

МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт математики, механики и компьютерных наук им. И. И. Воровича
Кафедра вычислительной математики и математической физики

Курс «Основы алгоритмизации и программирования»

Андрей Петрович Мелехов

**Лекция 1. Создание переносимых графических интерфейсов
пользователя (Graphical User Interface, GUI). Tkinter**

Ростов-на-Дону

2020

История: Application Programming Interface (API Windows), Microsoft Foundation Classes (MFC Windows, C++), Visual Basic, Delphi, C#, Java и т. д.

В Python для настольных приложений есть несколько библиотек создания переносимых графических интерфейсов пользователя (Graphical User Interface, GUI). Например, tkinter, wxPython, PyQt и др. Библиотека tkinter входит в стандартную установку Python. Для работы с ней не надо ничего доставлять, нужно просто подключить модуль tkinter к программе.

Программы, написанные на Python с GUI tkinter, переносимы, т. е. должны работать в разных системах (Windows, Unix, Mac OS X). Причем, создается графический интерфейс со свойственным каждой системе видом.

Библиотека Tk, на которой основана библиотека tkinter используется также языками сценариев Perl, Ruby, PHP, Common Lisp и Tcl.

Пример 1. Минимальная программа с визуальным интерфейсом tkinter

Рассмотрим простейший пример с tkinter. Откройте Pyzo, создайте в нем файл “pr1.pyw” и наберите в нем код примера 1. Расширение лучше выбирать pyw, вместо py. Тогда при запуске программы не будет появляться окно консоли.

```
from tkinter import *  # подключение библиотеки tkinter  
  
# Создаем метку и прикрепляем ее к верху формы (окно программы):  
Label(text = "Hello World").pack(side = TOP)  
  
# Если явно в программе не создавать главное окно,  
# то библиотека tkinter создаст его самостоятельно.
```

Создаем функцию без параметров (обработчик нажатия кнопки):

def hello():

print("На кнопку нажали")

Создаем кнопку, подключаем обработчик и прикрепляем

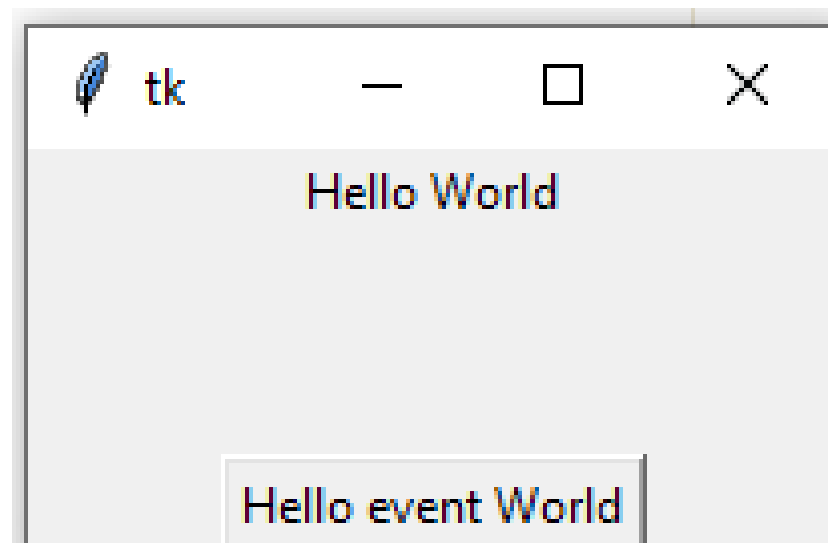
ее к низу формы:

Button(text = "Hello event World",

command = hello).pack(side = BOTTOM)

Запускаем цикл обработки сообщений:

mainloop()

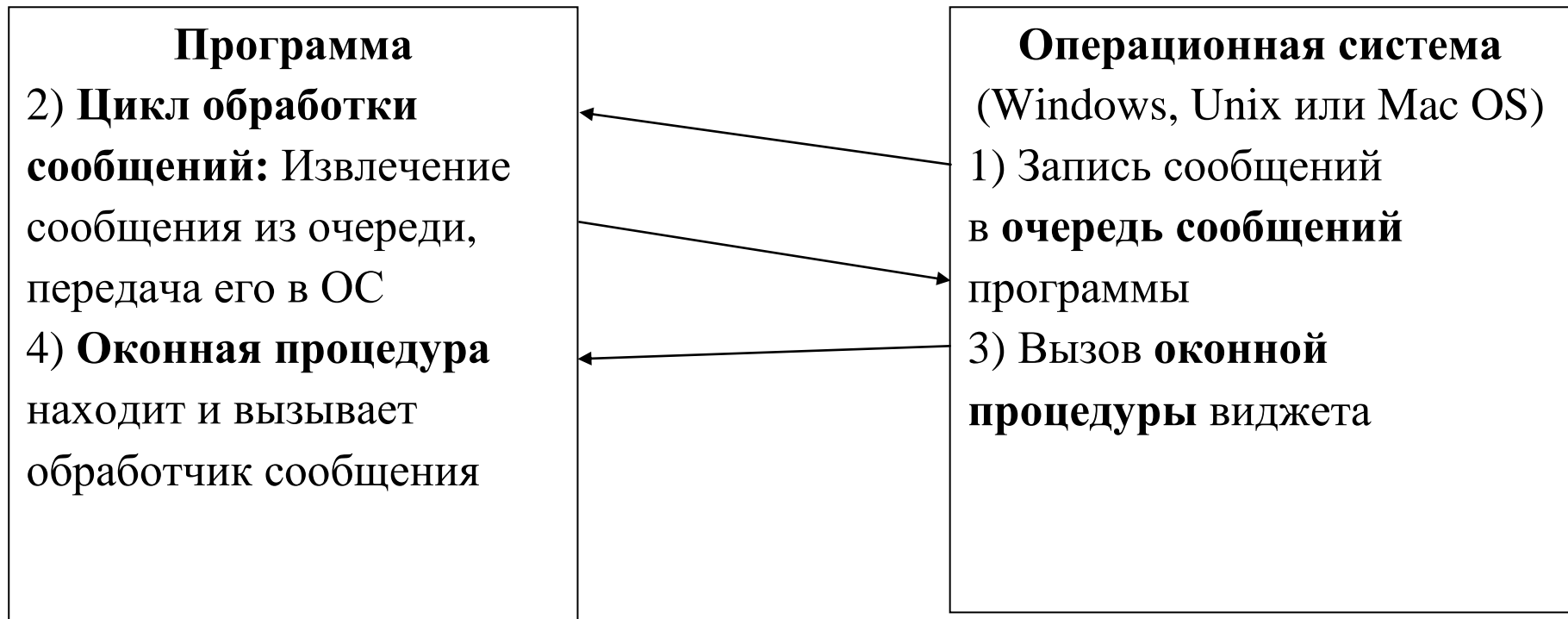


У двух компонент в этом примере выбран минимальный набор свойств: надписи `text` и обработчик `command` для кнопки. Метод `pack` размещает компоненты на форме.

При создании компонент можно задавать больше параметров (цвет, шрифт, отступы, граница и т.п.). Эти параметры можно менять в дальнейшем. Через параметр `command` можно подключить только один обработчик без параметров. Более сложные обработчики с параметрами подключаются иначе.

Схема работы программы:

- 1) Создание графического интерфейса, регистрация обработчиков событий.
- 2) Запуск цикла обработки сообщений.



