# Лабораторная работа № 1. Введение. Знакомство с интегрированной средой Lazarus

## Задача

Создать простейший пользовательский интерфейс, для программы сложения двух чисел (), содержащий объекты Button, TextBox — для ввода значений переменных и и вывода значения , Label — для поясняющих надписей.

Написать процедуру обработки события — щелчок по кнопке — сложение двух чисел. Для этого необходимо создать процедуру обработки события (двойным щелчком по объекту кнопка), объявить переменные, ввести значение переменных и , вычислить значение переменной и вывести значение с на экран.

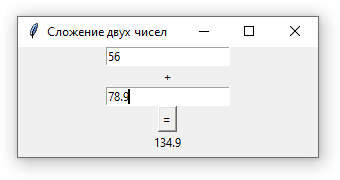
## Математическая модель

.

## Код программы

def button\_clicked():  
 try:  
 x = float(entry\_1.get())  
 y = float(entry\_2.get())  
 label\_sum.config(text=str(x + y))  
 except ValueError:  
 label\_sum.config(text="Ошибка. Введите цифры.")

## Результат



# Лабораторная работа № 2. Работа с графикой

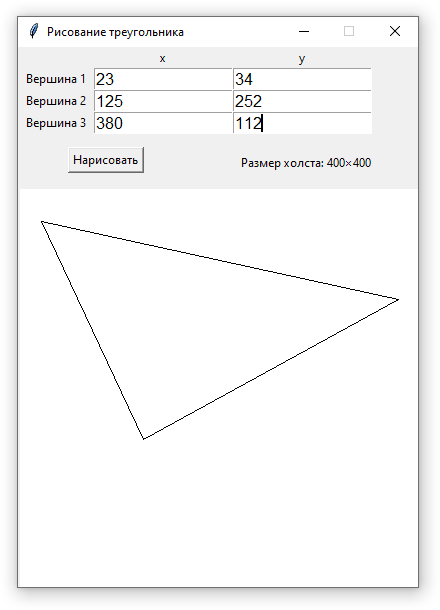
## Задача

Создать программу, рисующую треугольник, координаты вершин которого можно изменять через пользовательский интерфейс.

## Код программы

def draw():  
 cnvs.delete("all")  
 cnvs.create\_polygon(int(x1.get()), int(y1.get()), int(x2.get()), int(y2.get()),  
 int(x3.get()), int(y3.get()), fill='white', outline='black')

## Результат



# Лабораторная работа № 3. Построение графика заданной функции в указанном диапазоне

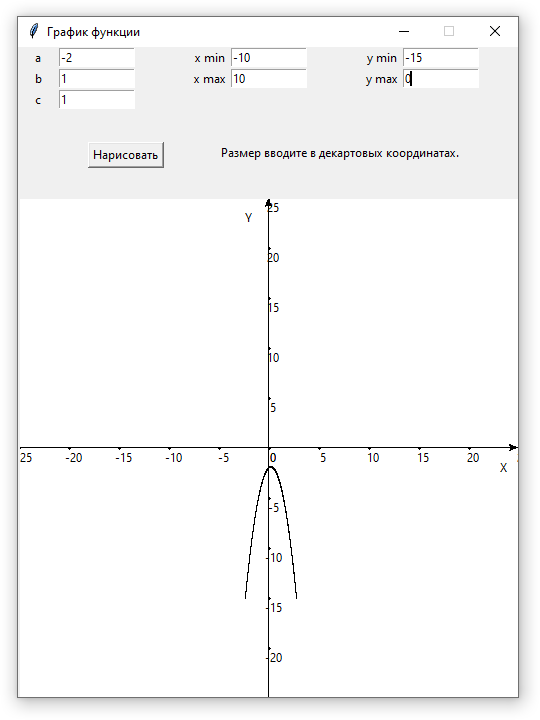
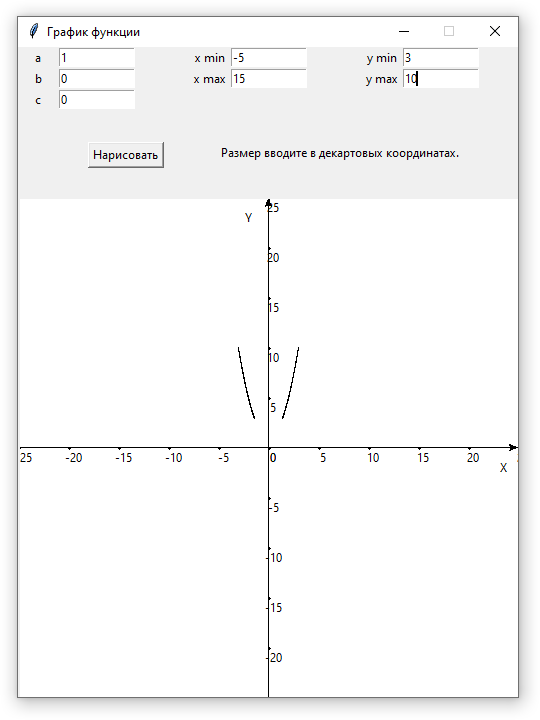
## Задача

