1.1. КРАТКИЕ СВЕДЕНИЯ О БИБЛИОТЕКЕ TK И МОДУЛЕ TKINTER

tkinter — это модуль Python, который работает с библиотекой **Tk**. С помощью этой библиотеки написано достаточно большое количество проектов. Основная причина использования этой библиотеки при создании приложений с GUI на языке Python заключается в том, что установочный файл Python уже включает модуль **tkinter** в состав стандартной библиотеки. Поэтому для использования возможностей библиотеки **Tk** достаточно подключить модуль **tkinter** с помощью инструкции **import**.

Возможные варианты подключения модуля **tkinter**:

• import tkinter;

• from tkinter import*;

• import tkinter as tk.

Можно импортировать из модуля **tkinter** отдельные классы, но такой способ используется достаточно редко. Библиотека **Tk** содержит компоненты графического интерфейса пользователя, написанные на языке Tcl. Не вдаваясь в подробности, модуль **tkinter** можно охарактеризовать как переводчик с языка Python на язык Tcl, который понимает библиотека **Tk**. Следует отметить, что все элементы управления графического интерфейса, создаваемые с использованием модуля **tkinter**, принято называть **виджетами** (от *англ.* **widgets**).

1.2. ОСНОВНЫЕ ЭТАПЫ СОЗДАНИЯ ОКОННОГО ИНТЕРФЕЙСА

Процесс создания графических интерфейсов представляет собой типичный пример так называемой быстрой разработки. В основе ее лежит технология визуального проектирования и событийного программирования. Смысл этой технологии заключается в том, что основную часть рутинной работы по созданию приложения берет на себя среда разработки. Задача же программиста заключается в создании различных диалоговых окон (визуальное проектирование) и функций по обработке событий (событийное программирование), происходящих в этих окнах. В качестве событий окна могут выступать следующие события: срабатывание временного фактора (таймера), нажатие кнопки, ввод текста, выбор пункта меню и др.

Для создания приложения с графическим интерфейсом, как правило, необходимо выполнить следующие этапы:

• создать главное окно приложения;

• создать необходимые виджеты и выполнить конфигурацию их
свойств (опций);

• определить события, на которые будет реагировать программа;

• написать обработчики событий, т. е. программные коды, которые будут запускаться при наступлении того или иного события;

• расположить виджеты в главном окне приложения;

• запустить цикл обработки событий.

Последовательность выполнения этапов может отличаться от вышеперечисленной, но первый и последний пункты всегда остаются на своих местах.

Каждая программа по созданию графического интерфейса должна начинаться с вызова конструктора **Tk()**, создающего объект окна. При помощи метода **geometry()** этого объекта можно дополнительно задать размеры окна, передав в качестве аргумента строку **«ширина × высота»**. Кроме этого, можно также создать заголовок окна, задав методу **title()** строковый аргумент с названием заголовка.

Если размеры и заголовок не указаны, то будут использоваться значения, заданные по умолчанию.

После создания любого виджета его необходимо разместить в окне. Расположение виджетов в окне приложения может быть произвольным, но, как правило, интерфейс приложения должен быть продуман до мелочей и организован по определенным стандартам.

Для размещения виджетов в окне приложения предназначены специальные методы, которые называются **менеджерами размещений**:

• **pack()** — располагает виджеты по определенной стороне окна при помощи аргумента **side**, который может принимать следующие четыре значения: **TOP** (вверху), **BOTTOM** (внизу), **LEFT**

(слева**)** и **RIGHT** (справа); значение по умолчанию — **TOP** (т. е. виджеты располагаются вертикально);

• **place()** — помещает виджет в точку окна с координатами **X** и **Y**, переданными ему в качестве аргументов с числовыми значениями в пикселях;

• **grid()** — позволяет разместить виджет в ячейку таблицы в соответствии с заданными числовыми параметрами **row** и **column**, определяющими номер строки и номер столбца для этой ячейки.

Завершает каждую программу так называемый цикл обработки событий окна, вызываемый методом **mainloop()**. Этот метод реагирует на любые действия пользователя в окне во время работы программы (закрытие окна для выхода из программы, изменение размеров окна и др.).

Объектно-ориентированный подход к созданию окна в Tkinter, пример:

1.3. ТЕКСТОВОЕ ПОЛЕ. ВИДЖЕТ LABEL. МЕТОД PACK()

Самым простым примером виджета может служить виджет **Label** (метка), который отображает обычный текст или изображение в окне приложения. Для создания метки используется конструктор **Label()**, которому в качестве аргументов необходимо передать имя окна и текст для самой метки в виде **text = „строка”**. Дополнительно в качестве аргументов конструктору можно задать вид и размер шрифта, его цвет и насыщенность, цвет фона и др. Шрифт можно задать двумя способами: в виде строки или в виде кортежа. Второй вариант является более предпочтительным, особенно в случае, когда имя шрифта состоит из двух и более слов.

Пример задачи: вывести текст в окне приложения.

Решение:

```
from tkinter import *
window = Tk() #Создаем окно
window.title("Вывод текста") #Создаем заголовок окна
window.geometry("500x300") #Задаем размеры окна
#Создаем первую метку
label = Label(window, text = "Кафедра Информатики, математики и
физики", font = "Arial 16") #Задаем шрифт как строку
label.pack(side = TOP) #Размещаем метку в окне
#Создаем вторую метку
label_1 =Label(window, text = "ФГБОУ ВО Братский государственный
университет", font = ("Arial", 24, "bold")) #Задаем шрифт как кортеж
label_1.pack(padx = 150, pady = 500) #Размещаем метку в окне
```

#Цикл обработки событий окна

window.mainloop()

Результат работы программы представлен на рисунке 1.

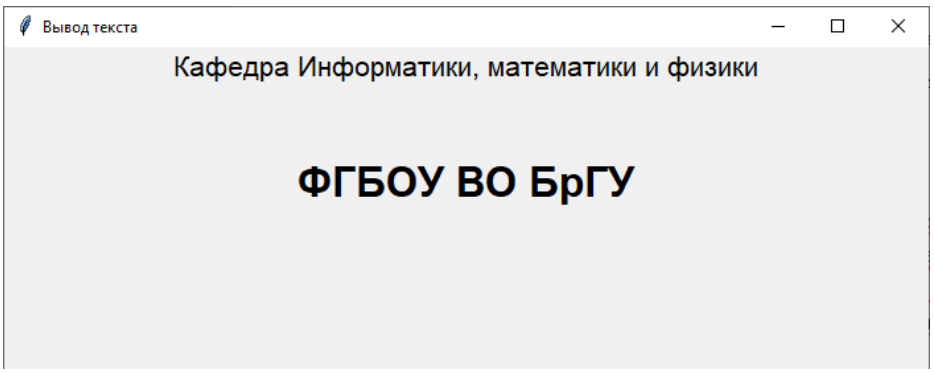


Рис. 1

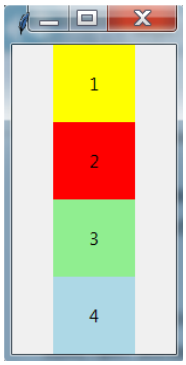
Если значение аргумента **side** метода **pack()** не задано, то поумолчанию будет использоваться значение **TOP**, т. е. создаваемые виджеты будут располагаться вертикально один под другим. Если **side = LEFT**, то все создаваемые виджеты будут располагаться друг за другом слева направо и т. д.

Пример задачи: реализовать различные варианты расположения текста в окне приложения.

Решение:

```
from tkinter import *
window = Tk() #Создаем окно
lab_1 = Label(width=8, height=3, bg='yellow', text="1")# Создаем
четыре метки
lab_2 = Label(width=8, height=3, bg='red', text="2")
lab_3 = Label(width=8, height=3, bg='lightgreen', text="3")
lab_4 = Label(width=8, height=3, bg='lightblue', text="4")
#Располагаем метки вертикально сверху вниз
lab_1.pack()
lab_2.pack()
lab_3.pack()
lab_4.pack()
#Располагаем метки вертикально снизу вверх
lab_1.pack(side = BOTTOM)
lab_2.pack(side = BOTTOM)
lab_3.pack(side = BOTTOM)
lab_4.pack(side = BOTTOM)
#Располагаем метки вертикально слева направо
lab_1.pack(side = LEFT)
lab_2.pack(side = LEFT)
lab_3.pack(side = LEFT)
lab_4.pack(side = LEFT)
#Располагаем метки вертикально справа налево
lab_1.pack(side = RIGHT)
lab_2.pack(side = RIGHT)
lab_3.pack(side = RIGHT)
lab_4.pack(side = RIGHT)
#Комбинированное расположение меток
lab_1.pack(side = BOTTOM)
lab_2.pack(side = TOP)
lab_3.pack(side = LEFT)
lab_4.pack(side = RIGHT)
#Комбинированное расположение меток
lab_1.pack(side = LEFT)
lab_2.pack(side = LEFT)
lab_3.pack(side = BOTTOM)
```

Результат работы программы представлен на рисунке 2.



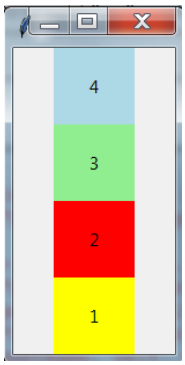


Рис.2

1.4. ГРУППИРОВКА ВИДЖЕТОВ. ВИДЖЕТ FRAME

Из результатов работы программы видно, что расположить четыре метки в виде квадрата (две метки сверху, две метки сверху), варьируя значения аргумента **side** не получается. Для решения задачи оптимизации размещения виджетов в окне приложения (в частности, для группировки виджетов в определенном порядке) используют вспомогательный виджет — фрейм (рамка), который является объектом класса **Frame**. В этом случае фреймы располагают в главном окне, а виджеты — во фреймах.

Объект **фрейм** создается при помощи конструктора **Frame()**, которому в качестве аргумента передается имя окна. После этого имя фрейма может быть передано в качестве первого аргумента конструктора виджета, чтобы указать, что именно данный фрейм, а не окно, является контейнером для виджета. При добавлении виджета во фрейм с помощью аргумента **side** можно указывать его привязку к определенной стороне фрейма, используя все те же константы **TOP**, **BOTTOM**, **LEFT** и **RIGHT**.