Пример 2. Создание главного окна, задание свойств виджетов

В следующем примере явно создадим главную форму, а также у **компонент** (их еще называют **виджетами**) установим некоторые свойства.

```
from tkinter import * # подключение библиотеки tkinter
```

```
# Метод Tk создает главное окно, ссылку сохраним
```

```
# в переменную root:
```

```
root = Tk( )
```

```
# по умолчанию размер окна будет выбран по его содержимому,
```

```
# положение окна выбирает операционная система
```

```
# Можно задать размеры и положение окна
```

```
# самостоятельно (W x H + X + Y):
```

```
root.geometry("400 x 200 + 150 + 100")
```

```
# Можно задать минимальные и максимальные размеры
```

```
# root.minsize(width = 150, height = 150)
```

```
# root.maxsize(width = 500, height = 500)
```

```
# Заголовок окна
```

```
root.title("Hello world!!!")
```

```
# Свойства виджета можно задавать
```

```
# (1) при создании,
```

```
# (2) обращаясь как к словарю (виджет['свойство'] = ... ) или
```

```
# (3) с помощью метода config:
```

```
# задаем при создании (1) text и цвет фона bg = background:
```

```
w = Label(root, text = "Hello config world", bg = 'black')
```

```
# первый параметр root – виджет, на котором
```


будет размещена метка.

w['fg'] = 'yellow' # (2) задаем цвет текста fg = foreground

w.config(font = ('times', 20, 'bold')) # (3) задаем шрифт

задаем высоту height (в строках), ширину width (в символах),

ширину db = borderwidth и стиль бордюра relief:

w.config(height = 3, width = 20, bd = 8, relief = RAISED)

задаем вид курсора мыши над виджетом:

w.config(cursor = "cross")

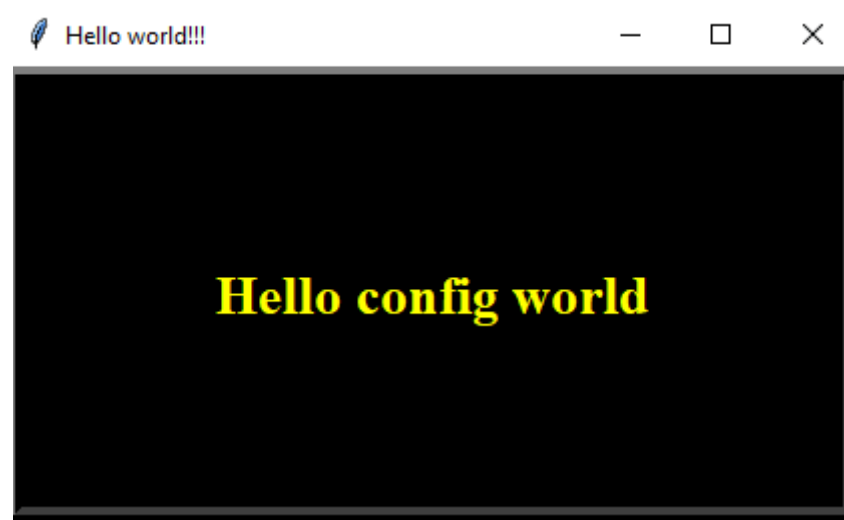
размещаем компонент с помощью метода pack:

w.pack(side = TOP, expand = YES, fill = BOTH)

– прижать к верхнему краю (можно не указывать, side = TOP по умолчанию)

– растягивать виджет при изменении размеров окна expand = YES

по двум направлениям fill = BOTH (можно еще = X или = Y)
Вывод главного окна и запуск цикла обработки сообщений:
root.mainloop()



Пример 3. Задание событий

В следующем примере создадим 3 кнопки и подключим к ним более сложные обработчики с параметрами. У первых двух кнопок будем использовать **свойство command** (как в примере 1) для подключения обработчиков, а к третьей кнопке подключим обработчик с помощью более универсального **метода bind**. Метод `bind` способен связывать с виджетом разнообразные обработчики событий с параметрами, не только нажатие на кнопку.

```
from tkinter import *
```

```
root = Tk( ) # — создаем главное окно
```

```
root.title("Привет мир!!!") # — заголовок окна
```

```
# Создаем метку и сохраняем ссылку lab1 на нее,  
# чтобы в дальнейшем обращаться к ней по этому имени:  
lab1 = Label(root, text = "Текст на метке", bg = '#F5F5F5',  
              fg = '#0000FF', font = ('times', 20, 'bold'))  
# Размещаем метку на форме, прижимаем ее к верхнему краю.  
# Метка будет растягиваться вместе с формой  
lab1.pack(side = TOP, expand = YES, fill = BOTH)  
  
# Создаем функцию с параметром x – обработчик нажатия  
кнопки 1 и 2.  
# Связывание через свойство command:  
def oncommand(x):  
    print("Нажали на кнопку", x)
```

Создаем две кнопки, задаем свойство command:

```
but1 = Button(root, text = "кнопка 1",  
               command = (lambda: oncommand(1)))  
but1.pack( )
```

```
but2 = Button(root, text = "кнопка 2",  
               command = (lambda: oncommand(2)))  
but2.pack( )
```

**# Т.к. command можно присваивать только функции без
параметров, создаем lambda-функцию без параметров
(обертку), внутри которой вызывается функция с параметрами.
Параметр передаем 1 и 2 для первой и второй кнопок,
соответственно.**

Создаем функцию с параметром event – обработчик нажатия кнопки 3.

Связывание через bind, в event будут передаваться ОС параметры события:

def onclick(event):

печатаем имя виджета и координаты мыши при щелчке:

print('Нажали на кнопку %s X = %s Y = %s'
% (event.widget, event.x, event.y))

при каждом нажатии будем увеличивать

padx и pady – внешние границы

x = int(str(but3['padx']))

x += 1

but3.config(padx = x, pady = x)

Создаем третью кнопку:

but3 = Button(root)

but3.config(text = "кнопка 3") # – устанавливаем текст на кнопке

Назначаем реакцию на событие <Button-1> (нажатие левой

кнопки мыши) – функцию onclick с одним параметром

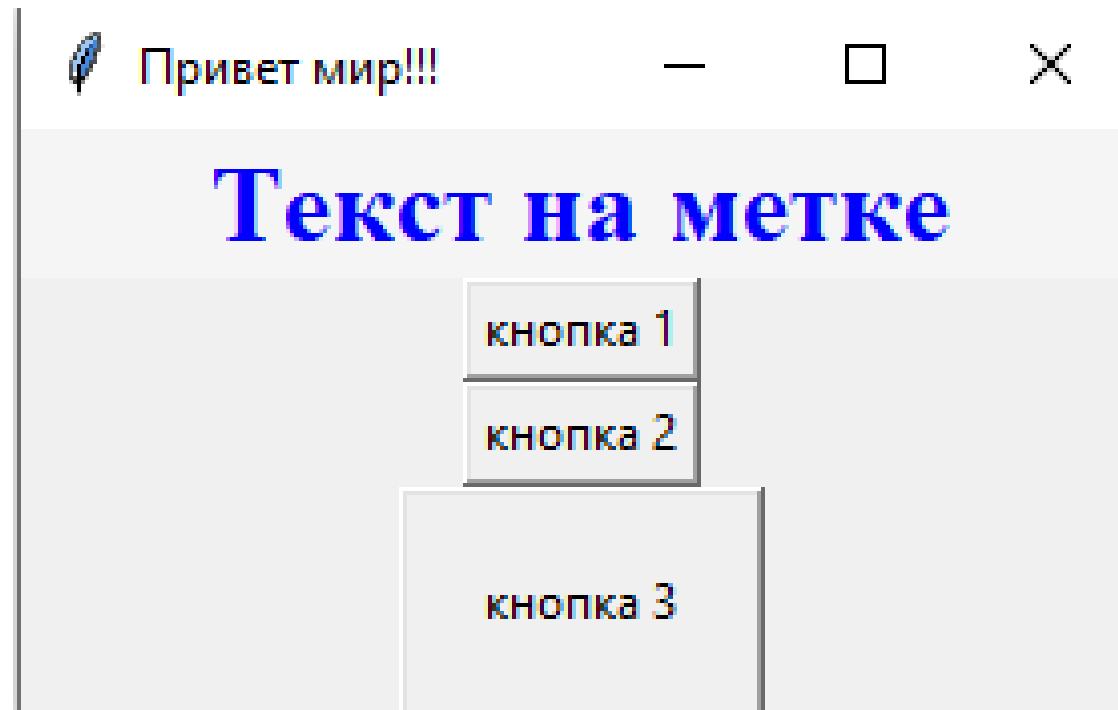
(используется операционной системой для передачи данных

