Практикум по алгоритмизации и программированию на Python

Иван Хахаев, 2009

Часть 3. Графика в Python и задачи моделирования.

Python может работать с несколькими графическими библиотеками, обеспечивая создание сложных приложений с развитым графическим пользовательским интерфейсом. В этой части мы научимся пользоваться самыми простыми графическими возможностями Python — управлением исполнителем «черепашка» для создания графических примитивов и перемещения на плоскости и использованием библиотеки Tkinter для задач моделирования математических функций и физических явлений.

Управление исполнителем «черепашка»

Исполнитель «черепашка» управляется командами относительных («вперёд-назад» и «направо-налево») и абсолютных («перейти в точку с координатами...») перемещений. Исполнитель представляет собой «перо», оставляющее след на плоскости рисования. Перо можно поднять, тогда при перемещении след оставаться не будет. Кроме того, для пера можно установить толщину и цвет. Все эти функции исполнителя обеспечиваются модулем `turtle` («черепаха»).

Приведённый ниже код создаёт графическое окно (рис. 1) и помещает перо («черепашку») в исходное положение.

```
import turtle
# Инициализация
turtle.reset()
# Здесь могут быть вычисления и команды рисования
turtle._root.mainloop()
# Эта команда показывает окно, пока его не закроют
```

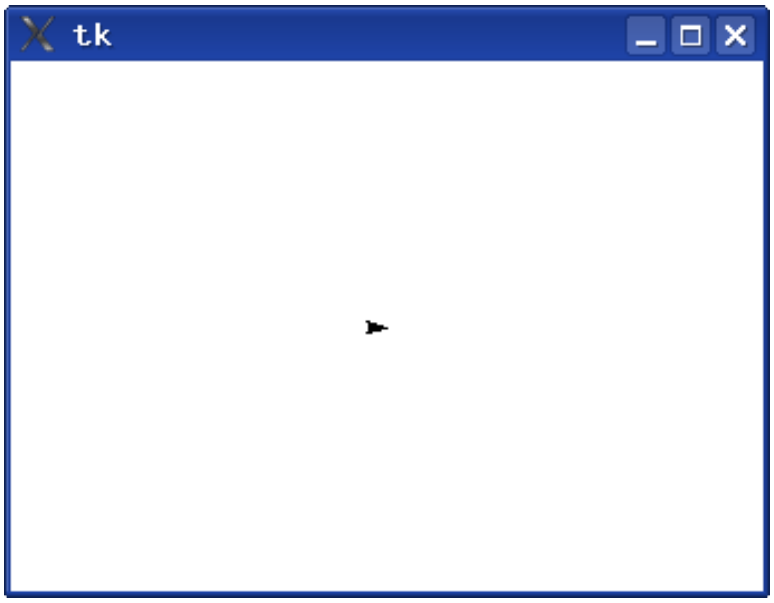

Рисунок 1. Окно рисование модуля `turtle`

Полученное окно имеет фиксированный размер, который зависит от версии Python, перо позиционируется в центре. Идея рисования заключается в перемещении пера («черепашки») в точки окна рисования с указанными координатами или в указанных направлениях на заданные расстояния, а также в проведении отрезков прямых, дуг и окружностей.

Текущее направление перемещение пера (соответствующее направлению «вперёд») указывается острием стрелки изображения «черепашки».

Полный список команд управления «черепашкой» (и, соответственно, рисования), а также функций, обеспечиваемых модулем, можно получить, набрав в окне выполнения любой системы программирования на Python команду `help('turtle')`.

Список этот довольно длинный, а среди предоставляемых функций имеются также математические, поскольку они могут быть востребованы при вычислении параметров отрезков, дуг и окружностей.

Команды, обеспечивающие рисование, приведены ниже.

Команда	Назначение	Пример
`up()`	Поднятие «пера», чтобы не оставалось следа его при перемещении	`turtle.up()`
`down()`	Опускание «пера», чтобы при перемещении оставался след (рисовались линии)	`turtle.down()`

Команда	Назначение	Пример
`goto(x,y)`	Перемещение «пера» в точку с координатами `x,y` в системе координат окна рисования	`turtle.goto(50,20)`
`color('строка_цвета')`	Установка цвета «пера» в значение, определяемое строкой цвета	`turtle.color('blue')` `turtle.color('#0000ff')`
`width(n)`	Установка толщины «пера» в точках экрана	`turtle.width(3)`
`forward(n)`	Передвижение «вперёд» (в направлении острия стрелки, см. рис. 2) на `n` точек	`turtle.forward(100)`
`backward(n)`	Передвижение «назад» на `n` точек	`turtle.backward(100)`
`right(k)`	Поворот направо (по часовой стрелке) на `k` единиц	`turtle.right(75)`
`left(k)`	Поворот налево (против часовой стрелки) на `k` единиц	`turtle.left(45)`
`radians()`	Установка единиц измерения углов в радианы	`turtle.radians()`
`degrees()`	Установка единиц измерения углов в градусы (включён по умолчанию)	`turtle.degrees()`
`circle(r)`	Рисование окружности радиусом `|r|` точек из текущей позиции «пера». Если `r` положительно, окружность рисуется против часовой стрелки, если отрицательно — по часовой стрелке.	`turtle.circle(40)` `turtle.circle(-50)`

Команда	Назначение	Пример
`circle(r,k)`	Рисование дуги радиусом `|r|` точек и углом `k` единиц. Вариант команды `circle()`	`turtle.circle(40,45)` `turtle.circle(-50,275)`

`fill(flag)`	В зависимости от значения `flag` включается `(flag=1)` и выключается `(flag=0)` режим закрашивания областей. По умолчанию выключен.	Круг:
		`turtle.fill(1)`
		`turtle.circle(-50)`
		`turtle.fill(0)`

`write('строка')`	Вывод текста в текущей позиции пера	`turtle.write('Начало координат!')`
`tracer(flag)`	Включение `(flag=1)` и выключение `(flag=0)` режима отображения указателя «пера» («черепашки»). По умолчанию включён.	`turtle.tracer(0)`
`clear()`	Очистка области рисования	`turtle.clear(0)`

При выключенном режиме отображения указателя «черепашки» рисование происходит значительно быстрее, чем при включённом.