Построить график функции в диапазоне (, )–(, ). Постоянные , , , , , , вводятся через пользовательский интерфейс.

## Код программы

WD = 500  
HG = 500  
ZR = 250  
KF = 10  
  
  
def func(x):  
 a = int(a\_const.get())  
 b = int(b\_const.get())  
 c = int(a\_const.get())  
 return a\*x\*x + b\*x + c  
  
  
def axis\_description():  
 canv.create\_text(WD-15, ZR+20, text='X')  
 canv.create\_text(ZR-20, 20, text='Y', )  
 vals = [i for i in range(-50, 51, 5)]  
 cords = [int(ZR + i\*KF) for i in vals]  
 for i in range(len(cords)):  
 canv.create\_oval(cords[i], ZR, cords[i]+2, ZR+2, fill='black')  
 canv.create\_text(cords[i]+5, ZR+10, text=str(vals[i]))  
 canv.create\_oval(ZR, cords[i], ZR+2, cords[i]+2, fill='black')  
 canv.create\_text(ZR+5, cords[i]+10, text=str(-vals[i]))  
  
  
  
def draw():  
 canv.delete("all")  
 canv.create\_line(0, 250, 500, 250, arrow=LAST)  
 canv.create\_line(250, 500, 250, 0, arrow=LAST)  
 axis\_description()  
 x\_mn = int(x\_min.get())  
 x\_mx = int(x\_max.get())  
 y\_mn = int(y\_min.get())  
 y\_mx = int(y\_max.get())  
 x = x\_mn  
 while x <= x\_mx:  
 xc = round(ZR + x\*KF)  
 yc = round(ZR - func(x)\*KF)  
 if ZR - y\_mn\*KF >= yc >= ZR - y\_mx\*KF:  
 canv.create\_oval(xc, yc, xc+1, yc+1)  
 x += 1/100

## Результат

# Лабораторная работа № 4. Движение тела, брошенного под углом к горизонту

## Задача

Изобразить местонахождение тела, брошенного с земли с начальной скорость под углом к горизонту , через равные интервалы времени ( – число равных временных интервалов).

## Математическая модель

Координаты произвольной точки траектории движения определяются уравнениями:

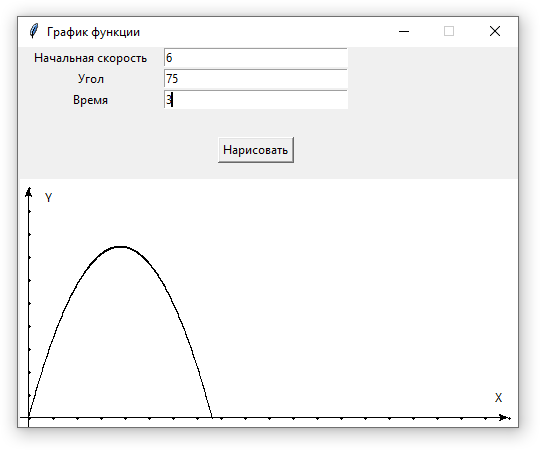
Для определения времени полета прировняем :