в программу):

but3.bind('<Button-1>', onclick)

but3.pack()

root.mainloop()




```
>>> (executing file "pr_tkinter3.pyw")
```

```
Нажали на кнопку 1
```

```
Нажали на кнопку 2
```

```
Нажали на кнопку .!button3 X = 33 Y = 9
```

```
Нажали на кнопку .!button3 X = 34 Y = 8
```

```
Нажали на кнопку .!button3 X = 35 Y = 8
```

```
Нажали на кнопку .!button3 X = 36 Y = 9
```

```
Нажали на кнопку .!button3 X = 37 Y = 9
```

```
Нажали на кнопку .!button3 X = 38 Y = 9
```

```
Нажали на кнопку .!button3 X = 37 Y = 14
```

```
Нажали на кнопку .!button3 X = 36 Y = 18
```

```
Нажали на кнопку .!button3 X = 37 Y = 18
```

```
Нажали на кнопку .!button3 X = 38 Y = 18
```

```
Нажали на кнопку .!button3 X = 39 Y = 17
```

```
Нажали на кнопку .!button3 X = 40 Y = 17
```

```
Нажали на кнопку .!button3 X = 41 Y = 17
```

```
Нажали на кнопку .!button3 X = 42 Y = 15
```

```
Нажали на кнопку .!button3 X = 43 Y = 14
```

```
Нажали на кнопку .!button3 X = 44 Y = 14
```

Пример 4. Виджет Frame и метод размещения компонент pack

Рассмотрим еще один виджет Frame. Он служит для разделения области окна программы на отдельные прямоугольные участки и размещения на них других виджетов (аналог панели в С# и Delphi). Разберем также подробнее метод размещения компонент pack.

```
from tkinter import *
```

```
root = Tk( ) # создаем главное окно
```

```
# Создаем один frame:
```

```
win1 = Frame(root, bg = '#555555')
```

```
# Фрейм будет растягиваться при изменении
```

```
# размеров главного окна:
```

```
win1.pack(expand = YES, fill = BOTH)
```

Расположим на фрейме последовательно 4 метки.

Метки прижаты к разным краям виджета.

```
Label(win1, text = "TOP 1", width = 10, height = 4,  
      bg = 'yellow').pack(side = TOP)
```

Метод pack размещает 1-ю метку и прижимает ее вверх

(занимает всю кромку).

```
Label(win1, text = "LEFT 2", width = 10, height = 4,  
      bg = 'red').pack(side = LEFT)
```

При размещении 2-й метки используется оставшаяся область и

метка прижимается влево и т. д. размещаются 3-я и 4-я метки.

```
Label(win1, text = "BOTTOM 3", width = 10, height = 4,  
      bg = 'green').pack(side = BOTTOM)
```

```
Label(win1, text = "RIGHT 4", width = 10, height = 4,  
      bg = 'white').pack(side = RIGHT)
```

**# При изменении размеров окна сохраняется принцип: место
под компоненты выделяется в порядке их размещения.
Если сжимать окно, то компоненты будут пропадать
в порядке обратном их размещения.**

Создаем второй frame:

win2 = Frame(root, bg = '#770077')

Фрейм прижимается вверх (по умолчанию)

растягивается по оси X:

win2.pack(expand = YES, fill = X)

Расположим на нем метку и редактор.

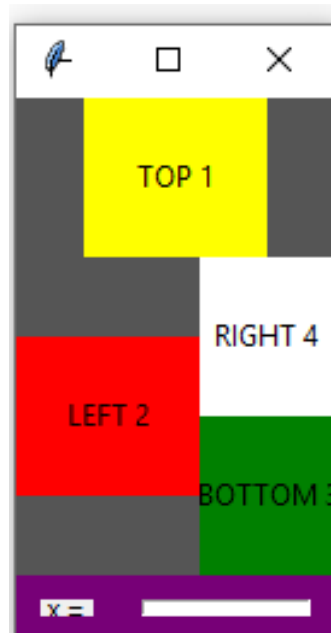
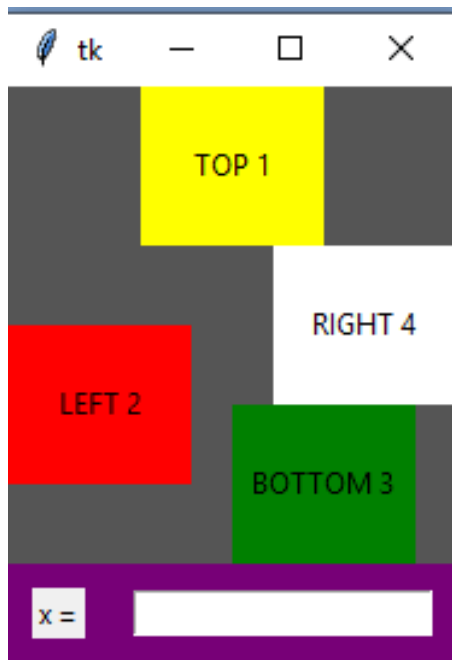
Метка прижата влево и добавляем ей отступы padx и pady:

Label(win2, text = "x=").pack(side = LEFT, padx = 10, pady = 10)

Редактор Entry тоже с отступами и растягивается по оси X:

```
Entry(win2).pack(side = LEFT, padx = 10, pady = 10,  
                expand = YES, fill = X)
```

```
root.mainloop()
```



Подробнее о методе размещения компонент pack можно посмотреть в списке литературы указанном в конце.

Например, посмотреть ссылки на электронные страницы
<https://younglinux.info/tkinter/pack> или
<https://metanit.com/python/tutorial/9.4.php>

Дальше будет пример еще с другими методами размещения виджетов.

Пример 5. Напишем простой «Калькулятор»

```
from tkinter import *      # подключаем tkinter

# Подключаем модуль с диалоговыми окнами tkinter:
from tkinter.messagebox import *

root = Tk( )               # создаем главное окно

# Устанавливаем минимальные и максимальные размеры окна:
root.minsize(width = 350, height = 150)
root.maxsize(width = 500, height = 300)

root.title("Калькулятор")   # заголовок окна
```

Создадим 3 фрейма: fr_xy, fr_op и fr_res
для размещения компонент.

фрейм fr_xy – компоненты для ввода чисел x, y:

fr_xy = Frame(root)

fr_xy.pack(side = TOP, expand = YES, fill = X)

На нем размещаем две метки и два редактора для ввода чисел x, y:

lx = Label(fr_xy, text = "x = ")

lx.pack(side = LEFT, padx = 10, pady = 10)

entX = Entry(fr_xy)

entX.insert(0, 0) # – в редактор записываем в позицию 0 число 0

entX.pack(side = LEFT, padx = 10, pady = 10)

Редактор будет выбран при старте (иметь фокус ввода):

entX.focus()


```
ly = Label(fr_xy, text = "y = ")  
ly.pack(side = LEFT, padx = 10, pady = 10)
```

```
entY = Entry(fr_xy)  
entY.insert(0, 0)  
entY.pack(side = LEFT, padx = 10, pady = 10)
```

Создание фрейма с заголовком fr_or – выбор операции:

```
fr_or = LabelFrame(root, text = "Операция:")  
fr_or.pack(side = TOP, expand = YES, fill = X)
```

Операцию будем выбирать с помощью виджета Radiobutton:

```
oper = ['+', '-', '*', '/'] # – список операций
```

```
# Вводим строковую переменную tkinter, ее свяжем с выбором
# Radiobutton
varOper = StringVar( )
# В цикле создаем 4 кнопки Radiobutton (связываем их
# с созданной переменной varOper):
for op in oper:
    Radiobutton(fr_op, text = op, variable = varOper,
                value = op).pack(side = LEFT,
                                padx = 20, pady = 10)

# Устанавливаем текущее значение '+':
varOper.set(oper[0])
```