Нужно заметить, что хотя углы поворота исполнителя изначально интерпретируются в градусах, при использовании тригонометрических функций модуля `turtle` (например, `turtle.sin()`) аргументы этих функций воспринимаются как радианы.

Проделаем упражнение с целью определить систему координат окна рисования. Приведённый ниже код формирует картинку, показанную на рис. 2.

```
# -*- coding: utf-8 -*-
import turtle
#
turtle.reset()
turtle.tracer(0)
turtle.color('#0000ff')
#
turtle.write('0,0')
#
turtle.up()
x=-170
```

```
y=-120
coords=str(x)+","+str(y)
turtle.goto(x,y)
turtle.write(coords)
#
x=130
y=100
coords=str(x)+","+str(y)
turtle.goto(x,y)
turtle.write(coords)
#
x=0
y=-100
coords=str(x)+","+str(y)
turtle.goto(x,y)
turtle.write(coords)
#
turtle.down()
x=0
y=100
coords=str(x)+","+str(y)
turtle.goto(x,y)
turtle.write(coords)
#
turtle.up()
x=-150
y=0
coords=str(x)+","+str(y)
turtle.goto(x,y)
turtle.write(coords)
#
turtle.down()
x=150
y=0
coords=str(x)+","+str(y)
turtle.goto(x,y)
turtle.write(coords)
#
turtle._root.mainloop()
```

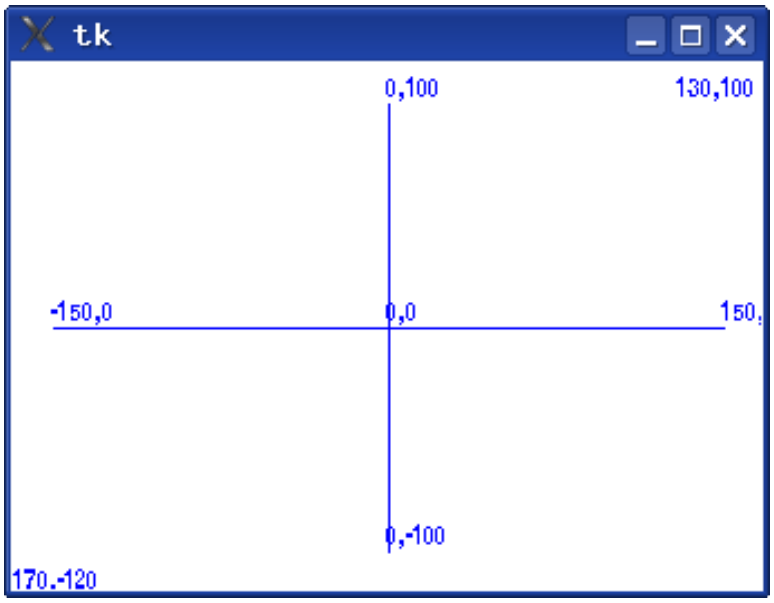

Рисунок 2. Система координат окна рисования

Здесь строка с координатами формируется «в лоб», путём конкатенации преобразованных в строки значений координат.

Картинка, показанная на рис. 3, сформирована нижеследующим кодом.

```
# -*- coding: utf-8 -*-
import turtle
#
turtle.reset()
turtle.tracer(0)
turtle.width(2)
#
turtle.up()
x=0
y=-100
turtle.goto(x,y)
turtle.fill(1)
turtle.color('#ffaa00')
turtle.down()
turtle.circle(100)
turtle.fill(0)
turtle.color('black')
turtle.circle(100)
turtle.up()
#
x=-45
y=50
turtle.goto(x,y)
turtle.down()
turtle.color('#0000aa')
turtle.fill(1)
```

```
turtle.circle(7)
turtle.up()
turtle.fill(0)
#
x=45
y=50
turtle.goto(x,y)
turtle.down()
turtle.color('#0000aa')
turtle.fill(1)
turtle.circle(7)
turtle.up()
turtle.fill(0)
#
x=-55
y=-50
turtle.goto(x,y)
turtle.right(45)
turtle.width(3)
turtle.down()
turtle.color('#aa0000')
turtle.circle(80,90)
turtle.up()
#
turtle.right(135)
x=0
y=50
turtle.goto(x,y)
turtle.width(2)
turtle.color('black')
turtle.down()
turtle.forward(100)
#
turtle._root.mainloop()
```