Пример задачи: с помощью фрейма реализовать вариант расположения четырех виджетов **Label** в виде квадрата.

Решение:

```
from tkinter import *
window = Tk() #Создаем окно
frame_top = Frame(window)#Фрейм для верхнего ряда меток
frame_bot = Frame(window)#Фрейм для нижнего ряда меток
# Создаем четыре метки с привязкой к конкретному фрейму, а не окну
lab_1 = Label(frame_top, width=8, height=3,
bg='yellow', text="1")
lab_2 = Label(frame_top, width=8, height=3,
bg='red', text="2")
lab_3 = Label(frame_bot, width=8, height=3,
bg='lightgreen', text="3")
lab_4 = Label(frame_bot, width=8, height=3,
bg='lightblue', text="4")
# Располагаем фреймы вертикально один под другим
frame_top.pack()
frame_bot.pack()
#Располагаем метки во фреймах слева направо
lab_1.pack(side = LEFT)
lab_2.pack(side = LEFT)
lab_3.pack(side = LEFT)
lab_4.pack(side = LEFT)
#Цикл обработки событий окна
```

```
window.mainloop()
```

Результат работы программы представлен на рисунке 3.

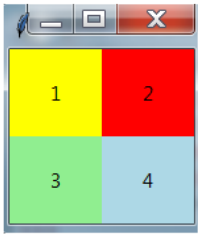


Рис.3

В библиотеке **Tk()** существует еще один класс **LabelFrame**. Объектом этого класса является фрейм с подписью, которую можно задать с помощью аргумента **text**.

Пример задачи: реализовать предыдущую задачу с помощью фрейма с подписью.

Решение: для решения задачи достаточно создать фреймы для меток с помощью конструктора класса **LabelFrame**.

```
from tkinter import *
window = Tk() #Создаем окно
frame_top = LabelFrame(window, text = "Верхний ряд")#Фрейм для
верхнего ряда меток
frame_bot = LabelFrame(window, text = "Нижний ряд" )#Фрейм для
нижнего ряда меток
# Создаем четыре метки с привязкой к конкретному фрейму, а не окну
lab_1 = Label(frame_top, width=8, height=3,
bg='yellow', text="1")
lab_2 = Label(frame_top, width=8, height=3,
bg='red', text="2")
lab_3 = Label(frame_bot, width=8, height=3,
bg='lightgreen', text="3")
lab_4 = Label(frame_bot, width=8, height=3,
bg='lightblue', text="4")
# Располагаем фреймы вертикально один под другим
frame_top.pack()
frame_bot.pack()
#Располагаем метки во фреймах слева направо
lab_1.pack(side = LEFT)
lab_2.pack(side = LEFT)
lab_3.pack(side = LEFT)
lab_4.pack(side = LEFT)
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 4.

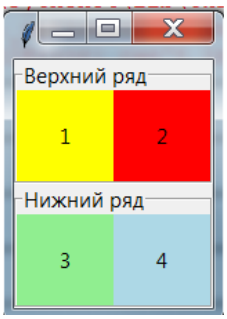


Рис.4

Метод **pack()** может получать дополнительные аргументы:

- **fill** — определяет возможность виджета заполнять свободное пространство вдоль какой-либо из осей, например, вдоль оси **x**: **fill = X**; значение по умолчанию — **NONE**, другие значения — **BOTH**, **X**, **Y**;
- **expand** — определяет положение виджета при расширении окна во весь экран; значение по умолчанию **expand = 0** определяет положение виджета вверху по вертикали и посередине по горизонтали; **expand = 1** определяет положение виджета посередине по вертикали и посередине по горизонтали;
- **padx, pady** — задают внешние отступы виджета от границ окна вдоль соответствующих осей на определенное количество пикселей (количество пикселей до края по горизонтали и по вертикали);
- **ipadx, ipady** — задают внутренние отступы виджета от границ окна вдоль соответствующих осей на определенное количество пикселей (количество пикселей до края по горизонтали и по вертикали);
- **anchor** (якорь) — определяет положение виджета в окне или фрейме в терминах сторон света и может принимать следующие значения: **N** (**North** — север), **S** (**South** — юг), **W** (**West** — запад), **E** (**East** — восток), а также их комбинации (например, **SE** (юго-восток, т. е. правый нижний угол)).

Пример задачи: реализовать различные варианты расположения и заполнения метки с текстом с помощью различных значений аргументов метода **pack()**.

Решение:

```
from tkinter import *
window = Tk() #Создаем окно
window.title("Вывод текста") #Создаем заголовок окна
window.geometry("500x300") #Задаем размеры окна
#Создаем метку
label = Label(window, bg = "blue" , fg = "yellow",text =
"БрГУ",font = "Arial 28") #Задаем шрифт как строку
#Размещаем метку в окне по вертикали — вверху, по горизонтали —
посередине
label.pack()
#Размещаем метку в окне по вертикали — посередине, по горизонтали
— посередине
label.pack(expand = 1)
#Заполняем всё пространство по горизонтали
label.pack(expand = 1, fill = X)
#Заполняем всё пространство по обеим осям
label.pack(expand = 1, fill = BOTH)
#Размещаем метку с текстом в левом верхнем углу
label.pack(anchor = NW)
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 5.

Рис.5

1.5. УПРАВЛЕНИЕ РАБОТОЙ ПРИЛОЖЕНИЯ С ПОМОЩЬЮ КНОПОК. ВИДЖЕТ BUTTON

Виджет **Button** — это элемент управления графического интерфейса **Кнопка**, который предназначен для совершения определенных действий в окне приложения. Кнопка может содержать текст или Изображение, передающее смысловое назначение кнопки.

Объект **Button** создается с помощью одноименного конструктора **Button()**. В качестве аргументов конструктору передается имя окна и набор опций. Каждый аргумент определяется как пара «**аргумент =значение»**. В качестве аргументов могут выступать цвет фона и текста, шрифт, ширина рамки и др. Определяющим аргументом является аргумент **command**. В качестве значения этого аргумента указывается имя функции или метода, которые должны быть вызваны при нажатии пользователя на данную кнопку (обычно в графических библиотеках такие функции называются **обработчиками событий**).

Размер кнопки по умолчанию определяется шириной и высотой текста. Для изменения этих параметров можно воспользоваться свойствами **width** и **height**. При этом единица измерения — знакоместо.

Краткое описание наиболее популярных аргументов приведенов таблице 1.

Таблица 1

Аргумент	Описание

Activebackground	Цвет фона активного элемента во время нажатия и установки курсора над кнопкой
Activeforeground	Цвет текста активного элемента во время нажатия и установки курсора над кнопкой
bg	Цвет фона
fg	Цвет переднего плана (цвет текста)
bd	Ширина рамки в пикселях (по умолчанию bd = 2)
font	Шрифт
height	Высота кнопки (для текста — в количестве строк, для изображений — в пикселях)
width	Ширина кнопки (для текста — в символах, для изображений — в пикселях)
highlightcolor	Цвет рамки при наведении курсора
command	Функция, вызываемая при нажатии на кнопку
image	Изображение для вывода вместо текста
justify	Вид выравнивания (LEFT — по левому краю, CENTER —по центру, RIGHT — по правому краю)
padx	Количество пикселей до границы рамки по горизонтали
pady	Количество пикселей до границы рамки по вертикали
relief	Вид рельефности рамки: SUNKEN — утопленная, RIDGE — выпуклая кайма, RAISEN — выпуклая, GROOVE — канавка
state	Состояние: NORMAL — рабочее, DISABLED — отключена
underline	Порядковый номер символа в тексте, который надо подчеркнуть (по умолчанию-1)
wraplength	Параметр, определяющий ширину, в которую вписывается текст

Следует отметить, что большинство аргументов, представленных в таблице 1, могут использоваться для обоих описанных выше виджетов (**Label** и **Button**) (у меток отсутствуют аргументы **command** и **state**). Значения, присвоенные различным аргументам, определяют внешний вид виджета. Аргументы можно изменять.

Библиотека **Tkinter** поддерживает следующие способы конфигурирования свойств виджетов:

- при создании объекта с помощью конструктора;
- с помощью методов **config()** или **configure()**;
- путем обращения к свойству виджета как к элементу словаря.

Пример задачи: реализовать различные способы конфигурирования свойств кнопки.

```
    from tkinter import *
    window = Tk() #Создаем окно
    window.title("Изменение свойств") #Создаем заголовок окна
    #Создаем кнопку
    btn = Button(window, bg = "blue", fg ="yellow", text = "Запуск
    программы",font = "Arial 16") #Задаем свойства кнопки в
    конструкторе
    #Функция для изменения свойств кнопки
    def change(): #Задаем свойства кнопки как элементы словаря
        btn['text'] = "Программа запущена..."
        btn['bg'] = '#000000'
        btn['fg'] = '#ffffff'
    #Связываем кнопку с функцией-обработчиком событий
    btn.configure(command = change) #Изменяем свойства кнопки с
помощью метода configure()
    #Размещаем кнопку в окне
    btn.pack()
    #Цикл обработки событий окна
    window.mainloop()
```

Результат работы программы представлен на рисунке 6.

Рис.6

Для получения значения каждого аргумента любого виджета можно воспользоваться методом cget(), передав ему в качестве аргумента имя соответствующего аргумента.

Задать цвет текста или фона можно двумя способами:

- в виде текстовой строки, например: “**red**”, “**blue**”, “**yellow**”, “**lightgreen**” и т. д.;
- в виде строкового представления шестнадцатеричного кода, например: “**#ff0000**” — красный, “**#ff7d00**” — оранжевый, “**#ffff00**” — желтый, “**#00ff00**” — зеленый, “**#007dff**” — голубой, “**#0000ff**” — синий, “**#7d00ff**” — фиолетовый, “**#000000**” — черный, “**#ffffff**” — белый, “**#555555**” — серый.

Пример задачи: написать программу, которая будет создавать две кнопки с разной функциональностью: при нажатии на одну из них происходит изменение цвета окна, а при нажатии на другую —выход из приложения.

