Готовый отчет можно сохранить, распечатать, изменить масштаб. Также при большом объеме информации в отчете есть возможность перехода по записи (рисунок 9).

Наименование	Количество	Единица	Место	Номер	Штрих-код	Дата
Актинид	1000	кг	1344000	1000	1344000	10/11
Актинид	100	кг	1344000	100	1344000	10/11
Актинид	1000	кг	1344000	1000	1344000	10/11
Актинид	1000	кг	1344000	1000	1344000	10/11
Актинид	1000	кг	1344000	1000	1344000	10/11
Актинид	1000	кг	1344000	1000	1344000	10/11
Актинид	1000	кг	1344000	1000	1344000	10/11
Актинид	1000	кг	1344000	1000	1344000	10/11
Актинид	1000	кг	1344000	1000	1344000	10/11
Актинид	1000	кг	1344000	1000	1344000	10/11

Рисунок 9 – Отчет

Разработанная программа автоматизирует процесс инвентаризации оборудования и позволяет выполнять следующие действия:

- просмотр содержания таблиц;
- ввод, редактирование и удаление данных в таблицах;
- выполнение и просмотр запросов и отчетов.

Созданный в Lazarus пользовательский интерфейс - удобен, прост, наглядно отображает функции и возможности которыми обладает программа.

Тестирование подтвердило, что программа корректно выполняет обработку данных и демонстрацию результатов. Всё это свидетельствует о работоспособности разработанной программы и позволяет сделать вывод о ее пригодности для эксплуатации.

1. Мансуров, К.Т. Основы программирования в среде Lazarus, 2015. – 772 с.
2. Новиков, П.В. Объектно-ориентированное программирование [Электронный ресурс]: учебно-методическое пособие к лабораторным работам/П.В. Новиков – Электрон.текстовые данные. – Саратов: Вузовское образование, 2017. – 124 с. – Режим доступа: <http://www.iprbookshop.ru/64650.html> (Дата обращения: 27. 11.2019).
3. Lazarus. - Режим доступа: <http://intuit.valrkl.ru/course-1265/index.html> (Дата обращения: 28. 11.2019).
4. Lazarus. Режим доступа: http://msk.edu.ua/ivk/Informatika/Uch_posobiya/Programmirovaniye/Lazarus/1_vvedenie_v_Lazarus.htm (Дата обращения: 28.11.2019).
5. Lazarus. - Режим доступа: <https://lazarus-rus.ru/> (Дата обращения: 28. 11.2019).

Шавтикова Л.М.

Использование языка Python для программирования графики

*Северо-Кавказская государственная академия
(Россия, Черкесск)*

doi: 10.18411/lj-12-2019-30

idsp: ljjournal-12-2019-30

Аннотация

В данной статье рассматривается пример создания графика двух функций в графическом режиме. Изучены модули языка программирования Python, позволяющих создавать графический интерфейс пользователя (GUI).

Ключевые слова: графический интерфейс пользователя, язык программирования Python, модули и библиотеки.

Abstract

This article discusses an example of creating a graph of two functions in graphical mode. We studied modules of the Python programming language that allow you to create a graphical user interface (GUI).

Keywords: graphical user interface, Python programming language, modules and libraries.

В Python реализовано много модулей и библиотек, позволяющих создавать графический интерфейс пользователя (GUI). Стандартным модулем, который устанавливается вместе с Python по умолчанию, является модуль Tkinter. У этого модуля понятный и ясный синтаксис и его можно использовать в простых программах с графическим интерфейсом. В данной работе используется Модуль turtle.