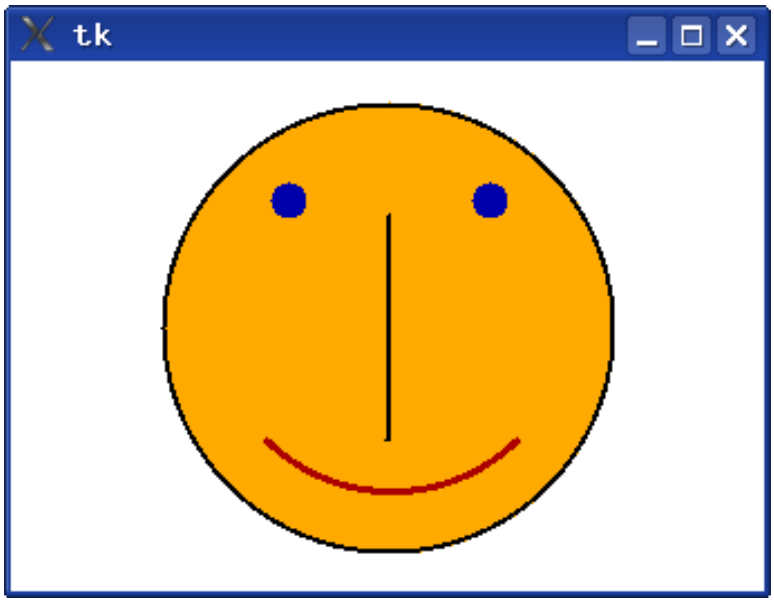

Рисунок 3. Пример формирования изображения

Для того, чтобы изобразить «улыбку», потребовалось после перемещения «пера» в начальную точку дуги (левую) повернуть «перо» на 45 градусов. Дело в том, что изначально направлением «вперёд» для «пера» является направление вправо (как показано на рис. 1). Окружности и дуги рисуются как касательные к этому «вектору», начинаясь в точке с текущими координатами «пера». Поэтому для «улыбки» потребовалось изменить направление «вектора».

Далее, «перо», первоначально сориентированное на 45 градусов вправо после прохождения дуги в 90 градусов соответственно изменило своё «направление». Поэтому для получения вертикальной линии его ещё пришлось «довернуть».

Можно поэкспериментировать с рисованием домиков, «солнышка» и более сложных композиций. Однако для формирования сложных кривых (например, графиков функций) с помощью этого модуля придётся многократно выполнять команду `goto(x,y)`. В этом легко убедиться попытавшись нарисовать, например, график параболы.

Задания и упражнения

1. Как в примерах кода, формирующего изображения на рис. 2 и 3 применить кортежи?
2. Напишите код для создания изображения «домика» (квадрат под треугольником) без подъёма пера при условии однократного перемещения по каждой линии.
3. Рассчитайте координаты и напишите код для создания изображения «солнца» (круг и расходящиеся от него отрезки) так, чтобы «лучи»

начинались на расстоянии 2 точки от круга (не менее 8-ми лучей).

4. Напишите код для построения графика степенной функции (`y=A*x^B`) с началом координат в левой нижней четверти окна рисования так, чтобы кривая проходила практически через всё окно.

Пользовательские подпрограммы и моделирование. Библиотека Tkinter.

Гораздо более серьёзными возможностями, чем библиотека `turtle`, в Python обладает библиотека `Tkinter`. Эта библиотека предназначена для организации графических интерфейсов (GUI — Graphical User Interface) программ на Python, но благодаря наличию элемента графического интерфейса (или, как говорят, «виджета») `canvas` («холст») в `Tkinter` можно использовать элементы векторной графики — кривые, дуги, эллипсы, прямоугольники и пр., а также рисовать прямые и кривые по координатам, рассчитанным по формулам.

Рассмотрим задачу построения графика некоторой функции по вычисляемым точкам с помощью `Tkinter`.

Поскольку `Tkinter` позволяет работать с элементами GUI, создадим окно заданного размера, установим для него заголовок и цвет фона «холста», а также снабдим окно программной «кнопкой». На «холсте» определим систему координат и нарисуем «косинусоиду».

```
# -*- coding: utf-8 -*-
import Tkinter
import math
#
tk=Tkinter.Tk()
tk.title("Sample")
#
button=Tkinter.Button(tk)
button["text"]="Закрыть"
button["command"]=tk.quit
button.pack()
#
canvas=Tkinter.Canvas(tk)
canvas["height"]=360
canvas["width"]=480
canvas["background"]="#eeeeff"
canvas["borderwidth"]=2
canvas.pack()
#
canvas.create_text(20,10,text="20,10")
canvas.create_text(460,350,text="460,350")
#
points=[]
```

```
ay=150
y0=150
x0=50
x1=470
dx=10
#
for n in range(x0,x1,dx):
    y=y0-ay*math.cos(n*dx)
    pp=(n,y)
    points.append(pp)
#
canvas.create_line(points,fill="blue",smooth=1)
#
y_axe=[]
yy=(x0,0)
y_axe.append(yy)
yy=(x0,y0+ay)
y_axe.append(yy)
canvas.create_line(y_axe,fill="black",width=2)
#
x_axe=[]
xx=(x0,y0)
x_axe.append(xx)
xx=(x1,y0)
x_axe.append(xx)
canvas.create_line(x_axe,fill="black",width=2)
#
tk.mainloop()
```

Посмотрим на результат (рис. 4), и разберём текст примера.

Итак, первые три строки программы понятны — устанавливается кодовая страница и подключаются библиотеки. Поскольку в примере должны использоваться тригонометрические функции, необходима библиотека `math`.

Затем создаётся так называемое «корневое» окно — говоря научным языком, «экземпляр интерфейсного объекта `Tk`», который представляет собой просто окно без содержимого.

Затем для этого окна устанавливается значение свойства `title` (создаётся заголовок).

Далее начинается заполнение окна интерфейсными объектами («виджетами» - widgets). В данном примере используется два объекта — кнопка и «холст». Для размещения объекта в окне используется функция `pack()`, а порядок объектов определяется порядком выполнения этой функции.