Решение:

```
    #Функция, переключающая цвет окна с желтого на зеленый и обратно
    def color_switch():
        if window.cget("bg") == "yellow": #Получаем цвет фона с
помощью метода cget()
            window.configure(bg = "green") # Цвет фона зеленый
(используем метод configure())
        else:
            window['bg'] = "yellow" #Цвет фона желтый (используем
"терминологию" словаря)

    from tkinter import *
    window = Tk() #Создаем окно
    window.title("Работа с кнопками") #Создаем заголовок окна
    window.geometry("500x300") #Задаем размеры окна
    #Создаем кнопку для выхода из программы
    btn_exit = Button(window, text = "Выход", bg ="#ff0000", fg =
"green", width = 12, command = lambda: window.destroy())
    #Создаем кнопку для изменения цвета фона
    btn_switch = Button(window, text = "Цвет окна",bg = "blue", fg =
"red", width = 15, font = ("Arial",16, "bold"), command =
color_switch)
    btn_switch.pack(padx = 150, pady = 50) #Размещаем обе кнопки в
окне
    btn_exit.pack(padx = 150, pady = 20)
    #Цикл обработки событий окна
    window.mainloop()
```

Результат работы программы представлен на рисунке 7.

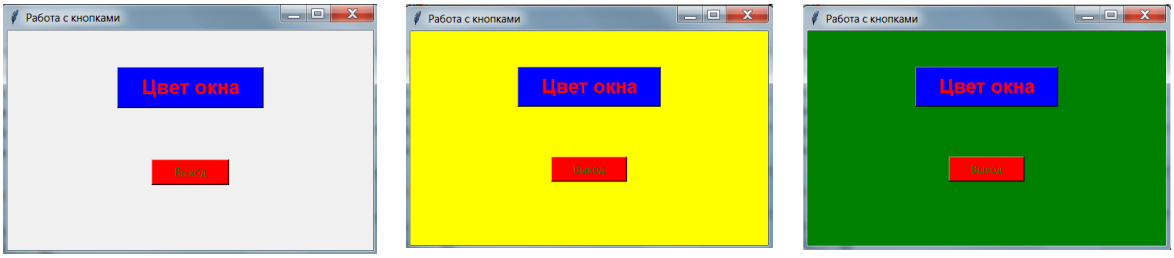


Рис.7

Пример задачи: реализовать объектно-ориентированную версию предыдущей программы.

Решение: создадим в программе собственный класс **My_Button**, объектами которого будут кнопки, которые могут выполнять два различных действия: изменять цвет фона окна и осуществлять выход из программы. В классе **My_Button** создадим следующие методы:

- конструктор с аргументами;
- метод **setFunc()** для конфигурирования обработчика событий; для преобразования строки с именем функции в исполняемый код в данном методе применим встроенную функцию **eval()**;
- метод **color_change()** по изменению цвета фона окна;
- метод **m_exit()**, реализующий выход из приложения с помощью встроенной функции **window.destroy().**

```
from tkinter import *
class My_Button:

#Конструктор с аргументами
    def __init__(self, mwindow, mtext = "Цвет окна", mwidth = 15,
mheight = 3,mbg = "blue", mfg = "red", mpdx =150, mpdy = 50):
        self.btn = Button(mwindow, text = mtext,width = mwidth,
height = mheight,bg = mbg, fg = mfg)
        self.btn.pack(padx = mpdx, pady = mpdy)
#Конфигурирование функции обработчика событий
    def setFunc(self, func):
        self.btn['command'] = eval('self.' + func)
#Функция, переключающая цвет окна с желтого на зеленый и обратно
    def color_change(self):
        if window.cget("bg") == "yellow":
            #Получаем цвет фона с помощью метода cget()
            window.configure(bg = "green") # Цвет фона зеленый
(используем метод configure())
        else:
            window['bg'] = "yellow" #Цвет фона желтый (используем
"терминологию" словаря)
#Функция для выхода из программы
    def m_exit(self):
        window.destroy()
window = Tk() #Создаем окно
window.title("Работа с кнопками") #Создаем заголовок окна
window.geometry("500x300") #Задаем размеры окна
#Создаем кнопку для изменения цвета фона
btn_switch = My_Button(window)
btn_switch.setFunc('color_change')#Связываем кнопку с обработчиком
(функция color_change)
#Создаем кнопку для выхода из программы
btn_exit = My_Button(window, "Выход", 12,2,"#ff0000", "green", 150,
20)
btn_exit.setFunc('m_exit')#Связываем кнопку с обработчиком (функция
m_exit)
#Цикл обработки событий окна
window.mainloop()
```

1.6. ВЫВОД СООБЩЕНИЙ С ПОМОЩЬЮ ДИАЛОГОВЫХ ОКОН. МОДУЛЬ MESSAGEBOX

В программе с графическим интерфейсом вывод сообщений пользователю реализуется в виде различных диалоговых окон. Для работы с диалоговыми окнами в пакете **tkinter** предназначены несколько модулей.

Любой модуль пакета необходимо импортировать отдельно, т. е. кроме инструкции **from tkinter import*** необходима дополнительная инструкция для подключения конкретного модуля. За вывод конкретного окна отвечает соответствующий метод.

Методы для вывода диалоговых окон с различными сообщениями содержатся в модуле **messagebox.**

Ниже представлены возможные варианты подключения этого модуля к программе и вызова его методов:

- **import tkinter.messagebox → tkinter.messagebox.askyesno();**
- **from tkinter.messagebox import* → askyesno();**
- **from tkinter import messagebox → messagebox.askyesno();**
- **from tkinter import messagebox as mb → mb.askyesno()** (вместо **mb** может быть любой идентификатор).

При создании диалогового окна можно указать заголовок окна, а также само выводимое сообщение. Оба этих параметра задаются в виде строковых значений для двух аргументов соответствующего метода.

В таблице 2 перечислены методы для вывода на экран диалоговых окон с различными сообщениями.

Таблица 2

Метод	Кнопки
showinfo()	OK
showwarning()	OK
showerror()	OK
askquestion()	Yes (возвращает строку ‘yes’) и No (возвращает строку ‘no’)
askokcancel()	OK (возвращает 1) и Cancel (не возвращает ничего)
askyesno()	Yes (возвращает 1) и No (не возвращает ничего)
askretrycancel()	Retry (возвращает 1) и Cancel (не возвращает ничего)

Методы, которые выводят диалоговое окно, содержащее единственную кнопку **ОК**, не возвращают никакого значения при нажатии на нее пользователем. Если же метод возвращает значение, то это значение можно использовать для дальнейшего условного ветвления.

Из данных таблицы 2 видно, что:

- из всех методов только метод **askquestion()** возвращает два значения;
- кнопка **No** метода **askyesno()** не возвращает значения;
- кнопка **Cancel** не возвращает значения.

Каждый из всех вышеперечисленных методов может иметь третий аргумент. Например, чтобы получить в диалоговом окне три кнопки (**Abort**, **Retry** и **Ignore**) в качестве третьего аргумента можно задать аргумент **type** = **“abortretryignore”**.

Пример задачи: реализовать вывод различных диалоговых окон в зависимости от выбора пользователя.

Решение:

```
from tkinter import *
import tkinter.messagebox as box
def dialog():
    var = box.askyesno("Выбор действий",
"Продолжаем ввод?")
    if var == 1:
        box.showinfo("Продолжение","Продолжаем...")
    else:
        box.showwarning("Прекращение", "Выход...")
window = Tk() #Создаем окно
window.title("Вывод сообщений") #Создаем заголовок окна
window.geometry("500x300") #Задаем размеры окна
#Создаем кнопку для выхода из программы
btn = Button(window, text = "Выбор решения", bg ="red", fg =
"#00ff00",width = 20, font = ("Arial", 16,"bold"), command = dialog)
btn.pack(padx = 100, pady = 100) #Размещаем кнопку в окне
```

```
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 8.

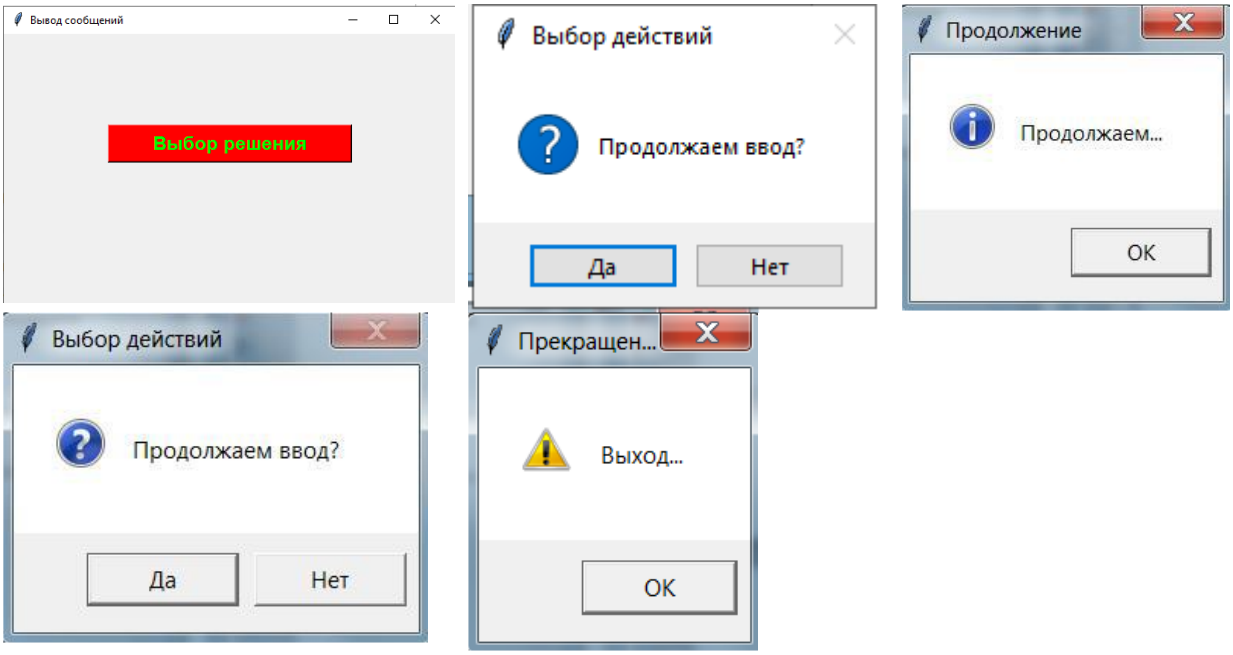


Рис. 8

1.7. ДИАЛОГОВЫЕ ОКНА ДЛЯ РАБОТЫ С ФАЙЛАМИ. МОДУЛЬ FILEDIALOG

Для работы с файлами через диалоговые окна необходимо использовать специальные методы модуля **filedialog**. Для получения имени открываемого файла предназначен метод **askopenfilename().**

Для получения имени сохраняемого файла предназначен метод **asksaveasfilename()**. Оба метода возвращают имя файла, который необходимо соответственно открыть или сохранить. При этом сами методы не открывают и не сохраняют файлы. Для того чтобы открыть или сохранить файл, необходимо использовать стандартные средства языка Python.

Пример задачи: реализовать возможность открытия и сохранения файлов с помощью диалоговых окон.

Решение:

```
from tkinter import *
from tkinter import filedialog as fd

def insertText():
    file_name = fd.askopenfilename()#Получаем имя файла
    f = open(file_name,'r',encoding='utf-8') #Открываем файл для
чтения
    s = f.read() # Считываем информацию из файла
    text.insert(1.0, s) #Вставляем считанную информацию в текстовое
поле
    f.close() #Закрываем файл
def extractText():
#Получаем имя файла, в который надо сохранить информацию
    file_name =fd.asksaveasfilename(
        filetypes=(("TXT files", "*.txt"),
                   ("HTML files", "*.html;*.htm"),
                   ("All files", "*.*")))
    f = open(file_name, 'w') #Открываем файл для записи
```

```
    s = text.get(1.0, END) #Считываем информацию из текстового поля
    f.write(s) #Записываем считанную информацию в файл
    f.close() #Закрываем файл
root = Tk() #Создаем окно
text = Text(width=50, height=25)
text.grid(columnspan=2)
b1 = Button(text="Открыть", command=insertText)
b1.grid(row=1,sticky=E)
b2 = Button(text="Сохранить", command=extractText)
b2.grid(row=1, column=1, sticky=W)
#Цикл обработки событий окна
root.mainloop()
```

В программе создаются две кнопки и одно многострочное поле. При нажатии кнопки **«Открыть»** открывается диалоговое окно для открытия файла. После выбора файла его содержимое отображается в текстовом поле. При нажатии кнопки **«Сохранить»** открывается диалоговое окно для сохранения информации в файл. После задания имени файла информация из текстового поля записывается в указанный файл.

Результат работы программы представлен на рисунках 9–12.

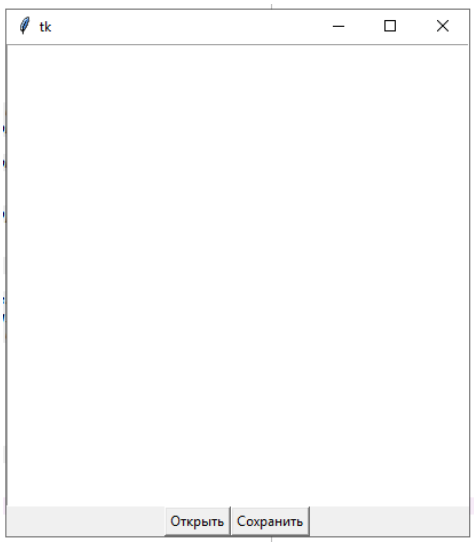


Рис.9

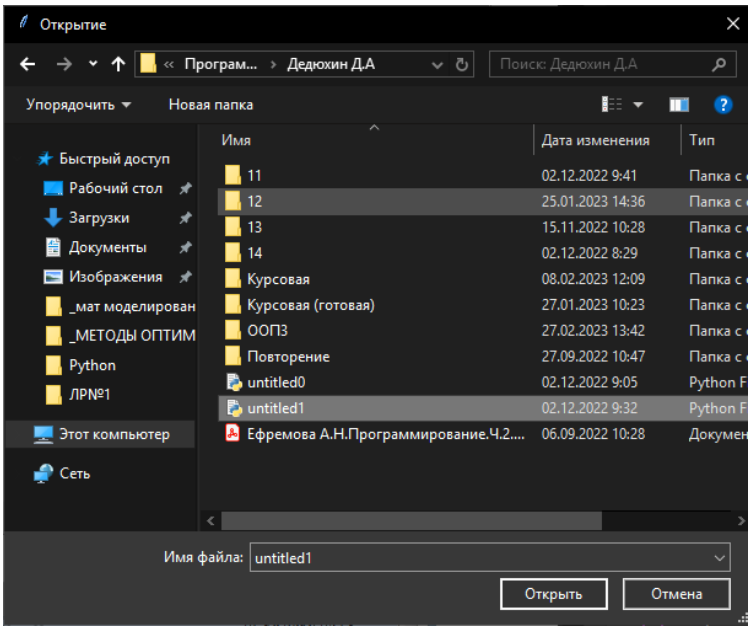


Рис.10

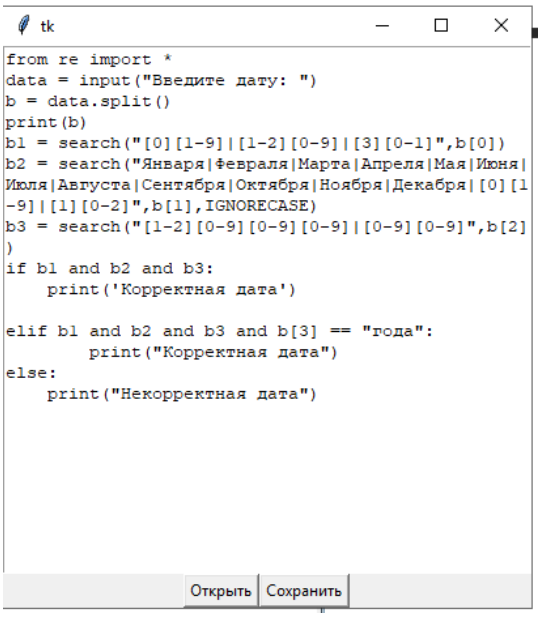


Рис.11

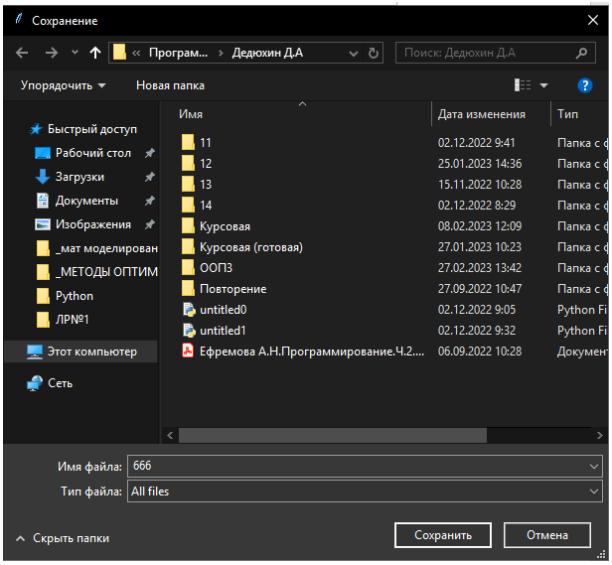


Рис.12

В качестве аргумента обоих методов можно задать переменную **filetype**, которая позволяет перечислить типы файлов, которые будут открываться или сохраняться, и их расширения.

Если диалоговые окна будут закрыты без выбора или указания имени файлов будет генерироваться соответствующее исключение.

1.8. ПРИЕМ ДАННЫХ ОТ ПОЛЬЗОВАТЕЛЯ. ВИДЖЕТ ENTRY

Для получения данных, введенных пользователем в приложении с графическим интерфейсом, в модуле **tkinter** существует виджет **Entry**, который представляет собой однострочное поле ввода. Объект ввода создается при помощи конструктора **Entry()** с передачей ему в качестве аргументов имени родительского контейнера (окна или фрейма), а также используемых аргументов, каждая из которых передается в виде пары **аргумент = значение**.

Как правило, виджет **Entry** для ввода текста располагают рядом с меткой **Label**, в которой описывается, что должен вводить пользователь, или рядом с кнопкой, которую пользователь может нажать, чтобы выполнить какие-то действия над введенными данными. Поэтому размещение виджетов в одном фрейме является, пожалуй, наиболее оптимальным вариантом размещения.

Свойства виджета **Entry** во многом схожи с двумя предыдущими виджетами. Наиболее часто используемые опции и их краткое описание представлены в таблице 3.

Таблица 3

Аргумент	Краткое описание
bd	Ширина рамки в пикселях (по умолчанию bd = 2)
bg	Цвет фона
fg	Цвет переднего плана (цвет текста)
font	Шрифт для текста
highlightcolor	Цвет рамки при наведении
selectbackground	Цвет фона выделенного текста
selectforeground	Цвет переднего плана выделенного текста
show	Использование маскирующих символов вместо видимых
state	Состояние: NORMAL — рабочее, DISABLED — отключено
width	Ширина поля ввода в символах

Пример задачи: реализовать использование виджета **Entry** для получения данных от пользователя.

Решение:

```
from tkinter import *
import tkinter.messagebox as box
window = Tk() #Создаем окно
window.title("Ввод данных") #Создаем заголовок окна
frame = Frame(window)#Создаем фрейм, в который будет размещено поле для ввода
entry = Entry(frame) #Создаем поле ввода и привязываем его к фрейму
#Функция для отображения данных, считанных из поля ввода
def dialog():
    box.showinfo("Приветствие", "Привет, " + entry.get())
#Создаем кнопку, которая будет вызывать функцию dialog()
btn = Button(frame, text = "Ввод", command = dialog)
#Создаем метку с поясняющим текстом
lb = Label(frame, text = "Введите имя: ")
#Добавляем все виджеты на фрейм
lb.pack(side = LEFT)
entry.pack(side = LEFT)
btn.pack(side = RIGHT, padx = 5)
frame.pack(padx = 20, pady = 20)
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 13.