Целью работы является вывести на экран в графическом режиме графики двух функций на интервале от $X_{нач}$ до $X_{кон}$ с шагом dx . Для использования методов и функций модуля необходимо выполнить импорт модуля. Это реализуется строкой: `from tkinter import *`. В Tkinter визуальные компоненты, используемые для построения пользовательского графического интерфейса, называют виджетами. Виджет — примитив графического интерфейса пользователя (GUI), имеющий стандартный внешний вид и выполняющий стандартные действия. Виджеты, описываемые в скрипте обязательно должны быть обработаны упаковщиком, который размещает их на форме. В нашем скрипте используется упаковщик Метод `grid`. Выделим в нашем скрипте две части. В головной части - описание функций, которые будут выполнять некоторый набор операций, а во второй – описание виджетов, которые будут организовывать интерфейс, для взаимодействия с пользователем. Для этого потребуются следующие виджеты: `Canvas` – полотно для графического представления функций; `Entry` – поле для ввода данных; `Button` – кнопка для задания действия: вывод обновлённого графика, после изменения значений, задаваемых через поле `Entry`; `Label` – метка, для описания полей `Entry`.

Напишем скрипт, который покажет, как используется виджет `Canvas`:

```
from tkinter import * # Импорт модуля
root = Tk() # Создание главного окна
root.title("Графика") # Надпись в заголовке окна
root.resizable(False, False) # Не менять размер окна
# по ширине и высоте
Kp = 0.7 # используем 70% экрана
MaxX = root.winfo_screenwidth()* Kp # Текущие размеры
MaxY = root.winfo_screenheight()* Kp # дисплея монитора
# в пикселях
cv = Canvas(root, width = MaxX, height = MaxY, bg = "white")
cv.grid(row = 0, columnspan = 11)
```

Через методы главного окна можно получить размеры дисплея монитора: `winfo_screenwidth()` # Размер по ширине в пикселях

`winfo_screenheight()` # Размер по высоте в пикселях

Привязка виджета к форме выполняется при его описании. При этом формируется идентификатор виджета, значение которого может быть присвоено переменной. Упаковка виджетов, будет выполняться методом `grid()`. В качестве аргументов передаётся номер строки, на которой будет размещаться полотно, и сколько столбцов оно будет занимать. Создадим скрипт с указанным текстом. Добавим в форму метки (`Label`) у нас их восемь. Для описания меток используется метод `Label()`, в аргументах которого надо указывать имя окна, к которому относится метка. Опишем все восемь меток:

```
lba0 = Label(root, text = "X:", width = 10, fg = "blue", font = "Ubutu, 12")
```

```

lba0.grid(row = 1, column = 0, sticky='e')
lba1 = Label(root, text = "Y:", width = 10, fg = "blue", font = "Ubutu, 12")
lba1.grid(row = 2, column = 0, sticky='e')
lba2 = Label(root, text = "Xmin:", width = 10, fg = "blue", font = "Ubutu, 12")
lba2.grid(row = 1, column = 2, sticky='e')
lba3 = Label(root, text = "Xmax:", width = 10, fg = "blue", font = "Ubutu, 12")
lba3.grid(row = 1, column = 4, sticky='e')
lba4 = Label(root, text = "Ymin:", width = 10, fg = "blue", font = "Ubutu, 12")
lba4.grid(row = 2, column = 2, sticky='e')
lba5 = Label(root, text = "Ymax:", width = 10, fg = "blue", font = "Ubutu, 12")
lba5.grid(row = 2, column = 4, sticky='e')
lba6 = Label(root, text = "Шаг меток:", width = 10, fg = "blue", font = "Ubutu, 12")
lba6.grid(row = 1, column = 6, sticky='e')
lba7 = Label(root, text = "Смещение:", width = 10, fg = "blue", font = "Ubutu, 12")
lba7.grid(row = 2, column = 6, sticky='e')

```

Для завершения формы необходимо добавить поля для ввода и вывода данных – виджеты Entry. Значение строкового типа в поле Entry ставим методом insert(): ent0.insert(index, s), где строка S вставляется перед символом, позиция которого задается параметром index. Удалить символы из виджета можно методом delete(): ent0.delete(first, last=None), где параметр first задаёт начальную позицию удаляемых символов, а параметр last последнюю. Прочитать значение, введённое пользователем в поле, можно методом get(): var = float(ent0.get()). В данном скрипте используются вещественные значения: float(). Два первых поля (ent0 и ent1) предназначены для вывода координат мыши. Инициализируются они нулевым значением, а оставшиеся поля, предназначенные для ввода, которые инициализируются начальными значениями:

```

Xmin = -10.0; Xmax = 10.0 # Границы по абсциссе
Ymin = -5.0; Ymax = 5.0 # Границы по ординате
dX = 50 # Шаг меток по осям
dY = 1.0 # Смещение графика