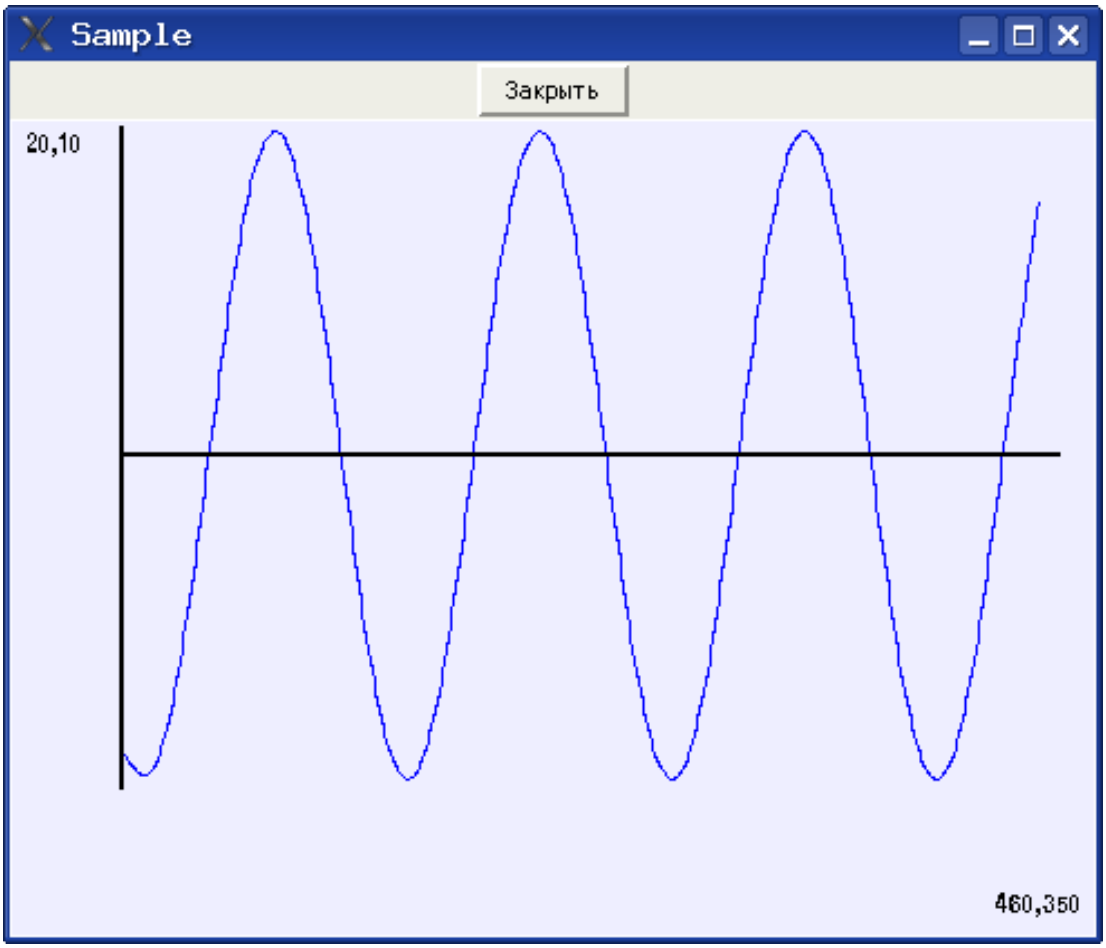

Рисунок 4. Окно `Tkinter` с кнопкой и графиком

Кнопка создаётся как экземпляр объекта `Button` библиотеки `Tkinter`, связанный с «корневым» окном. Для кнопки можно установить текст надписи (атрибут «`text`») и связать кнопку с выполнением какой-либо команды (функции или процедуры) с помощью атрибута «`command`».

В приведённом примере кнопка связана с командой закрытия окна и прекращения работы интерпретатора, однако ничто не мешает также закрывать окно нашего «приложения» обычным образом — с помощью стандартной кнопки закрытия окна в верхнем правом углу.

После создания и размещения кнопки создаётся «холст». Для элемента `canvas` указываются высота, ширина, цвет фона и отступ от границ окна (таким образом, окно получается несколько больше, чем элемент `canvas`). Размеры окна автоматически подстраиваются так, чтобы обеспечить размещение всех элементов.

После размещение виджета `canvas` в окне исследуем систему координат. Поскольку размеры окна уже нами заданы, полезно определить, где находится точка с координатами `(0,0)`. Как видно из попыток вывести значения координат с помощью функции `canvas.create_text()`, начало координат находится в

верхнем левом углу «холста».

Теперь, определившись с координатами, можно выбрать масштабные коэффициенты и сдвиги и сформировать координаты точек для рисования кривой.

Для функции `canvas.create_line()` в качестве координат требуется список пар точек (кортежей) `(x,y)`. Этот список формируется в цикле с шагом `dx`.

Для линии графика устанавливаются цвет и режим сглаживания. Это сглаживание обеспечивает некоторую «плавность» кривой. Если его убрать, линия будет состоять из отрезков прямых. Кроме того, для линий можно устанавливать толщину, как это показано для «осей координат».

Моделирование математических функций

Пусть требуется построить график функции, вид которой выбирается из списка. Здесь потребуется уже использование дополнительных интерфейсных элементов библиотеки `Tkinter`, а также создание собственных (пользовательских) процедур или функций для облегчения понимания кода.

Результат решения задачи (вариант внешнего вида «приложения») показан на рис. 5.