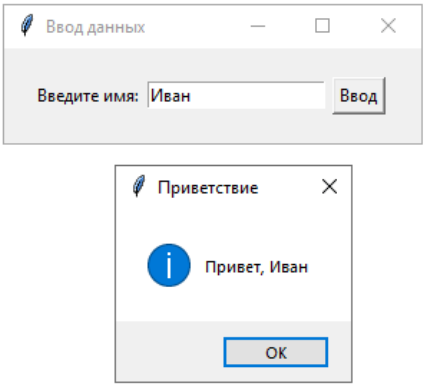


Рис.13

В отличие от свойств методы виджета **Entry** отличаются от методов двух предыдущих виджетов.

Основные методы виджета:

• **get()** — позволяет считать текст из текстового поля;

• **insert()** — позволяет вставить текст в текстовое поле; аргументы

• метода — позиция, в которую надо вставить текст (**0** — начало текстового поля, **END** — конец текстового поля, произвольное число —вставка будет производиться с указанной позиции), и сам текст;

• **delete()** — позволяет удалить текст из текстового поля; метод получает один или два аргумента: в первом случае удаляется один символ в указанной позиции, во втором случае удаляется срез между двумя указанными позициями, за исключением последней позиции; для очистки поля целиком для первого аргумента необходимо задать **0**, а для второго — **END**.

1.9. РАБОТА С МНОГОСТРОЧНЫМ ТЕКСТОМ. ВИДЖЕТ TEXT

Для ввода многострочного текста используется виджет **Text**, который создается с помощью соответствующего конструктора **Text()**.

Размеры виджета по умолчанию составляют 80x24 знакомест. С помощью аргументов **width** и **height** эти размеры можно изменять. По аналогии с предыдущими виджетами можно конфигурировать и другие свойства виджета (цвет фона, цвет текста, параметры шрифта и др.). Существует возможность реализации переноса слов на новую строку целиком, а не по буквам. Для этого необходимо задать аргументу **wrap** значение **WORD**.

Пример задачи: реализовать использование виджета **Text** для работы с многострочным текстом.

Решение:

```
from tkinter import *
window = Tk() #Создаем окно
window.title("Многострочный текст") #Создаем заголовок окна
#Создаем многострочное поле ввода
text = Text(width=30, height=10, bg = "blue", fg ="yellow", wrap = WORD)
text.pack()
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 14.

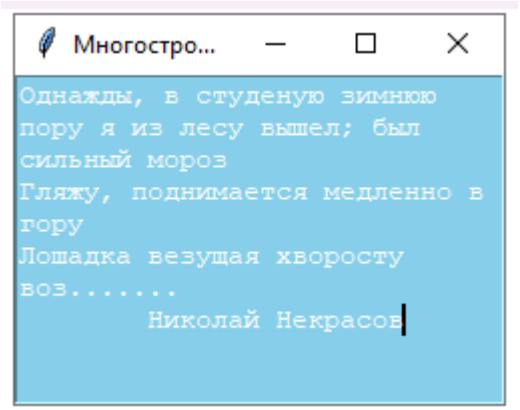


Рис.14

Основные методы виджета **Text** аналогичны методам виджета **Entry**: **get()**, **insert()** и **delete()**. Однако существуют и различия. Так, например, при вставке или удалении элемента в однострочном текстовом поле достаточно просто указать индекс удаляемого элемента.

Для выполнения этих операций в многострочном поле необходимо указывать два индекса: номер строки и номер столбца (номер символа в строке). При этом следует иметь в виду, что:

• строки нумеруются, начиная с единицы, а столбцы — начиная с нуля;

• методы **get()** и **delete()** не обязательно получают два аргумента, в случае необходимости получения доступа только к одному символу им можно передать только один аргумент с номером позиции.

Пример задачи: написать программу, которая позволяет производить стандартные операции с текстом в многострочном поле, а именно: добавлять в первую строку заданный текст, считывать с начала второй строки семь символов и выводить их в текстовом поле

Label, удалять весь текст из многострочного поля, начиная с восьмой позиции во второй строке.

Решение:

```
from tkinter import *
window = Tk() #Создаем окно
window.title("Многострочный текст") #Создаем заголовок окна
#Функция для вставки текста
def insertText():
    s = "Кафедра информатики,математики и физики"
    text.insert(1.0, s)
#Функция для считывания текста
def getText():
    s = text.get(2.0, 2.7)
    label['text'] = s
#Функция для удаления текста
def deleteText():
    text.delete(2.8, END)
#Создаем многострочное текстовое поле
text = Text(width=25, height=5)
text.pack()
#Создаем фрейм
frame = Frame(window)
frame.pack()

#Создаем три кнопки
b_insert = Button(frame, text="Вставить текст",command=insertText)
b_insert.pack(side=LEFT)
b_get = Button(frame, text="Считать текст", command=getText)
b_get.pack(side=LEFT)
b_delete = Button(frame, text="Удалить текст",command=deleteText)
b_delete.pack(side=LEFT)
#Создаем метку для вывода текста
label = Label()
label.pack()
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 15.

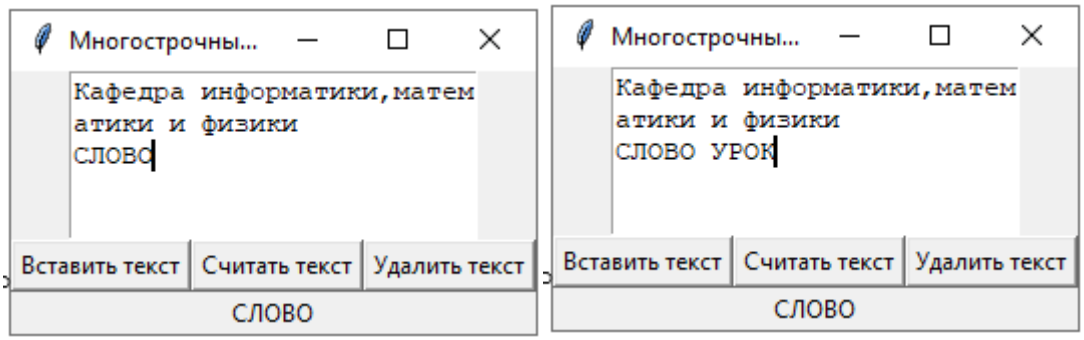


Рис.15

К особенности виджета **Text** можно отнести возможность различного форматирования разных частей текста в многострочном текстовом поле. Для этого можно использовать методы **tag_add()** и **tag_config()**.

Метод **tag_add()** используется для добавления тега. При этом необходимо задать имя тега и ту часть текста, для которой он предназначен. Метод **tag_config()** предназначен для задания стиля оформления для тега.

Пример задачи: реализовать форматирование текста в многострочном поле.

Решение:

```
from tkinter import *
window = Tk() #Создаем окно
window.title("Форматирование текста") #Создаем заголовок окна
#Создаем многострочное текстовое поле
text = Text(width=45, height=10)
text.pack()
#Вставляем текст
text.insert(1.0, "Первая строка\nВторая строка\nТретья строка")
#Форматируем первую строку
text.tag_add('first', 1.0, '1.end')
text.tag_config('first',    font    =    ("Calibri",    18,'bold'),
justify=CENTER)
#Форматируем вторую строку
text.tag_add('second', 2.0, '2.end')
text.tag_config('second',  font  =  ("Comic  Sans  Ms",  24,'bold'),
justify=RIGHT)
#Форматируем первое слово в третьей строке
text.tag_add('third', 3.0, '3.6')
text.tag_config('third', font = ("Courier New", 13,'bold'))
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 16.

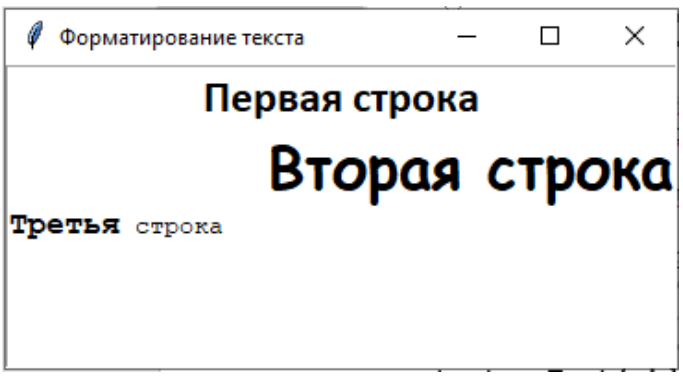


Рис.16

1.10. ИСПОЛЬЗОВАНИЕ ПОЛОСЫ ПРОКРУТКИ. ВИДЖЕТ SCROLLBAR

В том случае, если размер отображаемого текста превышает высоту виджета, то текст будет автоматически смещаться вниз. Для просмотра всего текста можно прокручивать его с помощью колесика мышки или клавиш со стрелками на клавиатуре. Однако, как правило, в этих случаях используют полосу прокрутки (скроллер).

Скроллеры представляют собой объекты класса **Scrollbar** и создаются с помощью одноименного конструктора, после чего связываются с нужным виджетом. Следует отметить, что скроллеры можно использовать не только с виджетами **Text**, но также и с другими виджетами.

Конфигурирование виджета **Scrollbar** подчиняется тем же правилам, что и конфигурирование других виджетов. Для задания варианта расположения виджета предназначен аргумент **orient**, который может принимать два значения: **HORIZONTAL** и **VERTICAL**. В качестве обработчика события необходимо указать виджет, в котором будет установлена полоса прокрутки и вариант ее расположения (**xview** — горизонтальная прокрутка, **yview** — вертикальная прокрутка). В свою очередь, виджет, в котором, устанавливается полоса прокрутки, также необходимо связать с ней с помощью одного из аргументов **xscrollcommand** или **yscrollcommand**.

Пример задачи: реализовать многострочное поле с вертикальной полосой прокрутки.

Решение:

```
from tkinter import *
window = Tk() #Создаем окно
window.title("Полоса прокрутки") #Создаем заголовок окна
#Создаем многострочное поле ввода
text = Text(width=30, height=5, bg = "skyblue", fg
="white",font=("Comic Sans MS",10,'bold'), wrap = WORD)
text.pack(side = LEFT)
#Создаем вертикальную полосу прокрутки
scroll = Scrollbar(orient = VERTICAL, command =text.yview)
scroll.pack(side = RIGHT, fill = Y)
#Конфигурируем поле ввода с полосой прокрутки
text.config(yscrollcommand = scroll.set)
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 17.

