УДК 004

Создание интерфейса для программы python3 с помощью библиотеки tkinter

Радионов Сергей Владимирович
Приамурский государственный университет им. Шолом-Алейхема
Студент

Аннотация
В данной статье показано построение интерфейса для программы, написанной на языке программирования python. Интерфейс реализован с помощью библиотеки tkinter. Для демонстрации построения интерфейса выбран интерфейс калькулятора.
Ключевые слова: python, tkinter, gui, интерфейс, программа.

Creating an interface for the python3 program using the tkinter library

Radionov Sergey Vladimirovich
Sholom Aleichem Priamursky State University
Student

Abstract
This article shows how to build an interface for programs written in the Python programming language. The interface is implemented using the tkinter library. To demonstrate the construction of the interface, the calculator interface is selected.
Keywords: python, tkinter, gui, interface, program.

При проектировании ПО немаловажную роль имеет пользовательский интерфейс приложения. Его важность заключается в том, что по интерфейсу пользователи, работающие с программой, оценивают ее в целом. Графический интерфейс предоставляет пользователям возможности удобной работы с приложениями, не требуя от них специальных навыков программирования. Казалось бы, что привлекательный интерфейс программы не должен производить сегодня на пользователей весомое впечатление, чтобы они забывали при этом о функционале продукта. Но первая реакция пользователей на продукт, к сожалению, напрямую зависит именно от интерфейса. К слову, интерфейс — механизм, обеспечивающий взаимодействие пользователя с приложением. Именно поэтому рациональная конструкция интерфейса приложения, обеспечивая простоту работы с ним, имеет первостепенное значение в создании приложений. В связи с этим, интерфейс должен прежде всего учитывать потребности конечного пользователя приложений. Если целевая аудитория разрабатываемой программы уже известна, то разработка пользовательских интерфейсов не представляет трудностей. Грамотно сконструированный интерфейс упрощает

освоение программы. И, следовательно, неудачный — наоборот снижает эффективность работы с приложениями, вызывает у клиентов разочарование. В итоге пользователи начинают игнорировать его и вообще полностью отказаться от него.

В статье И. Кутепова рассматривается пример построения вычислителя на основе нечеткой логики и применение языка Python для написания интегрированной среды разработки [2]. И.Е. Бронштейн в своей статье описал вывод типов для программного кода на языке Python. Сначала производится обзор описанных в научной литературе алгоритмов вывода типов для языков с параметрическим полиморфизмом. Затем даётся описание нового алгоритма, являющегося модификацией одного из предыдущих: алгоритма декартова произведения. Показывается, как модуль вывода типов, использующий новый алгоритм, анализирует различные конструкции языка Python. Представляются результаты работы над прототипом [3]. В работе Найденова В.В. протестирована производительность пары аналогичных приложений, реализующих CRUD логику с помощью прослойки ORM. Сравнивается SQLAlchemy – де-факто стандартный ORM для Python с динамическим ORM для C++ собственной разработки – YB.ORM. Сравнивается производительность при использовании CPython и PyPy. Проверяется влияние отключения логов на производительность [4]. В статье Бронштейн И.Е. рассматриваются виды дефектов, которые обычно встречаются в программном коде на языке Python. Показывается, что возможные дефекты для Python не похожи на те, что часто встречаются в коде на Си/Си++ и, следовательно, необходимо исследование дефектов в крупных проектах с открытым исходным кодом. Даётся классификация найденных дефектов на основе того, нужен ли для нахождения ошибки вывод типов. Показывается, что существует небольшая доля "простых" дефектов, но для обнаружения большинства дефектов вывод типов необходим. Рассматривается вопрос, какие конструкции языка Python должны поддерживаться при выводе типов для нахождения реальных дефектов [5]. В работе Д.А. Кузнецов рассматривается структура интерфейса программы «Фармацевтическая экономическая безопасность» для фармацевтических организаций. Описывается функциональное предназначение и возможности пунктов основного меню прикладной программы [6]. Ю.А. Котов, А.В. Шаповалов в своей статье рассмотрели интерфейс программной реализации экспертной системы для восстановления простой замены букв текста. Описаны базовые элементы интерфейса, включающие выбор функциональных методов и базовых операций (замена, сдвиг, перебор). Не менее значимы иностранные исследования в данной сфере [8-9].

Разработку нашего калькулятора начнем с создания скрипта python, дадим ему стандартное название «main.py». Сначала необходимо импортировать нужные классы (Рис.1). Из встроенной в python библиотеки tkinter импортируем все классы, а также метод messagebox, с помощью которого можно сделать модальные окна. Помимо этого импортируем

библиотеку ttk, которая является расширенной библиотекой tkinter. Далее напишем код отвечающий за запуск приложения (Рис.2). Разберем этот код поподробнее. Условие отвечает за то, чтобы код в нем выполнился только в случае запуска кода из скрипта с названием main, т.е. если мы импортируем код из нашего скрипта в другой, и запустим, код в условии не будет выполнен. В нашем случае это неважно, но писать это условие - обычная практика. Создаем переменную root, которая является окном tkinter. Создаем переменную нашего класса Calculator, который будет рассмотрен далее, и запускаем наше окно методом mainloop.

```
from tkinter import *
from tkinter import messagebox
import tkinter.ttk as ttk
```