Создаем 3-й фрейм fr_res – вычисление значения

и вывод результата:

fr_res = Frame(root)

fr_res.pack(side = TOP, expand = YES, fill = BOTH)

Обработчик кнопки:

def OnButtunResult():

Защищенный блок, будем пытаться перевести текст

из редактора Entry в число:

try:

x = float(entX.get()) # извлекаем число из 1-го редактора

except ValueError:

если не получилось, выдаем сообщение и выходим:

**showerror("Ошибка заполнения", "Переменная x не является
числом")**

```
return
```

```
# Защищенный блок 2:
```

```
try:
```

```
    y = float(entY.get())
```

```
except ValueError:
```

```
    showerror("Ошибка заполнения", "Переменная у не является  
                числом")
```

```
return
```

```
# В переменную op записываем выбранную операцию:
```

```
op = varOper.get( )
```

```
# Вычисляем:
```

```
if op == '+': res = x + y
```

```
elif op == '-': res = x - y
```

```
elif op == '*': res = x * y
```

```
elif op == '/':
```

```
if y != 0: res = x / y
```

```
else: res = 'NAN'
```

```
else:
```

```
res = 'операция выбрана неправильно'
```

```
# Вывод результата на метку:
```

```
lres['text'] = res
```

```
# Обработчик кнопки закончился.
```

```
# Создаем кнопку и метку, к кнопке присоединяем обработчик:
```

```
Button(fr_res, text = "=", width = 10,
```

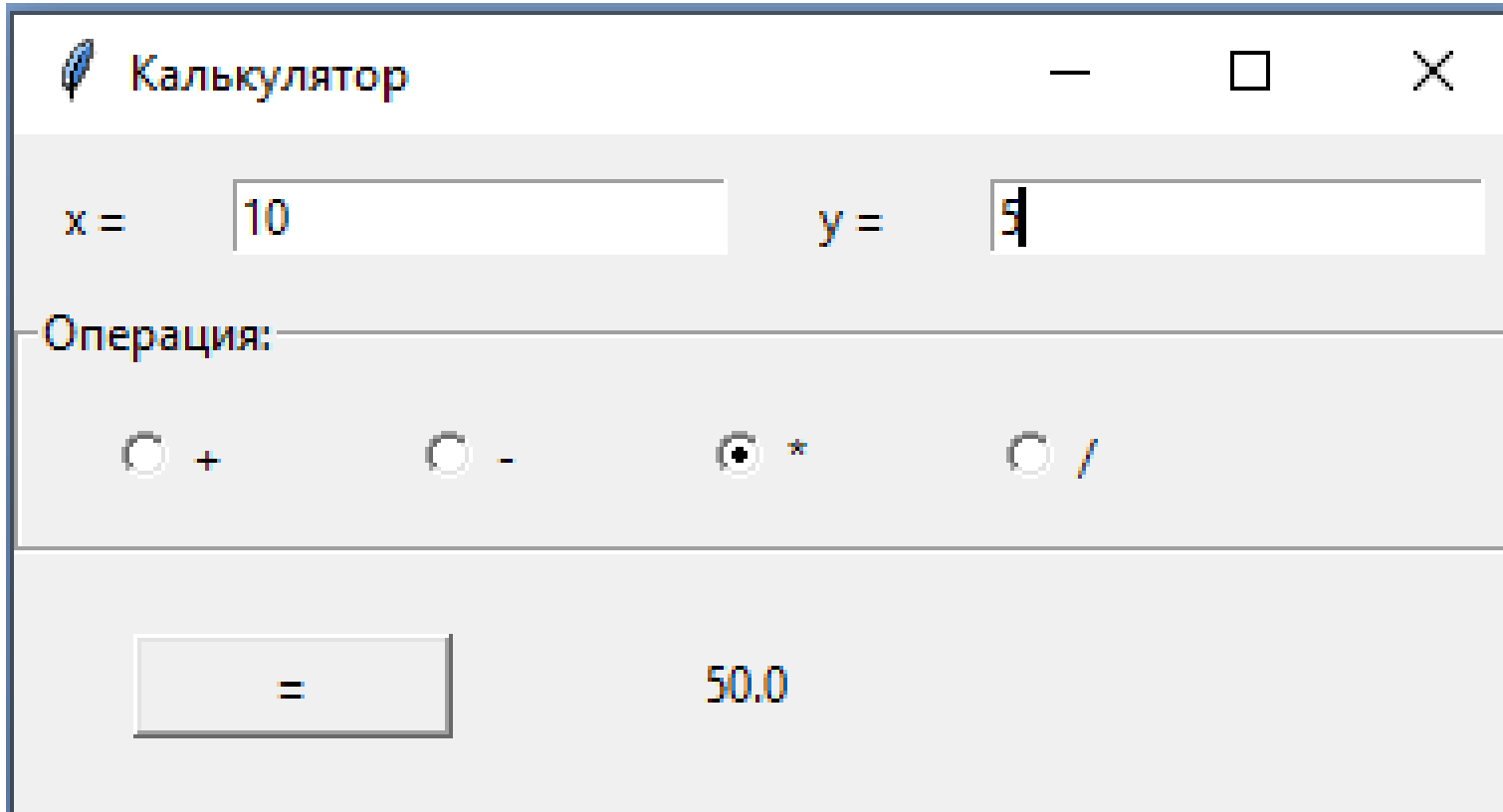
```
command = OnButtunResult).pack(side = LEFT,
```

```
padx = 30, pady = 20)
```

```
lres = Label(fr_res, text = "")
```

```
lres.pack(side = LEFT, padx = 30, pady = 20)
```

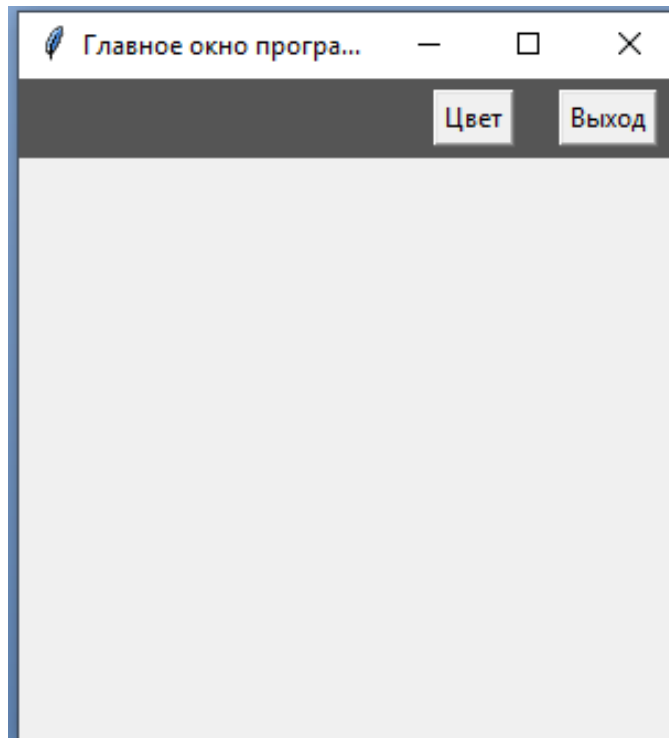
Запуск цикла обработки сообщений:
root.mainloop()



The image shows a screenshot of a Tkinter calculator window titled "Калькулятор". The window has a standard title bar with minimize, maximize, and close buttons. Inside the window, there are two input fields: "x =" with the value "10" and "y =" with the value "5". Below these fields is a section labeled "Операция:" containing four radio buttons for arithmetic operations: "+" (unselected), "-" (unselected), "*" (selected), and "/" (unselected). At the bottom of the window, there is an "=" button and a display area showing the result "50.0".