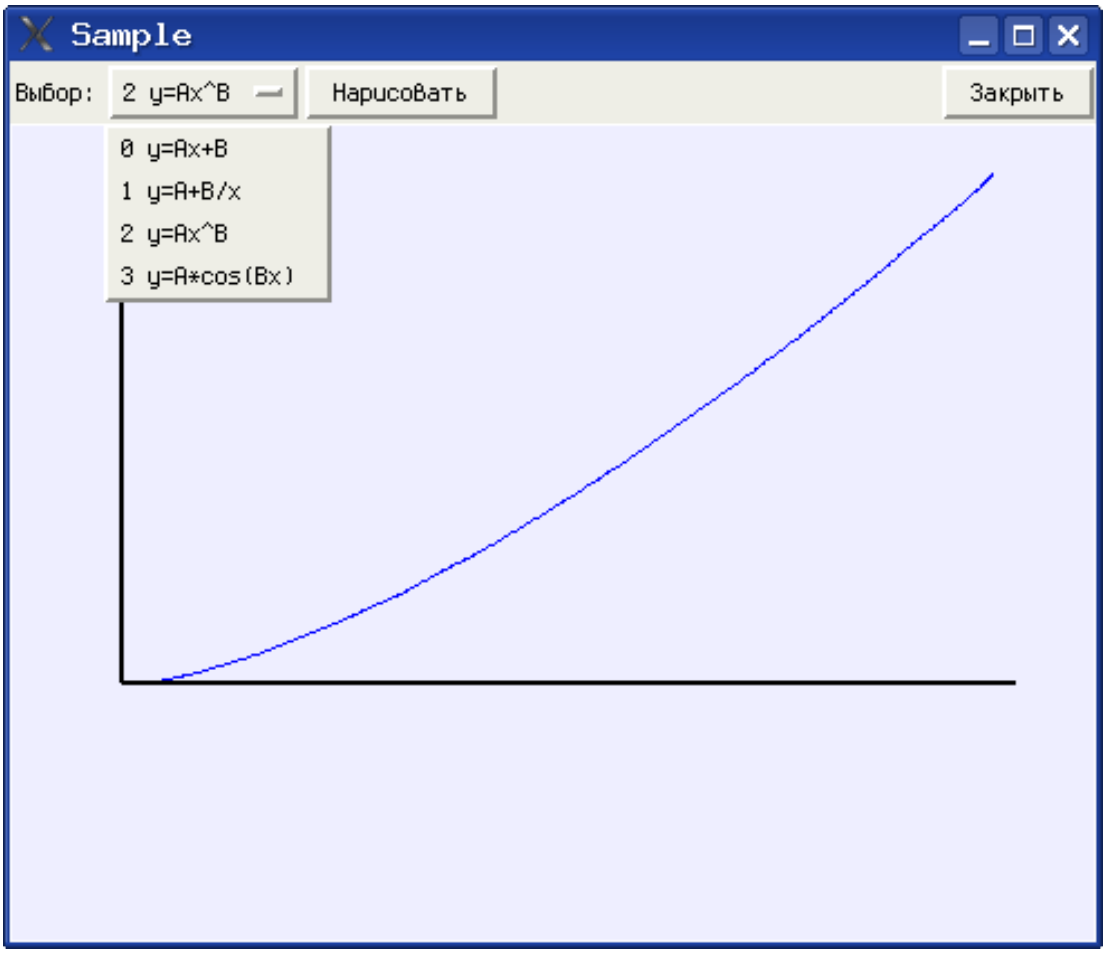

Рисунок 5. Построение графика для функции, выбираемой из списка

Для выбора вида математической функции используется раскрывающийся список, после выбора вида функции и нажатия на кнопку «Нарисовать» на «холсте» схематически рисуется график этой функции. Кнопка «Закрыть» закрывает «приложение».

Теперь посмотрим на код, который формирует такое «приложение» .

```
# -*- coding: utf-8 -*-
import Tkinter
import math
#
# Пользовательские процедуры
def plot_x_axe(x0,y0,x1):
    x_axe=[]
    xx=(x0,y0)
    x_axe.append(xx)
    xx=(x1,y0)
    x_axe.append(xx)
    canvas.create_line(x_axe,fill="black",width=2)
#
```

```
def plot_y_axe(x0,y0,y1):
    y_axe=[]
    yy=(x0,y1)
    y_axe.append(yy)
    yy=(x0,y0)
    y_axe.append(yy)
    canvas.create_line(y_axe,fill="black",width=2)
#
def plot_func0(x0,x1,dx,y0,y1):
    x0i=int(x0)
    x1i=int(x1)
    y0i=int(y0)
    y1i=int(y1)
    a=y1
    b=(y0-y1)/(x1-x0)
    points=[]
    for x in range(x0i,x1i,dx):
        y=int(a+b*x)
        pp=(x,y)
        points.append(pp)
    #
    canvas.create_line(points,fill="blue",smooth=1)
    plot_y_axe(x0i,y0i,y1i)
    plot_x_axe(x0i,y0i,x1i)
#
def plot_func1(x0,x1,dx,y0,y1):
    x0i=int(x0)
    x1i=int(x1)
    y0i=int(y0)
    y1i=int(y1)
    a=y0
    b=y0-y1
    points=[]
    for x in range(x0i,x1i,dx):
        y=int(a-y1i*b/x)
        pp=(x,y)
        points.append(pp)
    #
    canvas.create_line(points,fill="blue",smooth=1)
    plot_y_axe(x0i,y0i,y1i)
    plot_x_axe(x0i,y0i,x1i)
#
def plot_func2(x0,x1,dx,y0,y1):
    x0i=int(x0)
    x1i=int(x1)
    y0i=int(y0)
    y1i=int(y1)
    a=(y0-y1)/(15*x1)
    b=1+((y0-y1)/(x1-x0))
    points=[]
```

```
    for x in range(x0i,x1i,dx):
        y=y0i-int(a*(x-x0i)**b)
        pp=(x,y)
        points.append(pp)
    #
    canvas.create_line(points,fill="blue",smooth=1)
    plot_y_axe(x0i,y0i,y1i)
    plot_x_axe(x0i,y0i,x1i)
#
def plot_func3(x0,x1,dx,y0,y1):
    x0i=int(x0)
    x1i=int(x1)
    y0i=int(y0)
    y1i=int(y1)
    ay=150
    y0i=150
    points=[]
    for x in range(x0i,x1i,dx):
        y=y0i-ay*math.cos(x*dx)
        pp=(x,y)
        points.append(pp)
    #
    canvas.create_line(points,fill="blue",smooth=1)
    plot_y_axe(x0i,0,y0i+ay)
    plot_x_axe(x0i,y0i,x1i)
#
def DrawGraph():
    fn=func.get()
    f=fn[0]
    x0=50.0
    y0=250.0
    x1=450.0
    y1=50.0
    dx=10
    #
    if f=="0":
        canvas.delete("all")
        plot_func0(x0,x1,dx,y0,y1)
    elif f=="1":
        canvas.delete("all")
        plot_func1(x0,x1,dx,y0,y1)
    elif f=="2":
        canvas.delete("all")
        plot_func2(x0,x1,dx,y0,y1)
    else:
        canvas.delete("all")
        plot_func3(x0,x1,dx,y0,y1)
#
# Основная часть
tk=Tkinter.Tk()
```

```
tk.title("Sample")
# Верхняя часть окна со списком и кнопками
menuframe=Tkinter.Frame(tk)
menuframe.pack({"side":"top","fill":"x"})
# Надпись для списка
lbl=Tkinter.Label(menuframe)
lbl["text"]="Выбор:"
lbl.pack({"side":"left"})
# Инициализация и формирования списка
func=Tkinter.StringVar(tk)
func.set('0 y=Ax+B')
#
fspis=Tkinter.OptionMenu(menuframe,func,
    '0 y=Ax+B',
    '1 y=A+B/x',
    '2 y=Ax^B',
    '3 y=A*cos(Bx)')
fspis.pack({"side":"left"})
# Кнопка управления рисованием
btnOk=Tkinter.Button(menuframe)
btnOk["text"]="Нарисовать"
btnOk["command"]=DrawGraph
btnOk.pack({"side":"left"})
# Кнопка закрытия приложения
button=Tkinter.Button(menuframe)
button["text"]="Закрыть"
button["command"]=tk.quit
button.pack({"side":"right"})
# Область рисования (холст)
canvas=Tkinter.Canvas(tk)
canvas["height"]=360
canvas["width"]=480
canvas["background"]="#eeeeff"
canvas["borderwidth"]=2
canvas.pack({"side":"bottom"})

tk.mainloop()
```