Рисунок 1. Импортируемые библиотеки

```
if __name__ == '__main__':
    root = Tk()
    app = Calculator(root)
    root.mainloop()
```

Рисунок 2. Код запуска приложения

Далее необходимо описать наш класс Calculator (рис.3). Булевская переменная calculated будет использоваться для определения вычислено введенное выражение или нет. Конструктор класса принимает на вход окно tkinter в переменную master. В конструкторе объявляем внутриклассовую переменную окна, устанавливаем разрешение нашего окна в 300 пикселей по ширине и 400 по высоте. Меняем название нашего окна на “Calculator”, исключаем возможность изменения размера окна. Далее создадим метод класса init_gui (рис. 4) и добавим его в конструктор.

```
class Calculator:
    calculated = True
    def __init__(self, master):
        self.master = master
        self.master.geometry('300x400')
        self.master.title('Calculator')
        self.master.resizable(False, False)

        self.init_gui()
```

Рисунок 3. Класс Calculator

```
def init_gui(self):
    self.expression = Label(self.master, text='0', font=('Calibri', 16))
    self.expression.place(relx=0.9, y=20, anchor='ne')

    self.frame_buttons = Frame(self.master)
    self.frame_buttons.place(x=10, y=80, relwidth=1)

    btn_symbols = ['(', ')', 'C', '<-',
                   '1', '2', '3', '+',
                   '4', '5', '6', '-',
                   '7', '8', '9', '*',
                   '0', '.', '=', '/']

    s = ttk.Style()
    s.configure('my.TButton', font=('Calibri', 16))

    for i, symbol in enumerate(btn_symbols):
        btn = ttk.Button(self.frame_buttons, text=symbol, style='my.TButton', width=5)
        btn.grid(row=i//4, column=i%4, ipady=10)
        btn.bind('<ButtonRelease-1>', lambda ev, sym=symbol: self.btn_click(sym))
```

Рисунок 4. Метод init_gui класса Calculator

В этом методе мы создаем интерфейс нашего калькулятора. Первым делом на форму помещается Label, который предназначен для отображения нашего выражения. Родительским объектом установлено наше окно. По умолчанию ему задано значение ‘0’, также установлен шрифт Calibri, размер шрифта 16. Метод place переменной любого интерфейсного класса библиотеки tkinter размещает элемент относительно родительского объекта по указанным координатам. Координаты можно указать как в численном виде так и в процентном виде, относительно размера родительского элемента. Здадаим параметры place: «relx» равное 0.9, что говорит о том что позиция по x равна 90% от ширины окна, «y» равное 20 пикселей, якорь (anchor), устанавливающий по какой точке позиционировать элемент, приравняем к «ne», что означает верхний правый угол нашего выражения. Таким образом выражение будет выравниваться по правому краю.

Разместим разметочный элемент Frame под нашим выражением, ширину зададим равную нашему окну с помощью параметра «relwidth».

Далее создадим список btn_symbols, в котором укажем все символы отображаемые на каждой кнопке по порядку. У нас будет 16 кнопок, отображенных таблицей 4x4.

Создадим и сконфигурируем стиль «my.TButton». Он нужен для класса кнопки из расширенной библиотеки ttk для отображения более красивых и современных кнопок. Укажем шрифт Calibri и размер шрифта.

Далее создадим все 16 кнопок. Осуществим перебор списка btn_symbols с индексом в цикле for. Внутри цикла создаем кнопку, текст которой равен элементу из списка. Стиль (style) приравниваем к созданному ранее. Ширину указываем равную 5 единицам. Для размещения кнопок будем использовать метод grid, который позиционирует элементы в табличном виде. Строку (row) для размещения указываем равной i//4, что

означает индекс деленный на 4 нацело. Столбец (column) приравниваем к i%4, что означает остаток от деление индекса на 4. Боковые отступы от текста внутри кнопки (ipady) указываем равными 10. Назначаем кнопке событие при нажатии и отпускании на ней левой кнопкой мыши (ButtonRelease-1). В исполняемую функцию передаем метод класса Calculator btn_click (Рис. 5), указываем, что при вызове этого метода необходимо передать символ нажатой кнопки.

```
def btn_click(self, symbol):
    if symbol == '=':
        try:
            self.expression['text'] += '=' + str(eval(self.expression['text']))
            self.calculated = True
        except:
            messagebox.showerror('Ошибка', 'Неверное выражение')
    elif symbol == 'C':
        self.expression['text'] = '0'
        self.calculated = True
    elif symbol == '<-':
        expr = self.expression['text']
        if len(expr)==1:
            self.expression['text'] = '0'
        else:
            self.expression['text'] = expr[:expr.find('=')]
        self.calculated = False
    else:
        if self.calculated:
            self.expression['text'] = symbol
        else:
            self.expression['text'] += symbol
        self.calculated = False
```