Пример 6. Дополнительные окна (Toplevel) и стандартные диалоговые окна

В этом примере по нажатию кнопки будем создавать дополнительное окно (Toplevel). Также создадим несколько различных диалоговых окон. Дополнительную информацию см. [1, стр. 558 (Toplevel), 566 (диалоги)] и электронные ресурсы.



```
from tkinter import *
```

```
# подключаем диалоговые окна:
```

```
from tkinter.messagebox import *
```

```
# подключаем диалоговое окно выбор цвета:
```

```
from tkinter.colorchooser import askcolor
```

```
root = Tk( )
```

```
root.title("Главное окно программы")
```

```
root.geometry("300x300+250+250")
```

```
# Создаем фрейм для размещения на нем других компонент:
```

```
win1 = Frame(root, bg = '#555555')
```

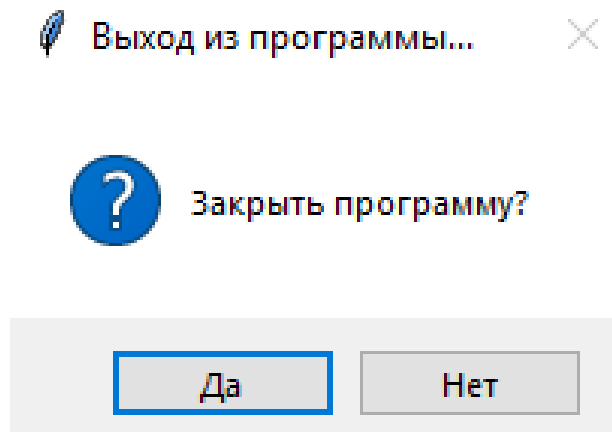


```
# Помещаем фрейм на форму методом pack.  
# Вместо параметра side (прижать к стороне) задаем  
# параметр anchor (якорь): компонент будет прижат к северной  
# (north) стороне и растянут по ширине.  
win1.pack(anchor = "n", expand = YES, fill = X)  
# Обработчик кнопки закрытия программы:  
def closeQuery( ):  
    # При нажатии на кнопку запускаем диалоговое окно.  
    # Спрашиваем у пользователя, закрывать ли программу:  
    if askyesno('Выход из программы...', 'Заккрыть программу? '):  
        showwarning('Диалоговое окно', 'Внимание!!!')  
    # Уничтожаем главное окно и поэтому закрываем программу:  
        root.destroy( )  
    else:  
        showinfo('Диалоговое окно', 'Информация')
```

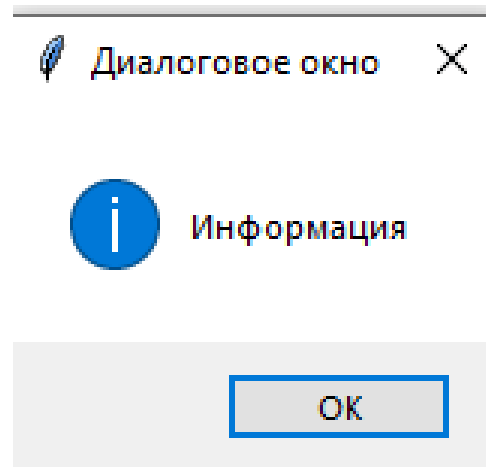
Создаем кнопку закрытия программы:

```
Button(win1, text = 'Выход', command = closeQuery).pack(  
    side = RIGHT, padx = 10, pady = 5)
```

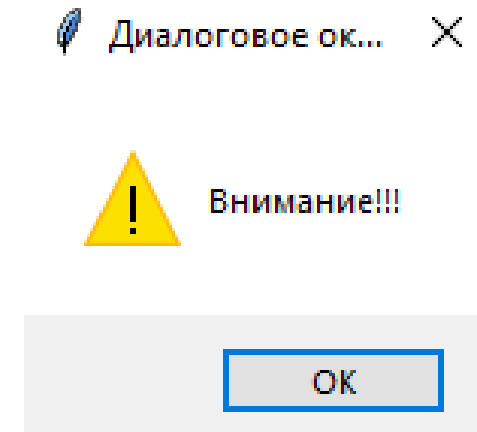
(1)



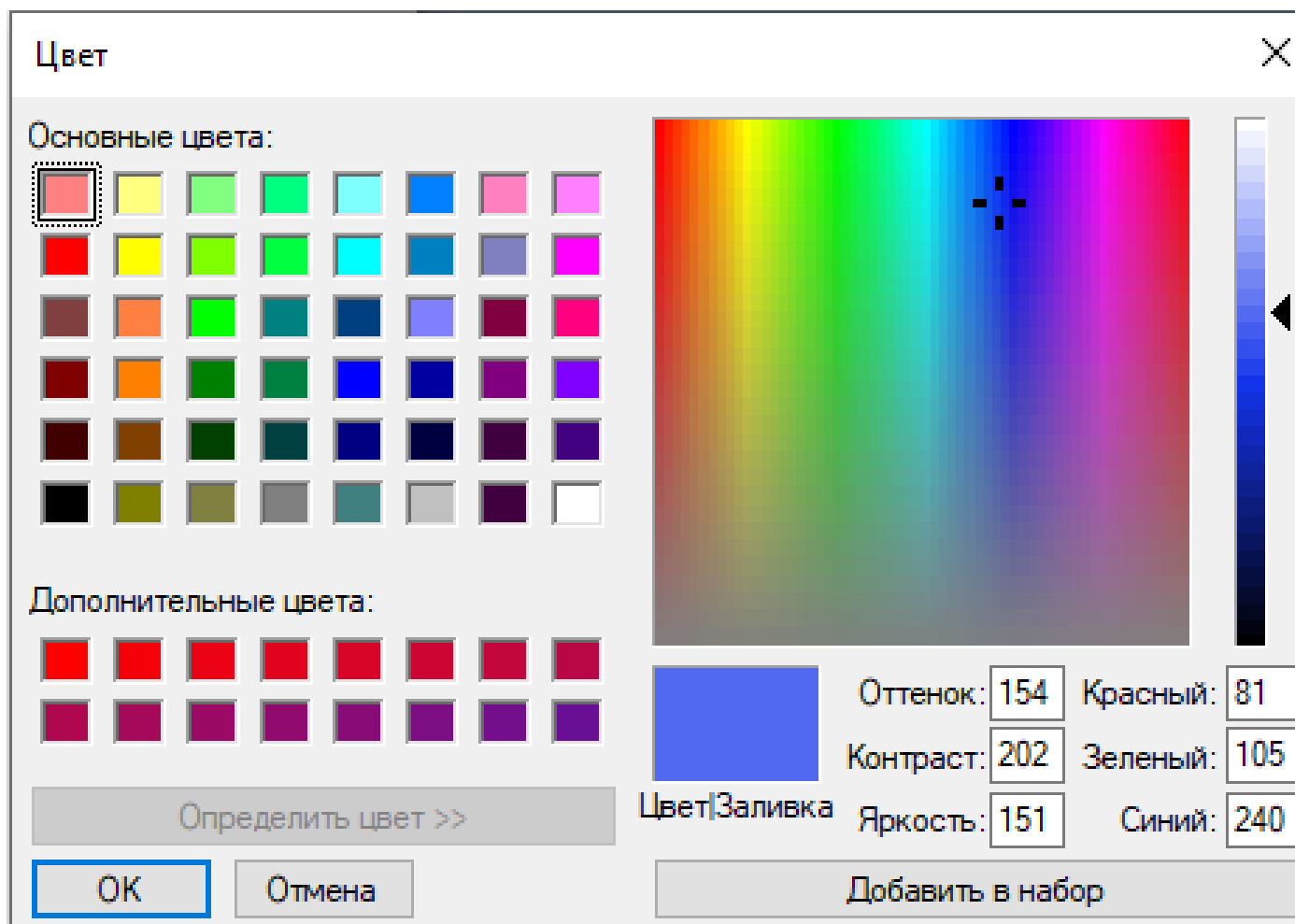
(2)



(3)