Основная часть программы (интерфейсная) начинается с момента создания корневого окна (инструкция `tk=Tkinter.Tk()`). В этом окне располагаются два интерфейсных элемента — рамка (`Frame`) и холст (`canvas`). Рамка является «контейнером» для остальных интерфейсных элементов — текстовой надписи (метки — `Label`), раскрывающегося списка вариантов (`OptionMenu`) и двух кнопок. Как видно, кнопка закрытия стала объектом рамки, а не корневого окна, но по-прежнему закрывает всё окно.

Для получения нужного расположения элементов метод `pack()` используется с указанием, как именно размещать элементы интерфейса (к какой

стороне элемента-контейнера их нужно «прижимать»).

Есть некоторые тонкости в создании раскрывающегося списка. Для успешного выполнения этой операции нужно предварительно сформировать строку (а точнее, объект `Tkinter.StringVar()`) и определить для этого объекта значение по умолчанию (это значение показывается в только что запущенном приложении). Затем в элементе `OptionMenu()` список значений дополняется. При выборе элемента списка изменяется значение именно этой строки и для дальнейшей работы нужно его анализировать, что и делается в процедуре `DrawGraph()`.

«Вычислительная» часть, а именно, все процедуры и функции, обеспечивающие вычисления координат точек и рисование линий, вынесена в начало текста программы.

Определение каждой пользовательской процедуры или функции обеспечивается составным оператором `def`. Поскольку функции занимаются только рисованием, они не возвращают никаких значений (т.е. результаты выполнения этих функций не присваиваются никаким переменным).

Процедура собственно рисования графика `DrawGraph()` вызывается при нажатии кнопки «Нарисовать», и имя этой функции является командой, которая сопоставляется кнопке.

Эта процедура берёт значение из списка (метод `get()`), выбирает первый символ получившейся строки и в зависимости от этого символа вызывает другие процедуры и функции для построения конкретных графиков с установленными масштабными коэффициентами.

Перед рисованием следующего графика математической функции холст очищается командой `canvas.delete("all")`.

Для построения графика каждой функции вычисляются собственные масштабные коэффициенты, поэтому их вычисление включено в код соответствующей процедуры. Кроме того, для графика нужны целые значения координат, поэтому в каждой процедуре выполняются соответствующие преобразования с помощью функции `int()`.

Для каждого графика требуется нарисовать оси, и действия по рисованию осей также вынесены в отдельные процедуры.

Таким образом, оказывается, что программу нужно читать «с конца», и писать тоже.

Моделирование физического явления: тело, брошенное под углом к горизонту

Теперь создадим небольшое приложение для моделирования движения тела, брошенного под углом к горизонту. Физическая задача формулируется следующим

образом:

Тело брошено под углом α к горизонту с начальной высоты $h_0=0$ и с начальной скоростью v_0. Определить величину угла α, при которой дальность полёта тела будет максимальной. Сопротивлением воздуха пренебречь.

Основные обозначения для решения задачи показаны на рис. 6.

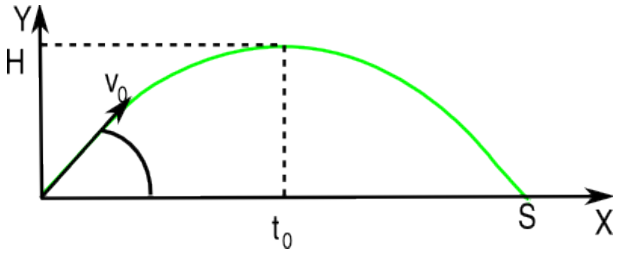

Рисунок 6. Обозначения для задачи о теле, брошенном под углом к горизонту

Напишем формулы, по которым определяются координаты тела x и y в зависимости от времени.

