Введение в Python и Eric

Иван Хахаев, 2009

Библиотека Tkinter и построение графиков функций

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.

Приходится с сожалением заметить, что систематического описания функций `Tkinter` пока (по состоянию на май 2009 года) на русском языке не существует (или оно достаточно труднодоступно).

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

Пусть требуется создать окно заданного размера, установить для него заголовок и цвет фона «холста», а также снабдить окно программной «кнопкой» для закрытия (ведь `Tkinter` предназначена для создания GUI). На «холсте» определим систему координат и нарисуем «косинусоиду» (`script-g03.py`).

```
# -*- 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["font"]="-*-terminus-*-r-*-*-12-*-*-*-*-*-*-*"
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()
```

Сначала посмотрим на результат (рис. 1), а потом разберём текст примера.

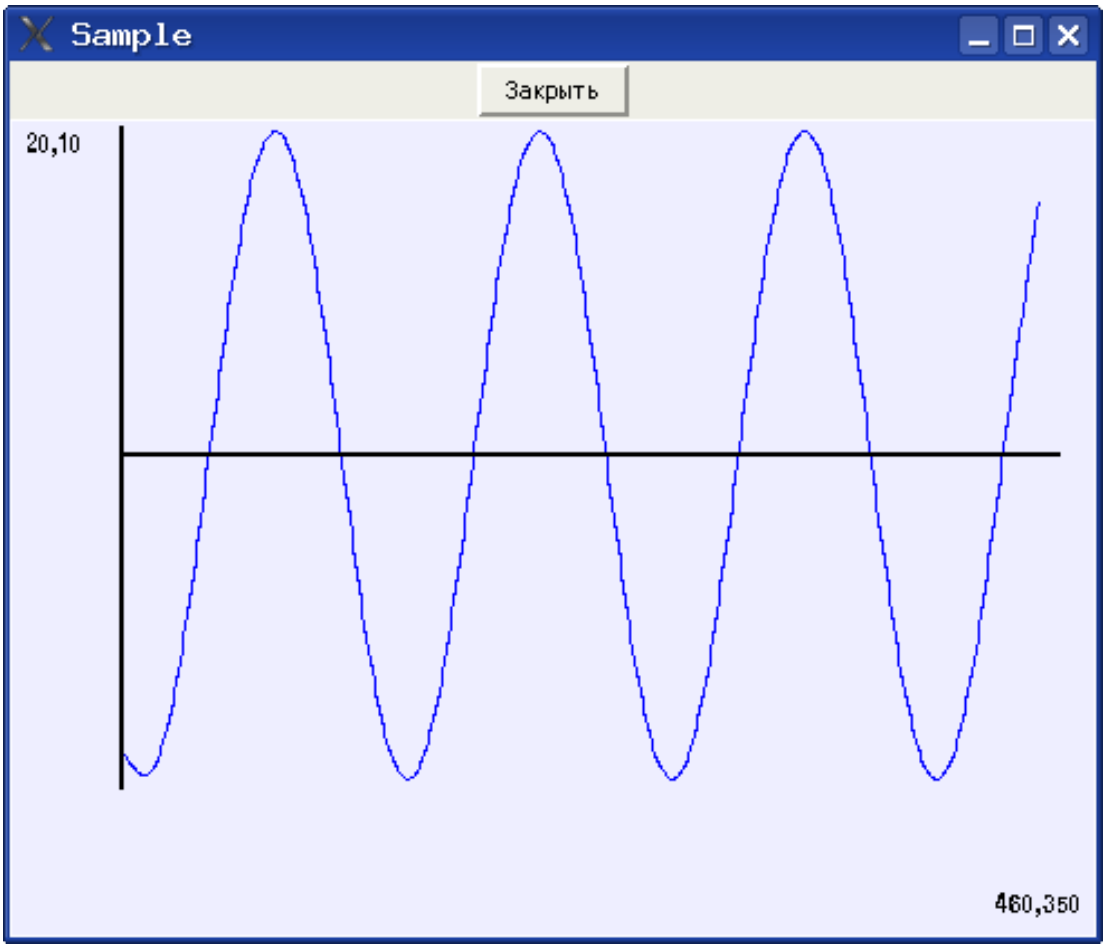

Рисунок 1. Окно `Tkinter` с кнопкой и графиком

Итак, первые три строки программы понятны — устанавливается кодовая страница и подключаются библиотеки. Поскольку в примере хочется использовать 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` (т.е. 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.

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

В качестве названия шрифта для кнопки в Linux следует брать строку, которую формирует программа `xfontsel`.

В приведённом примере кнопка связана с командой закрытия окна и прекращения работы интерпретатора. Если этот пример запустить из редактора IDE Eric, то после нажатия на кнопку «Закрыть» окно не исчезнет, поскольку будет ожидать закрытия интерпретатора Python. запущенного в IDE Eric. Для обеспечения корректной работы кнопки закрытия нужно запускать этот пример на выполнение командой

```
python script-g03.py
```

в терминале (например, Konsole), предварительно переместившись в каталог с файлами примеров.

Однако ничто не мешает закрыть окно нашего «приложения» обычным образом — с помощью стандартной интерфейсной кнопки закрытия окна.

После создания и размещения кнопки создаётся «холст». Для элемента `canvas` указываются высота, ширина, цвет фона и отступ от границ окна (т.е. 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».