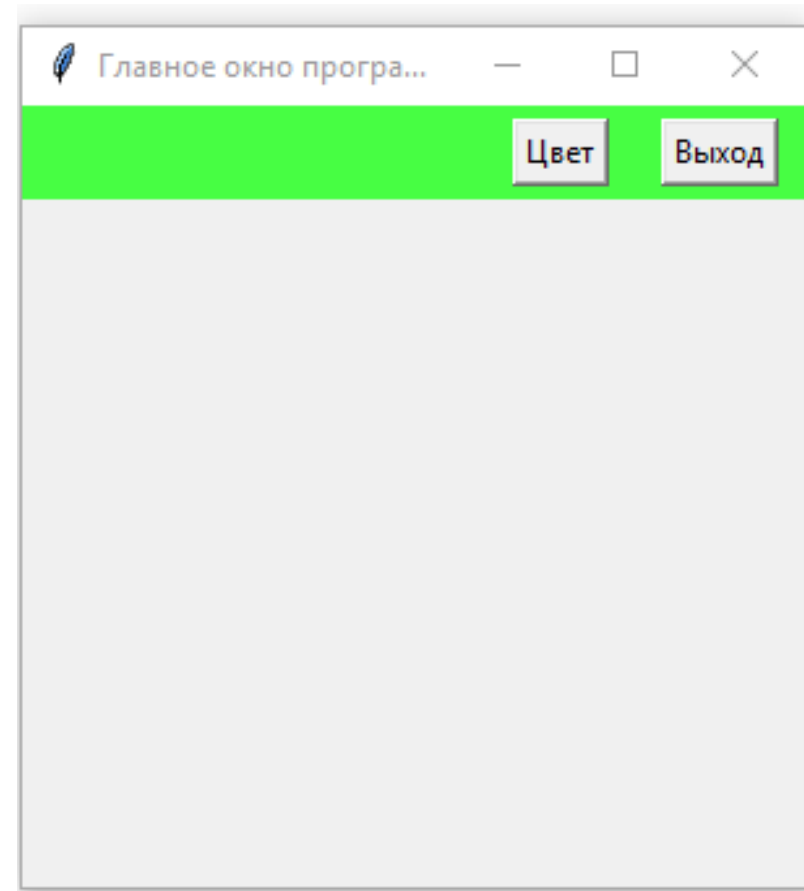
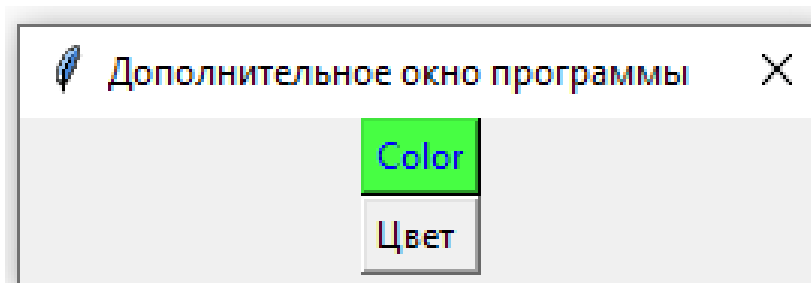
```
# Создаем обработчик кнопки выбора цвета:
def my_set_color(w):
    # Запускаем диалоговое окно выбора цвета:
    res = askcolor( )
    # Печатаем результат res, который вернул диалог:
    print("Диалоговое окно Цвет возвращает результат:", res)
    if res[1]: # если цвет был выбран, то
        w.config(bg = res[1])      # меняем цвет у виджета w
        win1.config(bg = res[1])  # и у фрейма главного окна
```



```
# Обработчик нажатия на кнопку "Цвет" главного окна.  
# В этом обработчике создаем новое окно (виджет Toplevel)  
def newWindow( ):  
    topL1 = Toplevel(root)      # создание нового окна  
    topL1.title("Дополнительное окно программы")  
    topL1.transient(root) # делаем его зависимым от главного окна  
    # topL1.grab_set( ) # – устанавливаем фокус ввода  
    # Создаем две кнопки в этом окне:  
    b1 = Button(topL1, text = 'Color',  
                command = (lambda: my_set_color(b1)), fg = 'blue')  
    b2 = Button(topL1, text = 'Цвет ',  
                command = (lambda: my_set_color(b2)))  
    b1.pack( )  
    b2.pack( )
```

Создаем кнопку "Цвет" главного окна:

```
Button(win1, text = 'Цвет', command = newWindow).pack(  
    side = RIGHT, padx = 10, pady = 5)  
root.mainloop( )
```



Когда размер программы увеличивается, становится сложно искать в тексте нужный код, исправлять и изменять программу. Поэтому код приходится структурировать. Например, выносить в отдельные функции повторяющиеся операции. Для каждого отдельного окна можно создать свой класс и все описывать в нем. Также удобно каждый такой класс выносить в отдельный модуль (файл).

Пример 7. Привязка событий с помощью метода bind

```
from tkinter import *
```

```
root = Tk( )
```

```
root.geometry("400x300+150+100")
```

```
root.title("Click me")
```

Создаем функции-обработчики событий:

def showPosEvent(event):

print('Widget = %s X = %s Y = %s'
 % (event.widget, event.x, event.y))

def onLeftClick(event):

print('Левая клавиша мыши')
showPosEvent(event)

def onMiddleClick(event):

print('Средняя клавиша мыши')
showPosEvent(event)


```
def onRightClick(event):  
    print('Правая клавиша мыши')  
    showPosEvent(event)
```

```
def onDoubleClick(event):  
    print('Левая клавиша мыши - двойной щелчок')  
    showPosEvent(event)
```

```
def onLeftDrag(event):  
    print('Движение')  
    showPosEvent(event)
```

```
def onKeyPress(event):  
    print('клавиша ', event.char)  
    print(event)  
    # Печать всех параметров из event:  
    #for a in dir(event):  
    #     if not a.startswith('__'):  
    #         print(a, '=>', getattr(event, a))
```

```
def onArrowKey(event):  
    print('клавиша стрелка вверх')
```

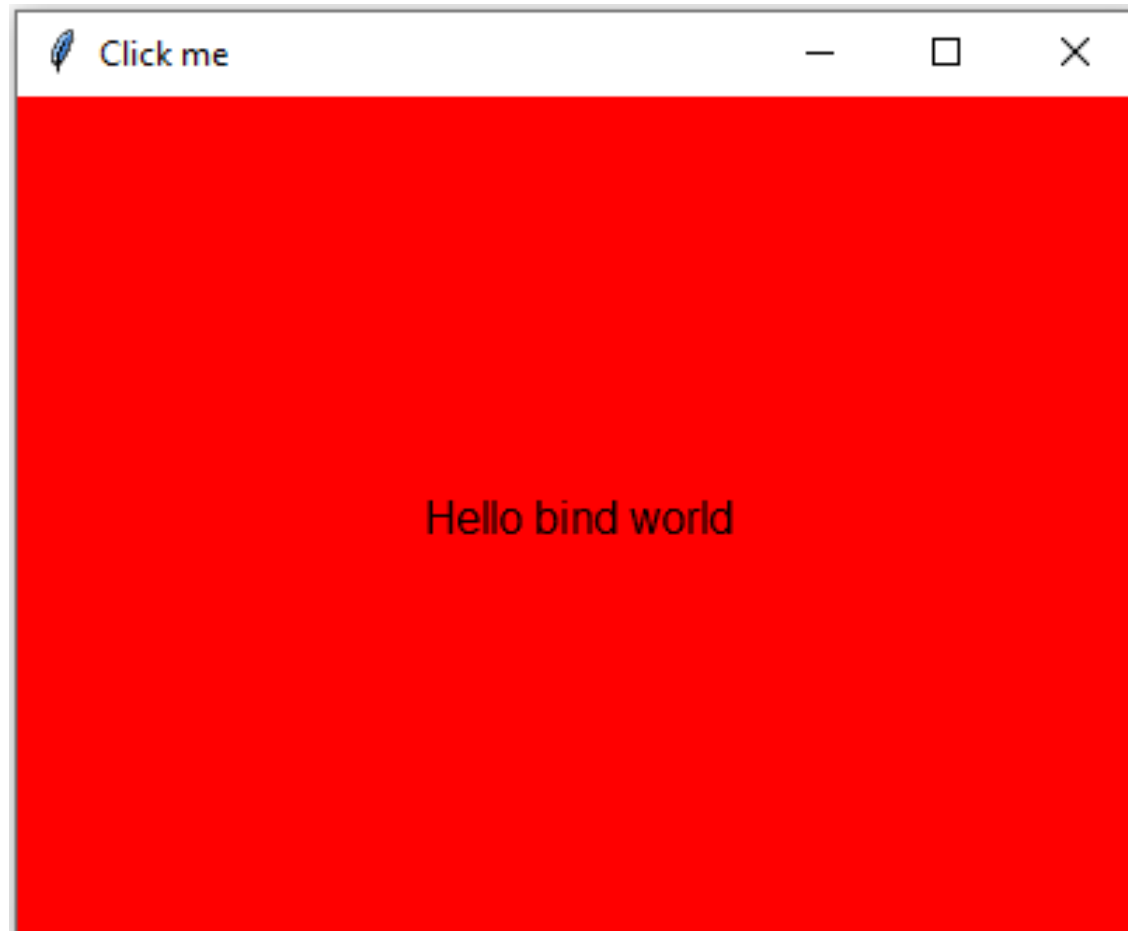
```
w = Label(root, text = 'Hello bind world')  
w.config(height = 5, font = 20)  
w.config(bg = 'red')
```