```

Далее добавляем кнопки (Button). При описании этих виджетов можно задать надпись, которая будет размещена на кнопке, параметры текста (шрифт, цвет) и что очень важно, привязать нажатие кнопки к вызову функции, которая получит событие от нажатия кнопки и обработает его: btn1 = Button(root, width = 20, bg = "#ccc",

```

text = "Рисовать")
btn1.grid(row = 1, column = 10)
btn1.bind("<Button-1>", Draw)

```

В первой строке указан идентификатор кнопки (btn1), параметр root показывает, что кнопка находится на форме. Метод grid() поместит кнопку в первой строке в десятой колонке. Для обработки события, связанного с кнопкой, будет вызываться функция Draw(). Код для второй кнопки будет аналогичен. Заключительный шаг для формирования окна скрипта – это организация непрерывного цикла, в котором будут обрабатываться события. Для организации цикла, в котором будет находиться скрипт в ожидании событий, используется метод mainloop(): root.mainloop().

В процессе написания скрипта созданы три условия для возникновения событий: кнопка "Рисовать", кнопка "Выход" и событие, связанное с кликом левой кнопки мыши. Реализуем обработку этих событий в виде функций: def Draw(event):

```

"""Вызов функций для рисования графиков"""
pass #
def Final(event):
    """Завершение работы"""
    window_deleted() #
def window_deleted():

```

```

""" Завершаем работу по [X]"""
if askyesno("Выход", "Завершить работу?"):
    root.destroy() #
def showXY(event):
    """Вывод координат мыши"""
    global ID1, ID2
    x = event.x; y = event.y
    ent0.delete(0,END) # Удалим старые
    ent1.delete(0,END) # значения
# Вычислим и введём новые
    ent0.insert(0, str(round(Xmin + x/Kx, 2)))
    ent1.insert(0, str(round(Ymin + (MaxY - y)/Ky, 2)))
    cv.delete(ID1) # Удалим старые
    cv.delete(ID2) # линии
# Нарисуем новые линии
    ID1 = cv.create_line(0,y,MaxX,y, dash = (3,5))
    ID2 = cv.create_line(x,0,x,MaxY, dash = (3,5))

```

В качестве параметра функций указана переменная *event*, через которую в функцию передаётся описание события. Функция *Final()* срабатывает по нажатию кнопки "Выход" и завершает работу скрипта. Путь закрытия окна – это кнопка [X] в правом верхнем углу формы. Для обработки этого события используется метод *protocol()*, который перехватывает событие, связанное с закрытием окна под кнопкой [X]. Функция *plotXY()* рисует линейки по краям полотна. Для прорисовки координатной линейки вначале рисуется прямоугольник, а затем на его сторонах наносятся метки в виде коротких линий и текстовые значения, соответствующие этим меткам. При нанесении значения выполняется его вычисление, округление до двух знаков после запятой, а затем преобразование в строковый тип. Рисование функций начинается с появления события, связанного с кликом по кнопке "Рисовать". При этом вызывается функция *Draw()*, в которой последовательно выполняются следующие операции: очистка полотна от ранее нарисованных объектов; получение данных из полей *Entry*. Результат работы скрипта представлен на рисунке 1.

Рисунок 1. Результат работы скрипта

1. Авриевич С.Р. , Язык программирования Python. Курс лекций.
2. Прохоренко Н.А., Дронов В.А., Python 3 и PyQt 5. Разработка приложений: СПб: БХВ-Петербург, 2017
3. Шапошникова С., Основы программирования на Python. Вводный курс, вер.2., <http://younglinux.info>, 2011