$$x(t)=v_{гор}\cdot t=v_0\cdot\cos(\alpha)\cdot t \tag{1}$$

$$y(t)=v_{верт}\cdot t=v_0\cdot\sin(\alpha)\cdot t-g\cdot t^2 \tag{2}$$

Выразив время через координату x (на основании формулы 1)

$$t=\frac{x}{v_0\cdot\cos(\alpha)} \tag{3}$$

и подставив выражение для времени в формулу для координаты y, получим уравнение траектории $y(x)$:

$$y(x)=x\cdot tg(\alpha)-x^2\cdot\frac{g}{2\cdot v_0^2\cdot\cos^2(\alpha)} \tag{4}$$

Поскольку сопротивление при движении тела отсутствует, горизонтальная составляющая скорости изменяться не будет, а изменение вертикальной

составляющей определяется влиянием ускорения свободного падения.

$$v_{гор}(t) = v_0 \cdot \cos(\alpha) \tag{5}$$

$$v_{верт}(t) = v_0 \cdot \sin(\alpha) - g \cdot t \tag{6}$$

Время t_0, через которое будет достигнута наивысшая точка траектории, найдём из условия $v_{верт}=0$.

$$t_0 = \frac{v_0 \cdot \sin(\alpha)}{g} \tag{7}$$

Максимальную высоту подъёма H найдём из уравнения вертикального движения (формула 2) в момент времени t_0.

$$H = y(t_0) = \frac{v_0^2 \cdot \sin^2(\alpha)}{2 \cdot g} \tag{8}$$

Полное время полёта T очевидно, равно $2t_0$, поэтому дальность полёта S определим как

$$S = v_{гор} \cdot T = v_0 \cdot \cos(\alpha) \cdot 2 \cdot t_0 = \frac{v_0^2 \cdot \sin(2 \cdot \alpha)}{g} \tag{9}$$

Все эти формулы понадобятся для вычисления координат точек траектории и параметров траектории при моделировании.

Текст программы с пользовательскими процедурами приведён ниже.

```
# -*- coding: utf-8 -*-
# Моделирование задачи о теле
# брошенном под углом к горизонту
#
import Tkinter
import math
#
def plot_x_axe(x0,y0,x1):
```

```
    x_axe=[]
    xx=(x0,y0)
    x_axe.append(xx)
    xx=(x1,y0)
    x_axe.append(xx)
    canvas.create_line(x_axe,fill="black",width=2)
#
def plot_y_axe(x0,y0,y1):
    y_axe=[]
    yy=(x0,y1)
    y_axe.append(yy)
    yy=(x0,y0)
    y_axe.append(yy)
    canvas.create_line(y_axe,fill="black",width=2)
#
def DrawGraph():
# Получаем и пересчитываем параметры
    dta=sc.get()
    alpha=dta*math.pi/180
    dtlbl=clist.get()
# Очищаем область для текста
    canvas.create_rectangle(x1i-90,y1i-
50,x1i+50,y1i+10,fill="#eeeeff")
# Считаем g=10, v0 подбираем, чтобы всё влезало в canvas
    g=10.0
    v0=63
#
    S=int((v0**2)*math.sin(2*alpha)/g)
    H=int(((v0**2)*(math.sin(alpha))**2)/(2*g))
#
    points=[]
    for x in range(x0i,x1i):
        xx=(x-x0)
        y=(xx*math.tan(alpha))-((xx**2)*g/
(2*(v0**2)*(math.cos(alpha)**2)))
    #
        if y > 0:
            yy=int(y0-y)
        else:
            yy=y0i
        #
        pp=(x,yy)
        points.append(pp)
# Собственно график
    canvas.create_line(points,fill=dtlbl,smooth=1)
    plot_x_axe(x0i,y0i,x1i)
# Параметры графика
    dtext="Дальность: "+str(S)
    vtext="Высота: "+str(H)
    dalnost=canvas.create_text(x1i-70,y1i-
```

```
30,text=dtext,fill=dtlbl,anchor="w")
    vysota=canvas.create_text(x1i-70,y1i-
10,text=vtext,fill=dtlbl,anchor="w")

#

# Основная часть
tk=Tkinter.Tk()
tk.title("Моделирование полёта")
# Верхняя часть окна со списком и кнопками
menuframe=Tkinter.Frame(tk)
menuframe.pack({"side":"top","fill":"x"})
# Надпись для списка
lbl=Tkinter.Label(menuframe)
lbl["text"]="Выбор цвета:"
lbl.pack({"side":"left"})
# Инициализация и формирование списка
clist=Tkinter.StringVar(tk)
clist.set('black')
#
cspis=Tkinter.OptionMenu(menuframe,clist,
    'red',
    'green',
    'blue',
    'cyan',
    'magenta',
    'purple',
    'black')
cspis.pack({"side":"left"})
# Кнопка управления рисованием
btnOk=Tkinter.Button(menuframe)
btnOk["text"]="Нарисовать"
btnOk["command"]=DrawGraph
btnOk.pack({"side":"left"})
# Кнопка закрытия приложения
button=Tkinter.Button(menuframe)
button["text"]="Закрыть"
button["command"]=tk.quit
button.pack({"side":"right"})
#
# Надпись для шкалы углов
lbl2=Tkinter.Label(tk)
lbl2["text"]="Угол, градусы:"
lbl2.pack({"side":"top"})
# Шкала углов
sc=Tkinter.Scale(tk,from_=0,to=90,orient="horizontal")
sc.pack({"side":"top","fill":"x"})
#
# Область рисования (холст)
canvas=Tkinter.Canvas(tk)
```

```
canvas["height"]=360
canvas["width"]=480
canvas["background"]="#eeeeff"
canvas["borderwidth"]=2
canvas.pack({"side":"bottom"})
#
# Установки осей координат
x0=50.0
y0=300.0
x1=450.0
y1=50.0
#
x0i=int(x0)
x1i=int(x1)
y0i=int(y0)
y1i=int(y1)
# Оси координат
plot_x_axe(x0i,y0i,x1i)
plot_y_axe(x0i,y0i,y1i)
#

tk.mainloop()
```

Результат работы с моделью показан на рис. 7.