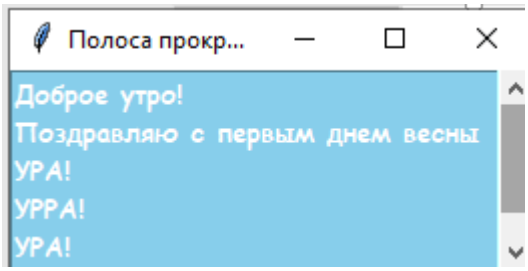


Рис.17

1.11. ВЫБОР ИЗ СПИСКА. ВИДЖЕТ LISTBOX

С помощью виджета **Listbox** можно добавлять в приложение список элементов, предлагаемых пользователю для выбора. Для создания списка используется конструктор **Listbox()**, которому в качестве аргументов необходимо передать имя родительского контейнера (например, окна или фрейма) и возможные опции, наиболее популярные из которых представлены в таблице 4.

Таблица 4

Опция	Краткое описание
bd	Ширина рамки в пикселях (по умолчанию bd =2)
bg	Цвет фона
fg	Цвет переднего плана (цвет текста)

font	Шрифт для текста
height	Количество строк в списке (по умолчанию height = 10)
selectbackground	Цвет фона выделенного текста
selectmode	Режим выбора: SINGLE — одиночный (по умолчанию),EXTENDED — множественный, позволяет выбирать любое количество элементов списка, используя клавиши Shift (сплошное выделение) и Ctrl (выборочное выделение), MULTIPLE — множественный, элементы выделяются щелчком левой кнопки мыши.
width	Ширина поля ввода списка в символах (по умолчанию width = 20)
yscrollcommand	Привязка к полосе прокрутки

Полоса прокрутки добавляется в виджет **Listbox** так же, как и в текстовое поле. В таблице 5 представлены основные методы объектов класса **Listbox**.

Таблица 5

Метод	Краткое описание
insert(n, str)	Добавление элемента str в список, где n — порядковый номер элемента в списке (0 — вставка в начало списка, END —вставка в конец списка), str — элемент списка в виде строки
get(n)	Получение элемента списка, где n — порядковый номер элемента в списке. Для получения нескольких элементов списка необходимо указать срез их индексов (например, вызов get(0, END) позволяет получить список всех элементов
curselection()	Возвращает порядковый номер (индекс) выбранного элемента списка. Если выделено несколько элементов, то метод возвращает кортеж индексов выбранных элементов
delete(n)	Удаляет элемент с порядковым номером n из списка. Для удаления нескольких элементов списка необходимо указать соответствующий срез индексов

Пример задачи: реализовать возможность выбора языка программирования из поля со списком. При нажатии кнопки на экране должно появляться диалоговое окно с названием выбранного языка.

Решение:

```
from tkinter import *
import tkinter.messagebox as box
#Функция по выводу диалогового окна
def dialog():
    box.showinfo("Выбор языка", "Вы выбрали язык "+
listbox.get(listbox.curselection()))
window = Tk() #Создаем окно
window.title("Работа со списком") #Создаем заголовок окна
#Создаем фрейм для виджетов
frame = Frame(window)
#Создаем виджет Listbox
listbox = Listbox(frame)
#Заполняем список
for i in('JavaScript','Python', 'Java','C/C++','PHP','C#','Visual
development tools','Kotlin', 'Swift','Go','Dart','Object C'):
        listbox.insert(END,i)
listbox.insert(3,"Julia ")
#Создаем кнопку
btn = Button(frame, text = "Выберите язык", bg ="lightgreen", command
= dialog)
#Размещаем виджеты
btn.pack(side = RIGHT, padx = 15)
listbox.pack(side = LEFT)
frame.pack(padx=30, pady=30)
#Цикл обработки событий окна
```

```
window.mainloop()
```

Результат работы программы представлен на рисунке 18.

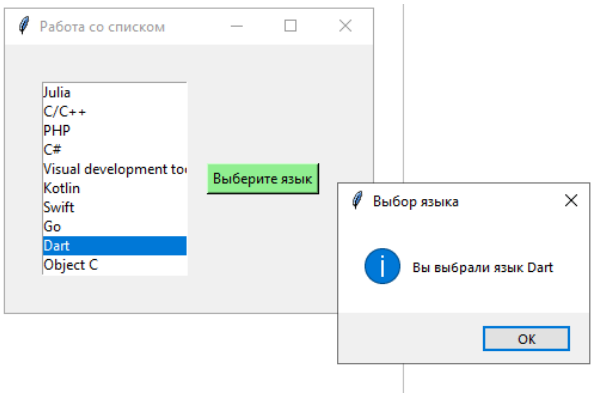


Рис.18

1.12. ИСПОЛЬЗОВАНИЕ ПЕРЕКЛЮЧАТЕЛЕЙ. ВИДЖЕТ RADIOBUTTON

Виджет **Radiobutton** работает по принципу переключателя и используется в программах с графическим интерфейсом для реализации выбора «один из многих». Реализовать множественный выбор с помощью виджета **Radiobutton** нельзя.

Все радиокнопки переключателя группируются вместе и связываются между собой через общую переменную. Разным положениям переключателя соответствуют разные значения этой переменной.

Сами переменные создаются с помощью специальных классовпеременных:

- **BooleanVar** — позволяет принимать своим экземплярам только значения булева типа (**0** или **1**);
- **IntVar** — позволяет принимать своим экземплярам только значения целого типа;
- **DoubleVar** — позволяет принимать своим экземплярам только значения вещественного типа;
- **StringVar** — позволяет принимать своим экземплярам только значения строкового типа.

Для создания переменных используются соответствующие конструкторы. Например, конструктор **DoubleVar()** инициализирует пустую переменную-объект вещественного типа, а конструктор **StringVar()** создает пустую строку.

Для создания самого объекта положения переключателя используется конструктор **Radiobutton()**. В качестве аргументов этому конструктору необходимо передать:

- имя родительского контейнера (например, окна или фрейма);
- текстовую строку в виде пары **text = 'текст'**, которая будет являться меткой;
- управляющую переменную-объект в виде пары **variable = имя переменной**;
- значение для присваивания в виде пары **value = значение**.

Каждый объект положения переключателя содержит метод **select()**. С помощью этого метода можно определить положение, в которое будет установлен переключатель при запуске программы.

Второй способ установки переключателя в определенное положение заключается в использовании метода **set()**.

Строковое значение, которое присваивается переменной в результате выбора переключателя, может быть получено из переменной-объекта с помощью метода **get()**.

Для снятия положения переключателя используется метод **deselect()**.

Пример задачи: реализовать возможность выбора языка программирования с помощью переключателей. При нажатии кнопки на экране должно появляться диалоговое окно с названием выбранного языка.

Решение:

```
from tkinter import *
import tkinter.messagebox as box
#Функция по выводу диалогового окна
def dialog():
```

```
        box.showinfo("Выбор языка", "Вы выбрали язык "+ book.get())
    window = Tk() #Создаем окно
    window.title("Работа с переключателем") #Создаем заголовок окна
    #Создаем фрейм для виджетов
    frame = Frame(window)
    #Создаем строковую переменную-объект для хранения результата
выбора
    book = StringVar()
    #Создаем виджеты положения переключателя
    r_1 = Radiobutton(frame, text = 'JavaScript', variable =book,
value = 'JavaScript')
    r_2 = Radiobutton(frame, text = 'Python', variable =book, value =
'Python')
    r_3 = Radiobutton(frame, text = 'Java', variable= book, value =
'Java')
    r_4 = Radiobutton(frame, text = 'C/C++', variable= book, value =
'C/C++')
    r_5 = Radiobutton(frame, text = 'PHP', variable =book, value =
'PHP')
    r_6 = Radiobutton(frame, text = 'C#', variable= book, value =
'C#')
    r_7 = Radiobutton(frame, text = 'Julia', variable =book, value =
'Julia')
    r_8 = Radiobutton(frame, text = 'Kotlin', variable =book, value =
'Kotlin')
    r_9 = Radiobutton(frame, text = 'Swift', variable =book, value =
'Swift')
    r_10 = Radiobutton(frame, text = 'Go', variable =book, value =
'Go')
    r_11 = Radiobutton(frame, text = 'Dart', variable =book, value =
'Dart')
    r_12 = Radiobutton(frame, text = 'Object C', variable =book, value
= 'Object C')

    #Устанавливаем переключатель при запуске в первое положение
    r_1.select()
    #book.set('Java') Можно и так установить переключатель в первое
положение
    #Создаем кнопку
    btn = Button(frame, text = "Выберите язык", bg ="lightgreen",
command = dialog)
    #Размещаем виджеты
    r_1.pack(padx = 10)
    r_2.pack(padx = 10)
    r_3.pack(padx = 10)
    r_4.pack(padx = 10)
    r_5.pack(padx = 10)
    r_6.pack(padx = 10)
    r_7.pack(padx = 10)
    r_8.pack(padx = 10)
    r_9.pack(padx = 10)
    r_10.pack(padx = 10)
    r_11.pack(padx = 10)
    r_12.pack(padx = 10)

    btn.pack()
    frame.pack(padx = 120,pady = 30 )
    #Цикл обработки событий окна
```

```
window.mainloop()
```

Результат работы программы представлен на рисунке 19.