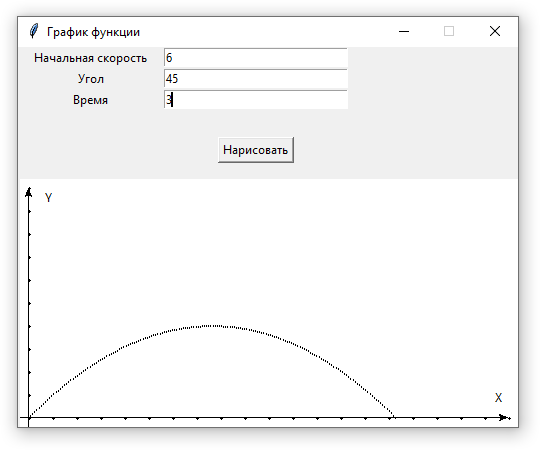
Тогда длительность временного интервала (шаг по ):

## Код программы

WD = 500  
HG = 250  
ZR\_X = 10  
ZR\_Y = 240  
g = 9.8  
  
  
def draw\_axis():  
 cnvs.create\_text(WD - 20, ZR\_Y - 20, text='X')  
 cnvs.create\_text(ZR\_X + 20, 20, text='Y')  
 x\_vals = [i/10 for i in range(21)]  
 y\_vals = [i/10 for i in range(11)]  
 x\_cords = [ZR\_X+x\_vals[i]\*240 for i in range(len(x\_vals))]  
 y\_cords = [ZR\_Y-y\_vals[i]\*230 for i in range(len(y\_vals))]  
 for i in range(len(x\_cords)):  
 cnvs.create\_oval(x\_cords[i], ZR\_Y, x\_cords[i] + 2, ZR\_Y + 2, fill='black')  
 for j in range(len(y\_cords)):  
 cnvs.create\_oval(ZR\_X, y\_cords[j], ZR\_X + 2, y\_cords[j] + 2, fill='black')  
  
  
def x\_func(t):  
 v = int(v0.get())  
 ang = int(angle.get()) \* math.pi/180  
 return v \* t \* math.cos(ang)  
  
  
def y\_func(t):  
 v = int(v0.get())  
 ang = int(angle.get()) \* math.pi/180  
 return v \* t \* math.sin(ang) - g \* t \* t / 2  
  
  
def draw():  
 cnvs.delete("all")  
 cnvs.create\_line(0, 240, 490, 240, arrow=LAST)  
 cnvs.create\_line(10, 250, 10, 10, arrow=LAST)  
 draw\_axis()  
 tm = float(time.get())  
 i = 0  
 xc = 0  
 yc = 0  
 while i <= tm:  
 xc = round(ZR\_X + x\_func(i)\*100)  
 yc = round(ZR\_Y - y\_func(i)\*100)  
 if yc <= 240:  
 cnvs.create\_oval(xc, yc, xc + 1, yc + 1)  
 i += tm/1000  
 print(xc)  
 print(yc)

## Результаты



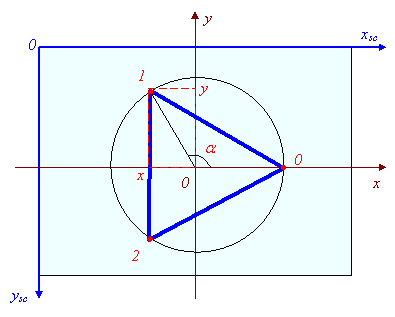


# Лабораторная работа № 5. Построение правильного n-угольника

## Задача

Построить правильный -угольник, количество углов которого задано и вводится через пользовательский интерфейс.

