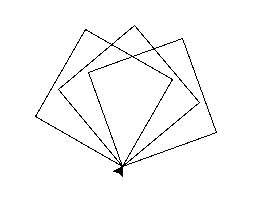
### Создание изображения графикой Turtle

1. Нарисовать квадрат с длиной стороны 150:



Угол квадрата – прямой, то есть равен 90 градусов.

1. Нарисовать несколько квадратов, с поворотом относительно друг друга:

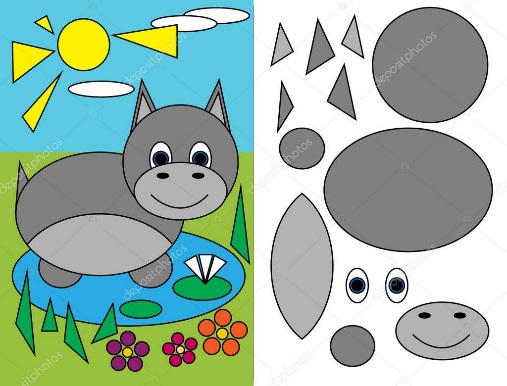


На изображении показан поворот черепашки на 20 градусов при переходе к следующему квадрату. Попробуйте задавать разные углы, например, 30 или 40.

1. Построить изображение «домика» (квадрат под треугольником) *без подъёма пера* при условии однократного перемещения по каждой линии.

### Создание анимации

1. Нарисовать и создать анимацию картинок:
2. Носорог



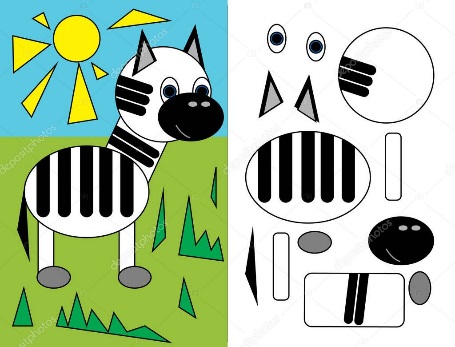
1. Тигр



1. Медведь



1. Зебра



# Список источников

1. Tkinter. Программирование GUI на Python. Курс // Сайт «Лаборатория линуксоида». – [Электронный ресурс]. – Режим доступа: https://younglinux.info/tkinter/course
2. Вечтомов, Е. М. Компьютерная геометрия: геометрические основы компьютерной графики : Учебное пособие для СПО / Е. М. Вечтомов, Е. Н. Лубягина //. – 2-е изд. – Москва : Издательство Юрайт, 2020. – 157 с.
3. Дёмин, А. Ю. Основы компьютерной графики: учебное пособие / А.Ю. Дёмин; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2011. – 191 с.
4. Поляков, К. Ю. Программирование. Python. C++. Часть 2: учебное пособие / К. Ю. Поляков. – М. : БИНОМ. Лаборатория знаний, 2019. – 176 с. : ил. ISBN 978-5-9963-4135-1
5. Шеремет, Г. Г. Геометрические преобразования и фрактальная геометрия : учебник / Г. Г. Шеремет. – Пермь : Пермский государственный гуманитарно-педагогический университет, 2013. – 188 c. – Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. – URL: https://www.iprbookshop.ru/32031.html (дата обращения: 04.03.2023). – Режим доступа: для авторизир. Пользователей

*Учебное пособие*

***ГРАФИКА НА ЯЗЫКЕ PYTHON***

*Гончаренко Наталья Николаевна*

*Редько Евгения Владимировна*

*Отпечатано с авторского оригинал–макета*

МУНИЦИПАЛЬНОЕ АВТОНОМНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ГОРОДА ХАБАРОВСКА «ЛИЦЕЙ ИННОВАЦИОННЫХ ТЕХНОЛОГИЙ»

680000, г. Хабаровск, ул. Гоголя, 24

тел. (4212) 32–47–36

<http://lit.khv.ru>