Рисунок 5. Метод btn_click класса Calculator

В методе btn_click мы смотрим с каким символом была нажата кнопка и в зависимости от этого символа принимаем необходимые действия. Если символ равен «=» тогда добавляем в текст выражения этот символ и вычисляем выражение с помощью встроенного в python метода eval, указываем что выражение посчитано. Если посчитать не удалось, вызываем модальное окно ошибки с текстом «Неверное выражение». Если символ равен «C» обнуляем выражение и указываем что выражение не посчитано. При символе равном «<» стираем последний символ при отсутствии в нем символа «=», иначе стираем его и все что стоит после него. Если символ 1 обнуляем выражение. Для всех других символов делаем следующее: если выражение не посчитано добавляем символ, если посчитано, стираем выражение и добавляем символ.

Конечный вид калькулятора представлен на рисунке 6.

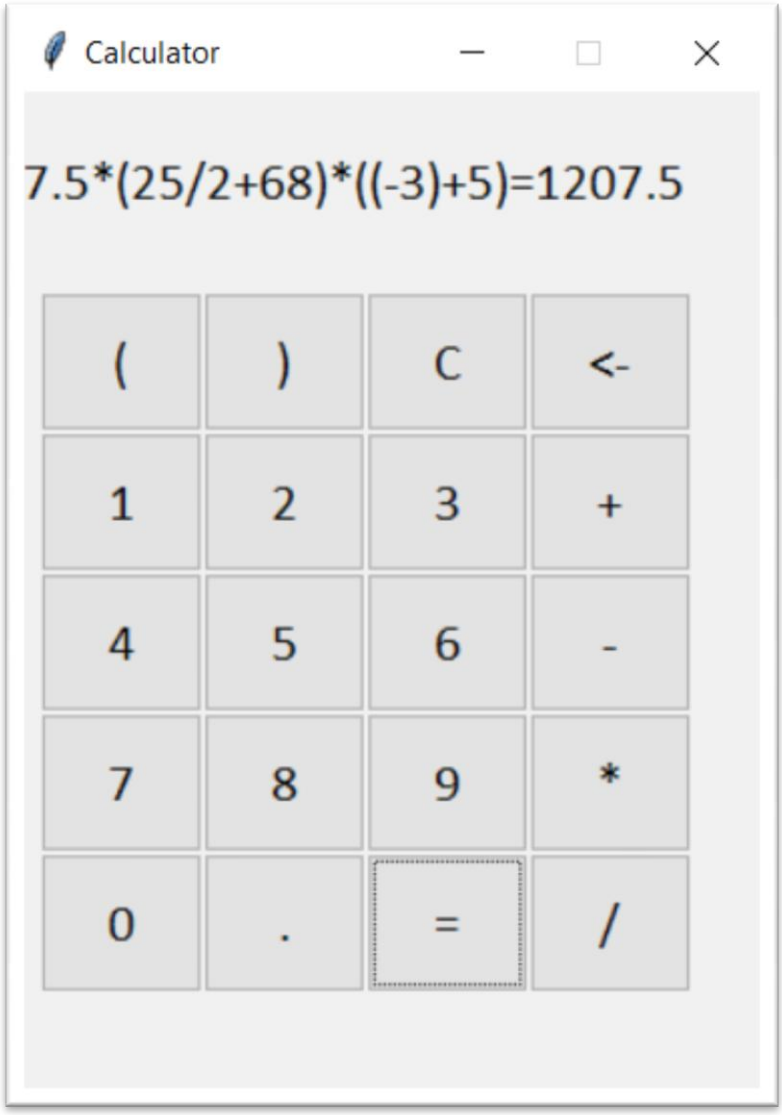

Рисунок 6. Внешний вид калькулятора

Таким образом, было рассмотрено создание калькулятора с помощью библиотеки tkinter языка программирования python3.

Библиографический список

1. Лутц М. Изучаем python. М., 2009.
2. Кутепов И. Применение языка python при проектировании нечеткого контроллера // Компоненты и технологии. 2013. № 8 (145). С. 148-154.
3. Бронштейн И.Е. Вывод типов для языка python // Труды Института системного программирования РАН. 2013. Т. 24. С. 161-190.
4. Найденов В.В. Тестирование производительности orm в языках python и c++ // RSDN Magazine. 2014. № 1. С. 05-08.
5. Бронштейн И.Е. Исследование дефектов в коде программ на языке python // Программирование. 2013. Т. 39. № 6. С. 25-32.
6. Кузнецов Д.А. Интерфейс программы "фармацевтическая экономическая безопасность" // Российский медико-биологический вестник им.

академика И.П. Павлова. 2009. № 2. С. 162-165.
7. Котов Ю.А., Шаповалов А.В. Интерфейс программы восстановления простой замены букв текста // Современные тенденции развития науки и технологий. 2016. № 4-4. С. 57-59.
8. Smith A. W. et al. Application program interface that enables communication for a network software platform : пат. 7117504 США. 2006.
9. Parikh V., Moore R., Cheng H. Application program interface for a graphics system : пат. 6456290 США. 2002.