## Математическая модель



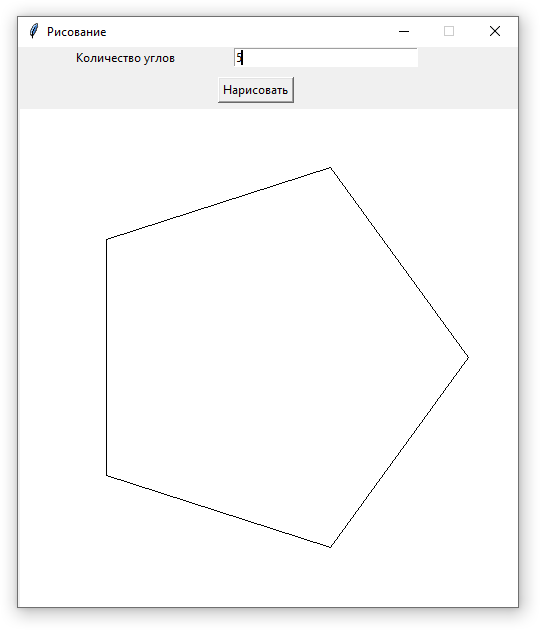
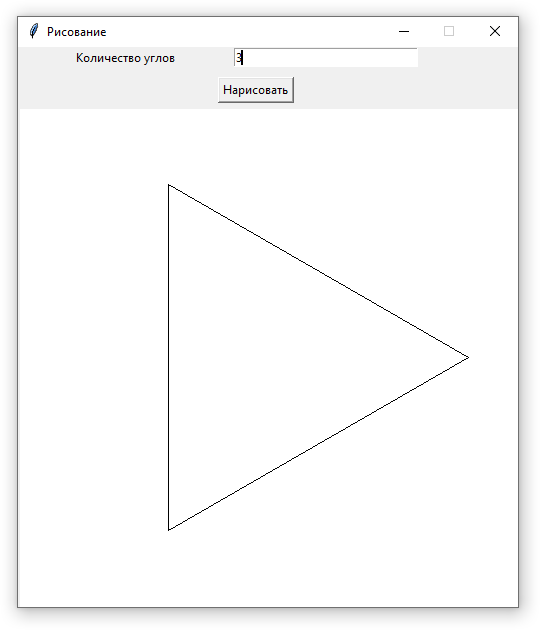
Пусть и — координаты центра, а — радиус описанной вокруг правильного многоугольника oкружности, — угловая координата первой вершины, тогда декартовы координаты вершин правильного — угольника определяются формулами:

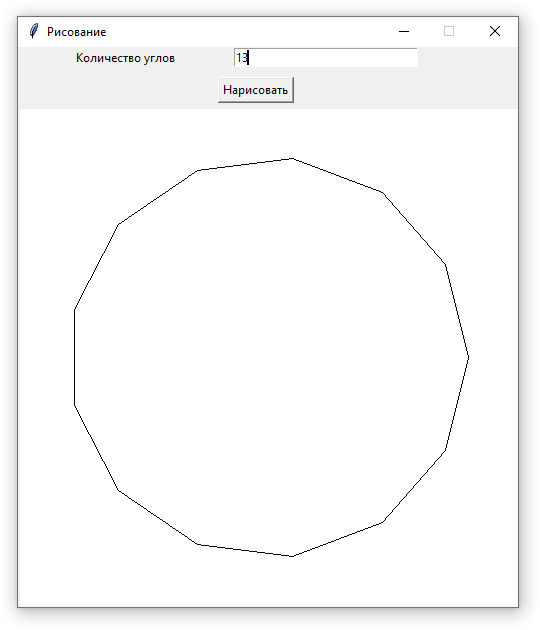
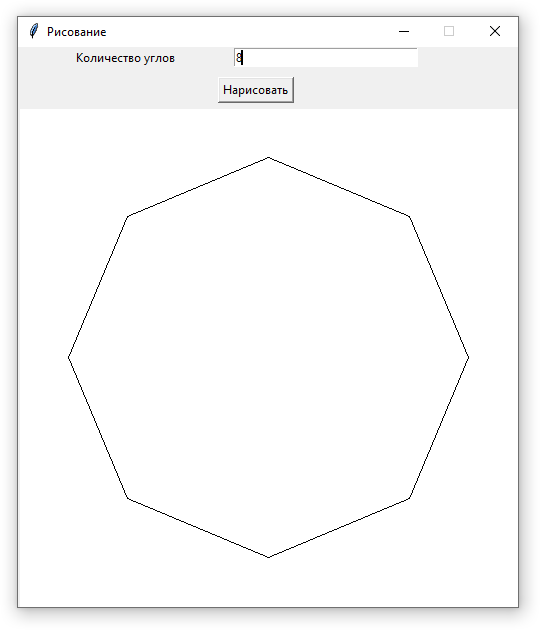
где и — координаты центра.