w.pack(expand = YES, fill = BOTH)
w.bind('<Button-1>', onLeftClick) **# щелчок левой кнопки мыши**
w.bind('<Button-2>', onMiddleClick) **# щелчок средней**
w.bind('<Button-3>', onRightClick) **# щелчок правой**
w.bind('<Double-1>', onDoubleLeftClick) **# двойной щелчок левой**
w.bind('<B1-Motion>', onLeftDrag) **# движение и нажатие левой**
кнопки мыши
w.bind('<KeyPress>', onKeyPress) **# нажатие клавиши на**
клавиатуре
w.bind('<Up>', onArrowKey) **# клавиша стрелка вверх**

w.focus()
mainloop()

Дополнительную информацию см. [1, стр. 585] и электронные ресурсы, например,

https://ru.wikiversity.org/wiki/Курс_по_библиотеке_Tkinter_языка_Python#Привязка_событий



```
Shells
Python
Widget = .!label X = 145 Y = 86
Движение
Widget = .!label X = 146 Y = 84
Движение
Widget = .!label X = 147 Y = 83
Движение
Widget = .!label X = 150 Y = 82
Движение
Widget = .!label X = 153 Y = 81
Движение
Widget = .!label X = 155 Y = 80
Движение
Widget = .!label X = 157 Y = 80
Движение
Widget = .!label X = 159 Y = 78
Левая клавиша мыши
Widget = .!label X = 117 Y = 77
Средняя клавиша мыши
Widget = .!label X = 176 Y = 87
Правая клавиша мыши
Widget = .!label X = 165 Y = 87
клавиша
<KeyPress event keysym=space keycode=32 char=' ' x=156 y=76>
клавиша ф
<KeyPress event keycode=65 char='ф' x=156 y=76>
клавиша стрелка вверх
```

Пример 8. Менеджеры размещения виджетов pack, grid и place

Для размещения компонент мы использовали метод **pack** (упаковщик). Он прижимает компоненты к какому-нибудь краю. Есть еще метод **grid** (сетка), позволяет разбивать контейнер сеткой и размещать компоненты в ее ячейки. Также есть метод **place** (место), при размещении указываются координаты размещения. В одном контейнере эти методы смешивать нельзя!

В главном окне разместим три фрейма, на каждом из них разместим компоненты тремя различными способами.

```
from tkinter import *  
import random          # Подключаем модуль random  
root = Tk( )  
root.title("Менеджеры размещения компонентов")  
root.geometry("400x400+300+250")
```

На первом фрейме используем метод pack:

```
win1 = Frame(root, bg = '#555555')
```

```
win1.pack(expand = YES, fill = BOTH)
```

Разметим метку и кнопку методом pack:

```
Label(win1, text = "Метод pack").pack(side = LEFT,  
                                         padx = 20, pady = 10)
```

При нажатии на кнопку будем слева добавлять еще кнопки:

```
def add1( ):
```

```
    Label(win1, text = "Метка").pack(side=LEFT, padx = 20, pady = 10)
```

```
Button(win1, text = "Добавить", command=add1).pack(side = LEFT,  
                                                    padx = 20, pady = 10)
```

На втором фрейме используем метод grid:

```
win2 = Frame(root, bg = '#888888')
```

```
win2.pack(expand = YES, fill = BOTH)
```

Поместим метку и кнопку в первой строке сетки.

```
Label(win2, text = "Метод grid").grid(row = 0, column = 0)
```

При нажатии на кнопку будем добавлять компоненты

в следующие строки:

```
s = 0
```

```
def add2( ):
```

```
    global s
```

```
    s += 1
```

```
    Label(win2, text = "x" + str(s) + " = ").grid(row = s,  
        column = 0, padx = 20, pady = 10)
```

```
    Entry(win2).grid(row = s, column = 1, padx = 20, pady = 10)
```



```
Button(win2, text = "Добавить", command = add2).grid(row = 0,  
              column = 0, padx = 20, pady = 10)  
add2( ) # Добавили компоненты методом add2  
  
# На третьем фрейме используем метод place:  
win3 = Frame(root, bg = '#BBBBBB')  
win3.pack(expand = YES, fill = BOTH)  
# Поместим кнопку на фрейм.  
# И при ее нажатии будем добавлять метку в случайное место  
# контейнера.  
random.seed( ) # Инициализация генератора случайных чисел  
def add3( ):  
    Label(win3, text = "Метод place").place(  
        x = random.randint(0, 400), y = random.randint(0, 100))
```

```
Button(win3, text = "Добавить", command = add3).place(  
    x = 10, y = 10)  
  
root.mainloop( )
```

Дополнительную информацию см. [1, стр. 726] и электронные ресурсы, например,

ru.wikiversity.org/wiki/Курс_по_библиотеке_Tkinter_языка_Python#Уп_аковщ_ики

Менеджеры размещения компонентов

Метод pack

Добавить

Метка

Метка

Метка

Добавить

x1 =

x2 =

Добавить

Метод place ice

Метод place

Метод place

Метод place

Метод place

Метод place

Пример 9. Виджет Canvas, рисование

```
from tkinter import *
```

```
root = Tk( )
```

```
root.title("Виджет Canvas, рисование ")
```

```
root.geometry("400x400+250+250")
```

```
# создаем фрейм для размещения на нем кнопки:
```

```
win = Frame(root, bd = 1, relief = RAISED)
```

```
win.pack(anchor = "n", expand = YES, fill = X)
```

```
# создаем область рисования Canvas:
```

```
can = Canvas(root)
```

```
can.pack(expand = YES, fill = BOTH)
```

Создаем обработчик кнопки 'Рисовать':

def ris():

can.create_line(20, 50, 200, 40, fill = 'red', width = 5)

can.create_arc(30,150,300,310, fill = 'blue')

can.create_rectangle(300,10, 350,40, width = 2, outline = 'green')