Упражнение: Нанесите на оси метки и проставьте значения в масштабе графика.

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

Результат решения задачи (вариант внешнего вида «приложения») показан на рис. 2.

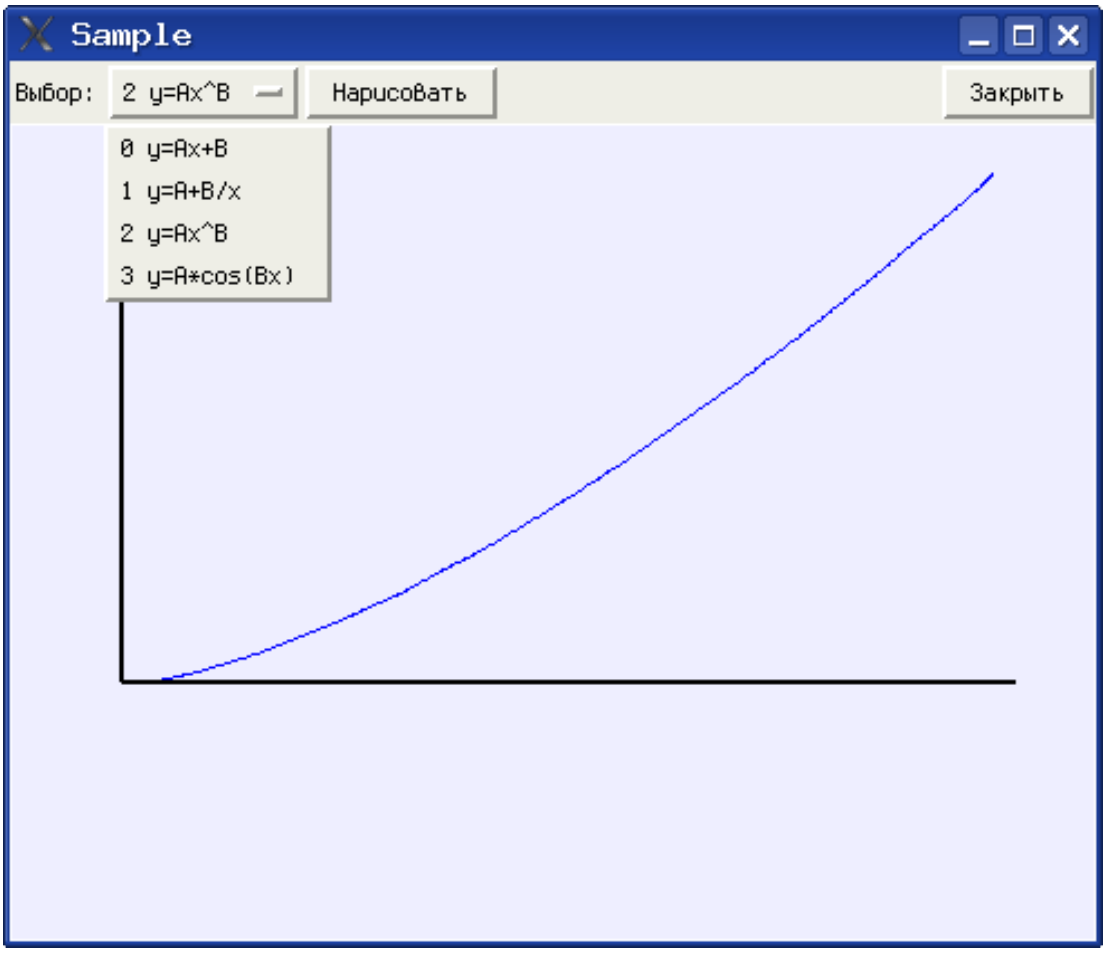

Рисунок 2. Построения графика для функции, выбираемой из списка

Для выбора функции используется раскрывающийся список, после выбора вида функции и нажатия на кнопку «Нарисовать» на «холсте» схематически рисуется график этой функции. Кнопка «Закрыть» закрывает «приложение» (с учётом замечаний, касающихся запуска приложения из IDE Eric).

Теперь посмотрим на код, который формирует такое «приложение» (`script-g03-a.py`).

```
# -*- coding: utf-8 -*-
import Tkinter
import math
#
# Определение функций
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 типа `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 функции `DrawGraph()`.

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

Определение каждой 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).

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

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

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

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

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

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

Замечание для Linux и версий Tk до 8.5:

В предыдущем примере для установки шрифта кнопки был использовано свойство «`text`». Однако не все интерфейсные элементы в `Tkinter` позволяют таким образом устанавливать шрифт. Чтобы установить одинаковый шрифт для кнопок и раскрывающегося списка, можно выполнить следующие действия.

1. Создать в «домашнем каталоге» с помощью любого текстового редактора (KWrite или Kate) файл с именем `.Xresources` (имя должно начинаться с точки!)
2. Записать в этом файле одну-единственную строку:

```
Tk*font: -*-terminus-*-r-*-*-12-*-*-*-*-*-*-*
```

3. Сохранить файл и перезагрузить сеанс работы (выйти из сеанса и войти снова)

Эта строка содержит название желаемого шрифта, записанное так, как оно формируется в программе `xfontsel` (об этом уже говорилось выше).

Упражнение: используя функцию вывода текста элемента `canvas` проверьте, влияет ли установка `Tk*font` на шрифты в области рисования.