## Код программы

WD = 500  
ZR = 250  
PHI = 0  
RADIUS = 200  
  
  
def translation\_to\_radians(x):  
 return x \* math.pi / 180  
  
  
def draw():  
 cnvs.delete('all')  
 n = int(angl.get())  
 ang = translation\_to\_radians(PHI)  
 x0 = round(RADIUS \* math.cos(ang))  
 y0 = round(RADIUS \* math.sin(ang))  
 p1 = (x0 + ZR, ZR - y0)  
 for i in range(n + 1):  
 ang = PHI + 2 \* math.pi \* i / n  
 xi = round(RADIUS \* math.cos(ang))  
 yi = round(RADIUS \* math.sin(ang))  
 p2 = (xi + ZR, ZR - yi)  
 print(p1, p2)  
 cnvs.create\_line(p1, p2)  
 p1 = p2

## Результаты





# Лабораторная работа № 6. Преобразование объекта на плоскости

## Задача

Построить произвольный объект в первой четверти системы координат (например, флажок; количество вершин объекта должно быть не меньше 7) и организовать следующие преобразования объекта:

* общее преобразование (изменение масштаба, симметричное отражение относительно осей, сдвиг),
* вращение объекта относительно начала координат,
* комбинированное преобразование (преобразование общего вида + вращение на произвольный угол относительно начала координат).

Матрица общего преобразования и угол поворота должны вводиться через пользовательский интерфейс, координаты вершин объекта можно указать явно в коде программы.

## Математическая модель

## Код программы

WD = 500  
ZR = WD / 2  
KF = 10  
FLAG = [(5, 5), (5, 16), (10, 16), (7, 13), (10, 10), (6, 10), (6, 5)]  
  
  
def dec\_to\_scr(cords: tuple):  
 return cords[0] \* KF + ZR, ZR - cords[1] \* KF  
  
  
def axis():  
 cnvs.delete("all")  
 cnvs.create\_line(0, 250, 500, 250, arrow=LAST)  
 cnvs.create\_line(250, 500, 250, 0, arrow=LAST)  
 cnvs.create\_text(WD - 20, ZR + 20, text='X')  
 cnvs.create\_text(ZR - 20, 20, text='Y', )  
 vals = [i for i in range(-50, 51, 5)]  
 cords = [int(ZR + i\*KF) for i in vals]  
 for i in range(len(cords)):  
 cnvs.create\_oval(cords[i], ZR, cords[i] + 2, ZR + 2, fill='black')  
 cnvs.create\_text(cords[i] + 5, ZR + 10, text=str(vals[i]))  
 cnvs.create\_oval(ZR, cords[i], ZR + 2, cords[i] + 2, fill='black')  
 cnvs.create\_text(ZR + 5, cords[i] + 10, text=str(-vals[i]))  
  
  
def transl\_to\_rad(x):  
 return x \* math.pi / 180  
  
  
def transfer():  
 axis()  
 if int(angle.get()) == 0:  
 matrix = ((float(a11.get()), float(a12.get())),  
 (float(a21.get()), float(a22.get())))  
 else:  
 matrix = ((math.cos(transl\_to\_rad(int(angle.get()))), math.sin(transl\_to\_rad(int(angle.get())))),  
 (-math.sin(transl\_to\_rad(int(angle.get()))), math.cos(transl\_to\_rad(int(angle.get())))))  
 new\_flag = list()  
 for each in FLAG:  
 new\_cords = (each[0]\*matrix[0][0] + each[1]\*matrix[1][0],  
 each[0]\*matrix[0][1] + each[1]\*matrix[1][1])  
 new\_flag.append(new\_cords)  
 new\_flag\_scr = [dec\_to\_scr(i) for i in new\_flag]  
 cnvs.create\_polygon(new\_flag\_scr)

## Результаты