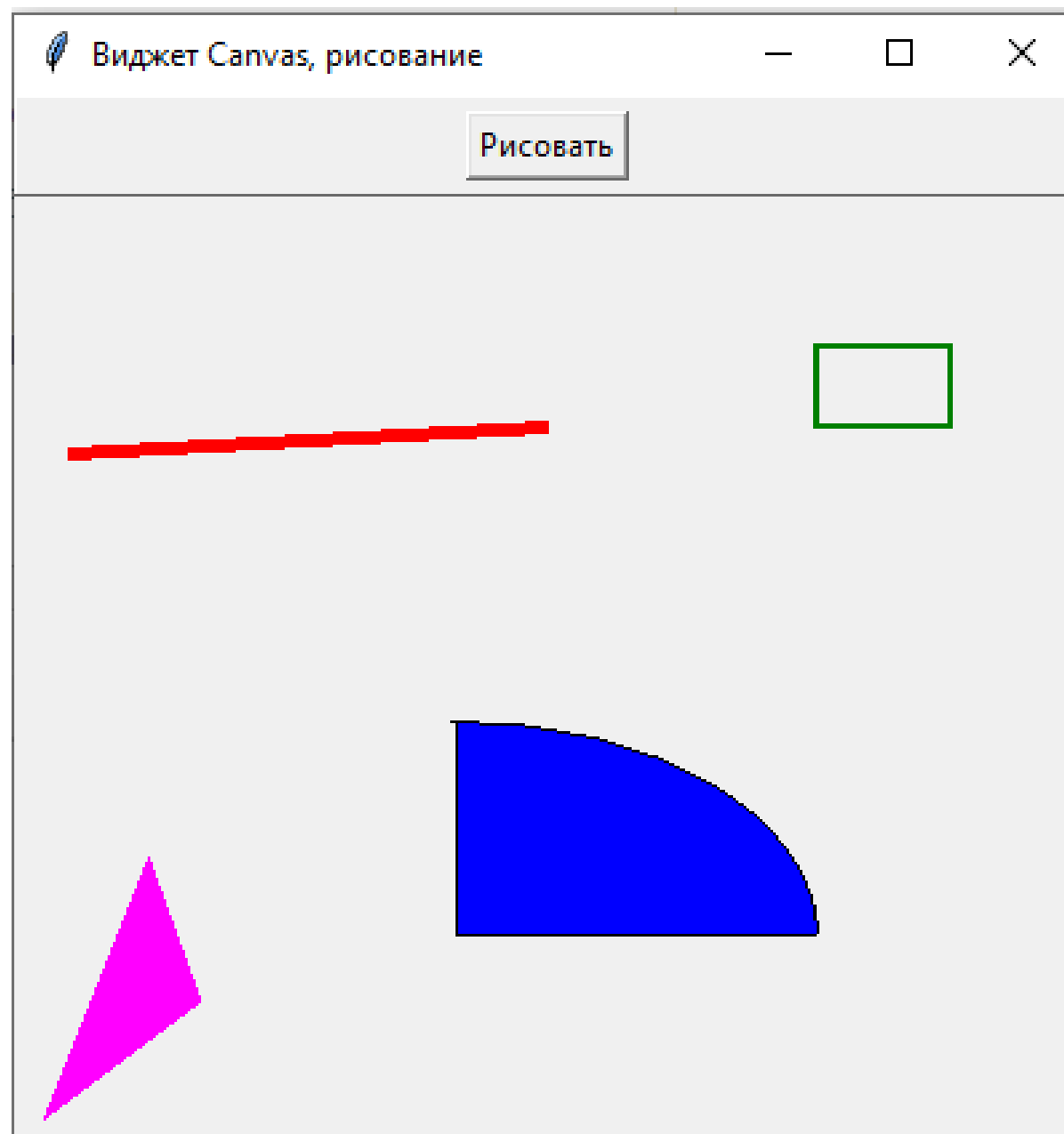
can.create_polygon(50,200,10,300,70,255,fill = 'magenta')

Создаем кнопку:

**Button(win, text = 'Рисовать', command = ris).pack(
padx = 10, pady = 5)**

root.mainloop()

Дополнительную информацию см. [1, стр. 709] и электронные ресурсы.



Пример 10. Динамическое выполнение кода на Python.

Методы `eval` и `exec`.

Методы `eval` и `exec` не относятся к методам `tkinter` и вообще к графическим интерфейсам. Но они могут расширить возможности программы. Методы позволяют выполнить код, записанный в строке. Метод `eval` вычисляет выражение, записанное в строке, и возвращает результат, а `exec` способен выполнять многострочные блоки кода.

В этом примере пользователь в двух редакторах может задать две целые величины `x`, `y`. А в третьем редакторе написать любую функцию от `x`, `y` по правилам Python и по нажатию кнопки, эта функция будет вычислена.

Эти методы опасны, особенно если программа используется в Internet. Пользователь может записать не математическую

функцию, а, например, код удаления файлов. Поэтому нужно как-то ограничить его.

```
from tkinter import *  
from math import * # подключаем математические функции  
root = Tk( )  
root.title('Методы eval и exec')  
# создаем фрейм для размещения компонент задающих x, y:  
win1 = Frame(root)  
win1.pack(anchor = 'n', expand = YES, fill = X)  
  
lx = Label(win1, text = 'x = ')  
lx.pack(side = LEFT, padx = 10, pady = 10)  
  
entX = Entry(win1)  
entX.insert(0, 0)
```



```
entX.pack(side = LEFT, padx = 10, pady = 10)
```

```
entX.focus( )
```

```
ly = Label(win1, text = "y = ")
```

```
ly.pack(side = LEFT, padx = 10, pady = 10)
```

```
entY = Entry(win1)
```

```
entY.insert(0, 0)
```

```
entY.pack(side = LEFT, padx = 10, pady = 10)
```

создаем второй фрейм для задания функции и вычисления:

```
win2 = Frame(root)
```

```
win2.pack(anchor = "n", expand = YES, fill = X)
```

```
Label(win2, text = "Функция: ").pack(side = LEFT,  
    padx = 10, pady = 10)
```

```
entF = Entry(win2)
entF.pack(side = LEFT, padx = 10, pady = 10, expand = YES,
          fill = X)
entF.insert(0, "x + y")
```

Создаем обработчик кнопки:

```
def res( ):
    try:
        x = int(entX.get())
    except ValueError:
        showerror("Ошибка заполнения",
                  "Переменная x не является целым числом")
    return
    try:
        y = int(entY.get())
```

```
except ValueError:
```

```
    showerror("Ошибка заполнения",
```

```
        "Переменная у не является целым числом")
```

```
    return
```

```
    F = entF.get( )      # считываем текст из редактора
```

```
    # print(x, y, eval(F))
```

```
    labF['text'] = eval(F) # выполняем его и результат
```

```
    # записываем на метку
```

```
# Создаем кнопку и метку:
```


```
Button(win2, text = 'Вычислить', command = res).pack(
```

```
    side = LEFT, padx = 10, pady = 5)
```

```
labF = Label(win2, text = " ")
```

```
labF.pack(side = LEFT, padx = 10, pady = 10)
```

```
root.mainloop()
```

 Методы eval и exec

x =

y =

Функция:

6

Литература

1. Лутц М. Программирование на Python, том I, 4-е издание. Пер. с англ. СПб.: Символ-Плюс, 2011. 992 с.
2. Лутц М. Изучаем Python, 4-е издание. Пер.с англ. СПб.: Символ-Плюс, 2011. 1280 с.
3. Лутц М. Программирование на Python, том II, 4-е издание. Пер. с англ. СПб.: Символ-Плюс, 2011. 992 с.

Интернет-ресурсы

4. <https://www.python.org>, <https://docs.python.org/3/> (Documentation Python)
5. <https://docs.python.org/3/library/tkinter.html#module-tkinter> (tkinter doc)
6. <http://www.pythonware.com/library/>, <http://effbot.org/tkinterbook/>
7. <https://www.tcl.tk/>, <https://www.tcl.tk/man/tcl8.6/TkCmd/contents.htm> (Tk Documentation)
8. <https://tkdocs.com/tutorial/index.html> (TkDocs)

Интернет-ресурсы

(русскоязычные электронные учебники)

9. <https://younglinux.info/tkinter.php> (Tkinter. Программирование GUI на Python. Курс)
10. <https://metanit.com/python/tutorial/9.1.php> (Глава 9 электронного учебника по Python. «Создание графического интерфейса»)
11. https://ru.wikiversity.org/wiki/Курс_по_библиотеке_Tkinter_языка_Python#Entry (Курс по библиотеке Tkinter языка Python)
12. <https://pythonru.com/uroki/obuchenie-python-gui-uroki-po-tkinter#toc> (Обучение Python GUI (уроки по Tkinter))