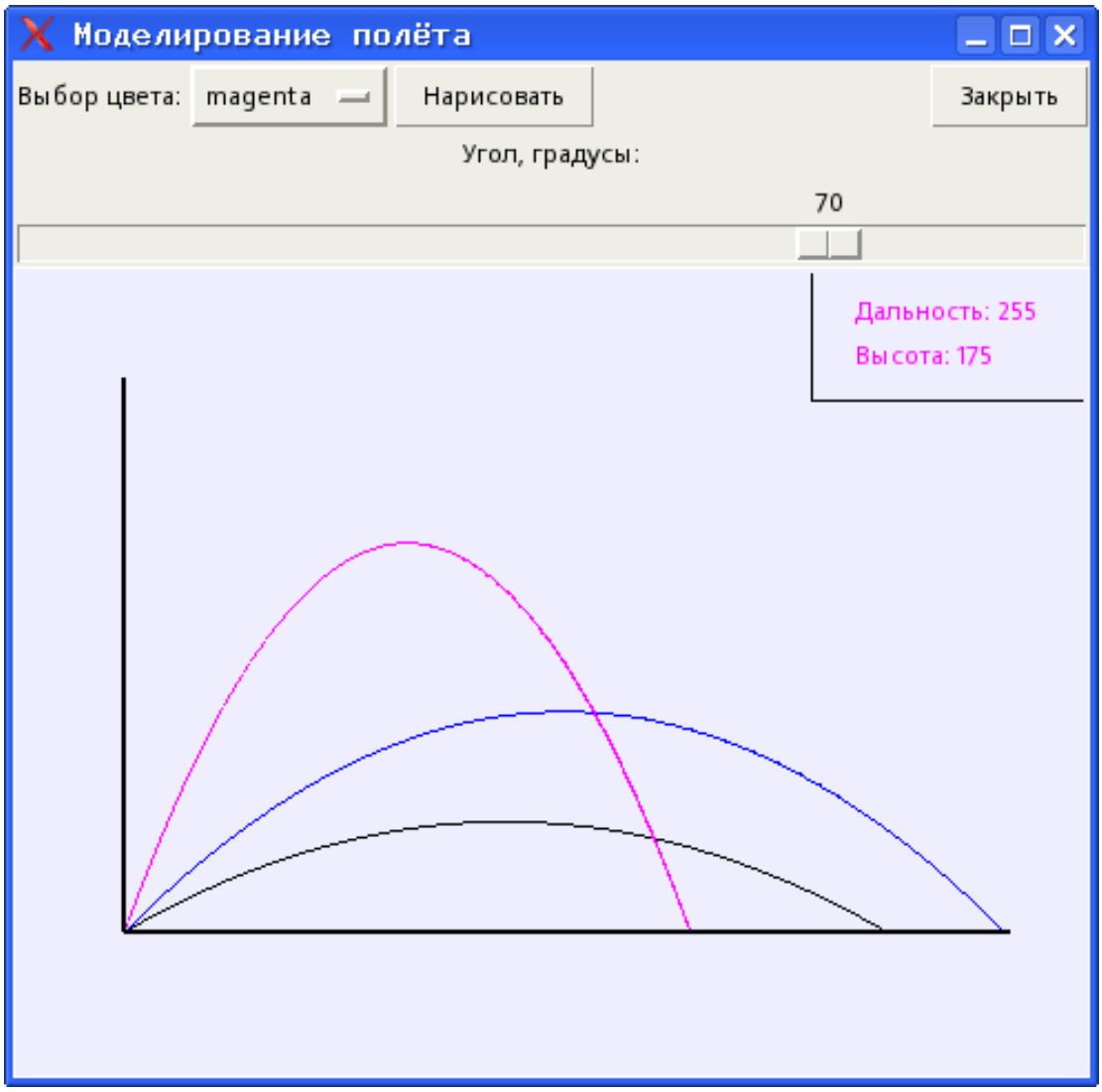

Рисунок 7. Поиск угла для достижения максимальной дальности на модели

Реализация модели имеет ряд особенностей. Во-первых, величина ускорения свободного падения (`g`) принята как `10`. Во-вторых, модуль начальной скорости выбран так, чтобы при любых значениях угла вся траектория попадала в область графика. Не совсем правильно с точки зрения принципа разделения программ и данных установка значений для `g` и v_0 прямо в коде, но такое решение значительно упрощает работу с моделью.

«Ползунок» на шкале установки углов показывает значения в градусах, а для правильных вычислений в тригонометрических функциях эти значения нужно перевести в радианы.

Высота и дальность «полёта» пишутся для каждой траектории соответствующим цветом в прямоугольнике в верхнем правом углу. Для каждой следующей траектории этот прямоугольник рисуется заново и текст

переписывается.

В этой главе рассмотрены некоторые базовые возможности библиотеки `Tkinter` и использования Python для создания моделей. Интересующиеся могут найти более подробную информацию на необъятных просторах Интернета.

Задачи и упражнения

1. Для примера, показанного на рис. 4 нанесите на оси метки и проставьте значения в масштабе графика.
2. Напишите процедуры формирования текстов указания координат для примера, показанного на рис. 2.
3. Напишите процедуры для нанесения меток и вывода значений по горизонтальной и вертикальной осям для примера моделирования математических функций.
4. В модели тела, брошенного под углом у горизонту, напишите процедуры вывода метки «точки падения» и метки максимальной высоты для каждой траектории.
5. Модифицируйте код для «моделирования полёта» так, чтобы можно было изменять начальную скорость, а график автоматически масштабировался в области рисования.