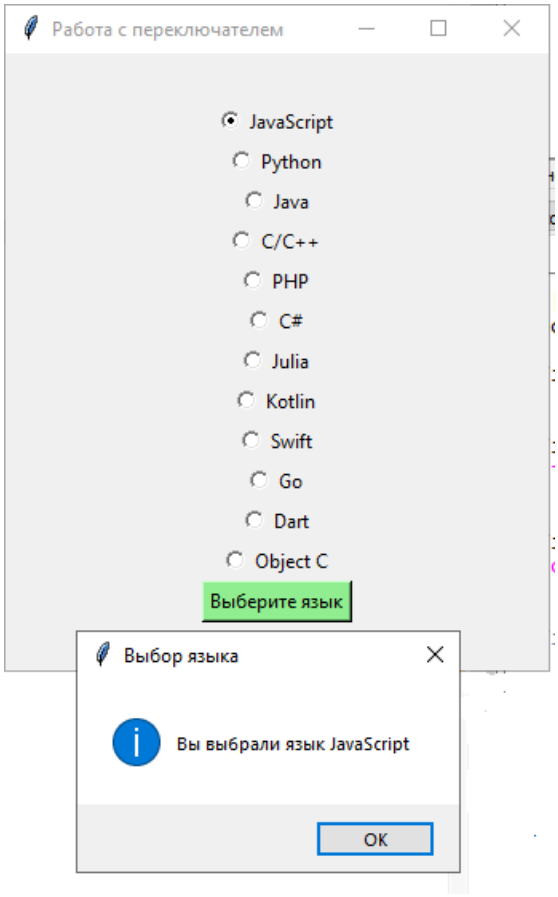


Рис.19

1.13. РАБОТА С ФЛАЖКАМИ. ВИДЖЕТ CHECKBUTTON

Виджет **Checkbutton** (флажок) используется в программах с графическим интерфейсом для реализации как выбора «один из многих», так и множественного выбора «один из многих».

Как и в случае с переключателями, в графических интерфейсах обычно используется группа таких виджетов. Однако в отличие от переключателей пользователь может выбрать как один, так и несколько флажков одновременно.

Как и в случае с переключателями, каждому флажку необходимо поставить в соответствие переменную-объект, однако устанавливать связи между флажками не нужно. Переменные-объекты используются для получения сведений о состоянии флажков. По значению связанной с виджетом переменной можно установить, активирован ли данный флажок, после чего определиться с ходом выполнения программы.

Связанные с флажками переменные могут быть только тех же четырех типов, что и переменные, использующиеся при работе с переключателями (т. е. **BooleanVar**, **IntVar**, **DoubleVar** и **StringVar**).

Для создания объекта флажка используется конструктор **Checkbutton()**. В качестве аргументов этому конструктору необходимо передать пять аргументов:

• имя родительского контейнера, например окна или фрейма;

• текстовую строку в виде пары **text = 'текст'**, которая будет являться меткой;

• управляющую переменную-объект в виде пары **variable = имя переменной**;

• значение для присваивания в виде пары **onvalue = значение** для случая, когда флажок активирован;

• значение для присваивания в виде пары **offvalue = значение** для случая, когда флажок не активирован (сброшен).

Программно включать флажки можно (по аналогии с переключателями) с помощью методов **select()** и **set()**. Для программного выключения флажка используется метод **deselect()**.

Значение, которое присвоено в результате установки флажка, может быть получено из переменной-объекта с помощью метода **get()**.

Состояние флажка можно поменять на противоположное, используя метод **toggle()**.

Пример задачи: реализовать предыдущую задачу с помощью флажков.

Решение:

```
from tkinter import *
import tkinter.messagebox as box
#Функция по выводу диалогового окна
def dialog():
    s = "Вы выбрали: "
    if var_1.get() == 1: s += "\nJava"
    if var_2.get() == 1: s += "\nC++"
    if var_3.get() == 1: s += "\nPython"
    if var_4.get() == 1: s += "\nJulia"
    if var_5.get() == 1: s += "\nC#"
    if var_6.get() == 1: s += "\nJavaScript"
    if var_7.get() == 1: s += "\nPHP"
    box.showinfo("Выбор языка", s)
window = Tk() #Создаем окно
window.title("Работа с флажками") #Создаем заголовок окна
#Создаем фрейм для виджетов
frame = Frame(window)
#Создаем целочисленные переменные-объекты для хранения результата
выбора
var_1 = IntVar()
var_2 = IntVar()
var_3 = IntVar()
var_4 = IntVar()
var_5 = IntVar()
var_6 = IntVar()
var_7 = IntVar()
#Создаем флажки

check_1 = Checkbutton(frame, text = 'Java', variable= var_1,onvalue
= 1, offvalue = 0)
check_2 = Checkbutton(frame, text = 'C++', variable= var_2,onvalue
= 1, offvalue = 0)
check_3 = Checkbutton(frame, text = 'Python', variable=
var_3,onvalue = 1, offvalue = 0)
check_4 = Checkbutton(frame, text = 'Julia',variable = var_4,onvalue
= 1, offvalue = 0)
check_5 = Checkbutton(frame, text = 'C#', variable= var_5,onvalue =
1, offvalue = 0)
check_6 = Checkbutton(frame, text = 'JavaScript',variable =
var_6,onvalue = 1, offvalue = 0)
check_7 = Checkbutton(frame, text = 'PHP', variable= var_7,onvalue
= 1, offvalue = 0)
#Создаем кнопку
btn = Button(frame, text = "Выбор языка", bg ="lightgreen", command
= dialog)
#Размещаем виджеты
btn.pack(side = RIGHT)
check_1.pack(side = LEFT)
check_2.pack(side = LEFT)
```

```
check_3.pack(side = LEFT)
check_4.pack(side = LEFT)
check_5.pack(side = LEFT)
check_6.pack(side = LEFT)
check_7.pack(side = LEFT)
frame.pack(padx = 20,pady = 30 )
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 20.

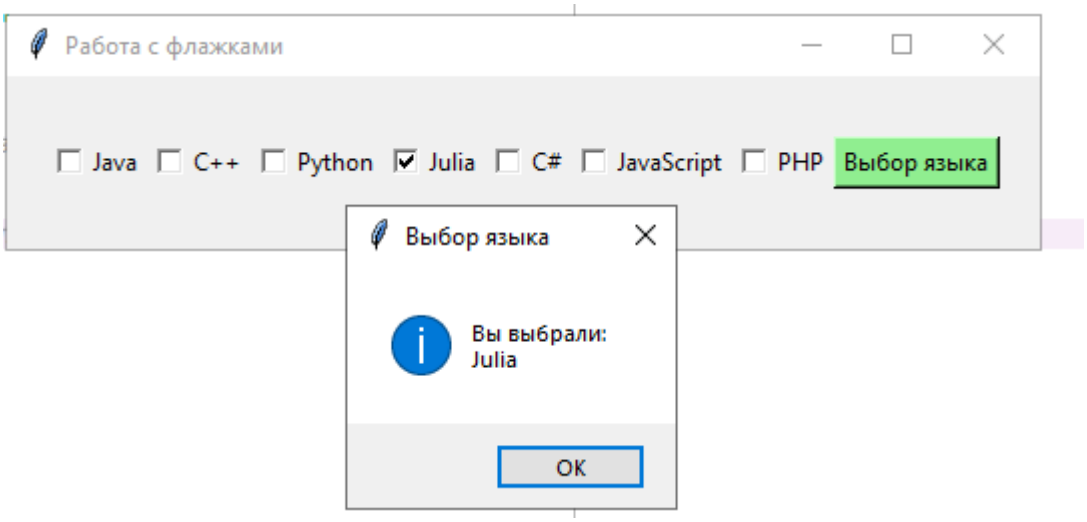


Рис.20

1.14. СОЗДАНИЕ МЕНЮ. ВИДЖЕТ MENU

Для управления работой приложения с графическим пользовательским интерфейсом, как правило, используется меню. При выборе того или иного пункта меню выполняется предназначенная для него команда. В результате выполнения команд меню, как правило, выполняется какое-то действие или открывается какое-то диалоговое окно.

В библиотеке **Tk** меню представляет собой объект класса **Menu**. Этот объект должен быть связан с тем виджетом, в котором это меню будет располагаться. Обычно в роли такого виджета выступает главное окно приложения. Для связывания окна и меню аргументу **menu** конструктора окна необходимо присвоить экземпляр **Menu** через имя связанной с экземпляром переменной. Для создания меню как объекта используется конструктор **Menu()**.

Для добавления пунктов меню используется метод **add_command()**. В качестве аргумента метода выступает метка с названием пункта меню.

Пример задачи: реализовать простейшее меню из двух пунктов «Файл» и «Справка».

Решение:

```
from tkinter import *
window = Tk() #Создаем окно
window.title("Работа с меню") #Создаем заголовок окна
#Создаем меню в главном окне
mainmenu = Menu(window)
#Добавляем пункты меню
mainmenu.add_command(label = "Файл")
mainmenu.add_command(label = "Справка")
#Конфигурируем окно с меню
window.config(menu = mainmenu)
#Цикл обработки событий окна
window.mainloop()
```

Результат работы программы представлен на рисунке 21.

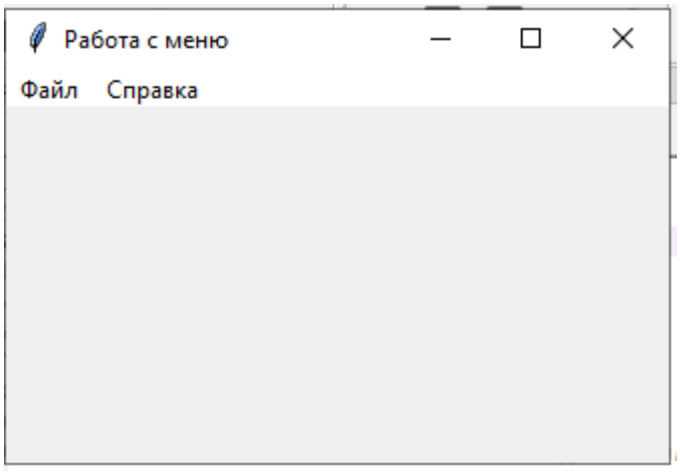


Рис.21

Пункты меню «Файл» и «Справка» — это команды. Для связи с нужной функцией-обработчиком в эти команды можно добавить аргумент **command**. Однако в большинстве приложений, как правило, команды добавляют не к пунктам основного меню, а к пунктам дочерних меню, которые, в свою очередь, реализуются в виде раскрывающихся списков при выборе пункта главного меню. В результате выбор пункта главного меню не запускает никакие процедуры обработки, а приводит лишь к раскрытию соответствующего списка с командами, которые уже могут быть запущены по щелчку мыши.

Для реализации такого подхода создаются новые экземпляры класса **Menu**, которые связываются с главным меню с помощью метода **add_cascade()**.

Пример задачи: добавить для каждого пункта предыдущего меню раскрывающиеся списки.

Решение:

```
from tkinter import *
window = Tk()
window.title("Меню с выпадающим списком")
#Создаем меню в главном окне
mainmenu = Menu(window)
window.config(menu=mainmenu)
#Создаем пункты подменю для пункта меню "Файл"
filemenu = Menu(mainmenu, tearoff=0) #Создаем еще один объект Menu
filemenu.add_command(label="Открыть...",font=("Comic Sans
MS",10,'bold')) #Добавляем в него пункты меню
filemenu.add_command(label="Новый",font=("Comic Sans MS",10,'bold'))
filemenu.add_command(label="Сохранить...",font=("Comic Sans
MS",10,'bold'))
filemenu.add_command(label="Выход",font=("Comic Sans MS",10,'bold'))
#Создаем пункты подменю для пункта меню "Справка"
helpmenu = Menu(mainmenu, tearoff=0) #Создаем еще один объект Menu
helpmenu.add_command(label="Помощь",font=("Comic Sans
MS",10,'bold'))#Добавляем в него пункты меню
helpmenu.add_command(label="О программе",font=("Comic Sans
MS",10,'bold'))
#Связываем два созданных меню с главным меню
mainmenu.add_cascade(label="Файл", menu=filemenu)
mainmenu.add_cascade(label="Справка", menu=helpmenu)
window.mainloop()
```

Результат работы программы представлен на рисунке 22.

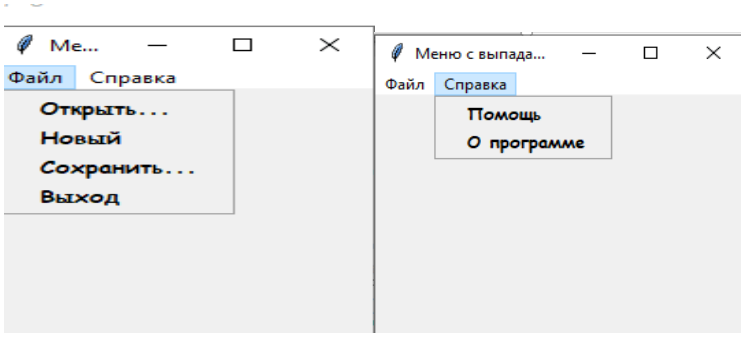


Рис.22

Из текста программы видно, что команды добавляются не к основному меню, а к дочерним меню (**filemenu** и **helpmenu**). В качестве родительского окна для дочерних меню указывается не главное окно приложения (**window**), а основное меню (**mainmenu**).

Для управления возможностью откреплення дочернего меню в конструкторе класса **Menu** можно использовать аргумент **tearoff**. При значении аргумента **tearoff** = **0** открепить дочернее меню будет невозможно. В противном случае с помощью щелчка мыши по специальной линии можно было бы реализовать плавающее меню (когда **tearoff** = **0**, эта линия отсутствует в окне приложения).

Используя описанный выше подход можно реализовывать в программах многоуровневые меню.

Пример задачи: реализовать многоуровневое меню для предыдущей задачи путем добавления в пункт меню «Справка» дополнительного уровня.

Решение:

```
from tkinter import *
window = Tk()
window.title("Многоуровневое меню")
#Создаем меню в главном окне
mainmenu = Menu(window)
window.config(menu=mainmenu)
#Создаем пункты подменю для пункта меню "Файл"
filemenu = Menu(mainmenu, tearoff=0) #Создаем еще один объект Menu
filemenu.add_command(label="Открыть...") #Добавляем в него пункты меню
filemenu.add_separator()#Добавляем разделитель
filemenu.add_command(label="Новый")
filemenu.add_separator()#Добавляем разделитель
filemenu.add_command(label="Сохранить...")
filemenu.add_separator()#Добавляем разделитель
filemenu.add_command(label="Выход")
#Создаем пункт подменю "Помощь" для пункта меню "Справка"
helpmenu = Menu(mainmenu, tearoff=0) #Создаем еще один объект Menu
#Добавляем еще один уровень меню к пункту подменю "Помощь"
helpmenu1 = Menu(helpmenu, tearoff=0)
helpmenu1.add_command(label="Локальная справка")
helpmenu1.add_separator()#Добавляем линию- разделитель
helpmenu1.add_command(label="На сайте")
#Связываем два созданных пункта меню с пунктом подменю "Помощь"
helpmenu.add_cascade(label="Помощь", menu=helpmenu1)
helpmenu.add_separator()#Добавляем линию- разделитель
#Создаем пункт подменю "О программе" для пункта меню "Справка"
helpmenu.add_command(label="О программе") #Связываем два созданных
меню с главным меню
mainmenu.add_cascade(label="Файл", menu=filemenu)
mainmenu.add_cascade(label="Справка", menu=helpmenu)
window.mainloop()
```

Результат работы программы представлен на рисунке 23.

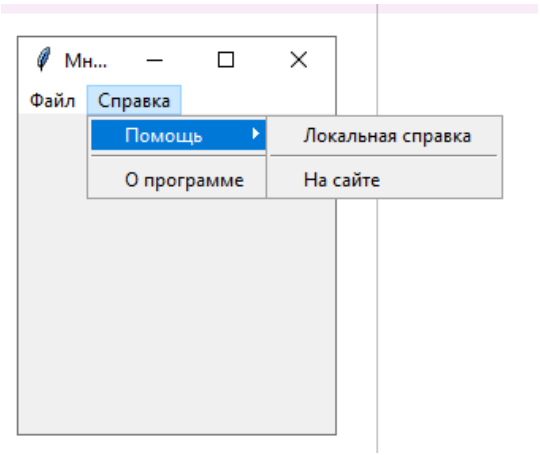
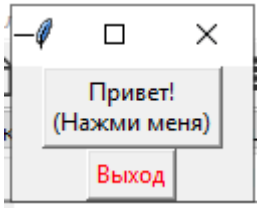


Рис.23

Замечание. С помощью метода **add_separator()** в пункты меню можно добавлять линию-разделитель. Обычно такой прием используют в тех случаях, когда необходимо разделить группы команд.

Объектно-ориентированный подход

```
import tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.pack()
        self.create_widgets()

    def create_widgets(self):
        self.hi_there = tk.Button(self)
        self.hi_there["text"] = "Привет!\n(Нажми меня)"
        self.hi_there["command"] = self.say_hi
        self.hi_there.pack(side="top")

        self.quit = tk.Button(self, text="Выход", fg="red",
                              command=self.master.destroy)
        self.quit.pack(side="bottom")

    def say_hi(self):
        print("Привет, хорошего дня!")

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```



ПРАКТИЧЕСКИЕ ЗАДАНИЯ

1. Составьте программу, которая будет представлять окно с расположенными в нем семью кнопками разного цвета, соответствующими семи цветам радуги. При нажатии на каждую из кнопок на экран должно выводиться название цвета (например, фиолетовый) и его шестнадцатеричный код (#7d00ff). Цвет надписи должен соответствовать цвету выбранной кнопки.

Возможный результат работы программы представлен на рисунке:

2. Составьте объектно-ориентированную программу, в которой объектами будут выступать блоки, состоящие из поля для ввода тек- ста (виджет **Entry**), кнопки (виджет **Button**) и метки для вывода тек- ста (виджет **Label**).

Комментарии к программе:

1) создайте в программе класс, например, с именем **Group_Widgets**, атрибутами которого будут три разных виджета (**Entry**, **Button** и **Label**);

2) включите в состав методов класса следующие методы:

− конструктор;

− метод, который будет считывать введенные данные из поля данных (виджет **Entry**), сортировать их по возрастанию и выводить результат в виджете **Label**;

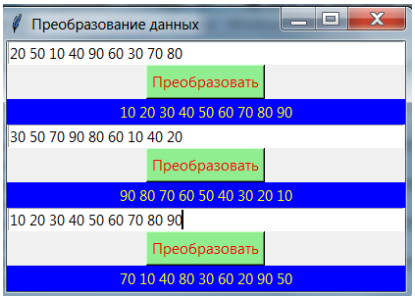
− метод, который будет считывать введенные данные из поля данных (виджет **Entry**), сортировать их по убыванию и выводить результат в виджете **Label**;

− метод, который будет считывать введенные данные из поля данных (виджет **Entry**), перемешивать их случайным образом и выводить результат в виджете **Label**;

− метод, который будет связывать кнопку с одним из вышеперечисленных методов;

3)в основной программе создайте три объекта класса **Group_Widgets**, каждый из которых будет преобразовывать введенные пользователем данные одним из трех вышеперечисленных способов (сортировать по возрастанию, сортировать по убыванию, перемешивать случайным образом) и протестируйте работу программы.

Возможный результат работы программы представлен на рисунке:



3. Составьте программу с пользовательским графическим интерфейсом, которая будет позволять:

• добавлять в список элемент, введенный пользователем в текстовое поле;

• удалять из списка выбранные пользователем элементы;

• сохранять выбранные пользователем элементы в файл, при этом каждый сохраняемый элемент должен записываться с новой строки.

Комментарии к программе.

1) создайте в программе один список (**Listbox**), одно поле ввода (**Entry**) и три кнопки (**Button**), например, с именами «Добавить», «Удалить» и «Сохранить»;

2) добавьте в список вертикальную полосу прокрутки (**Scrollbar**);

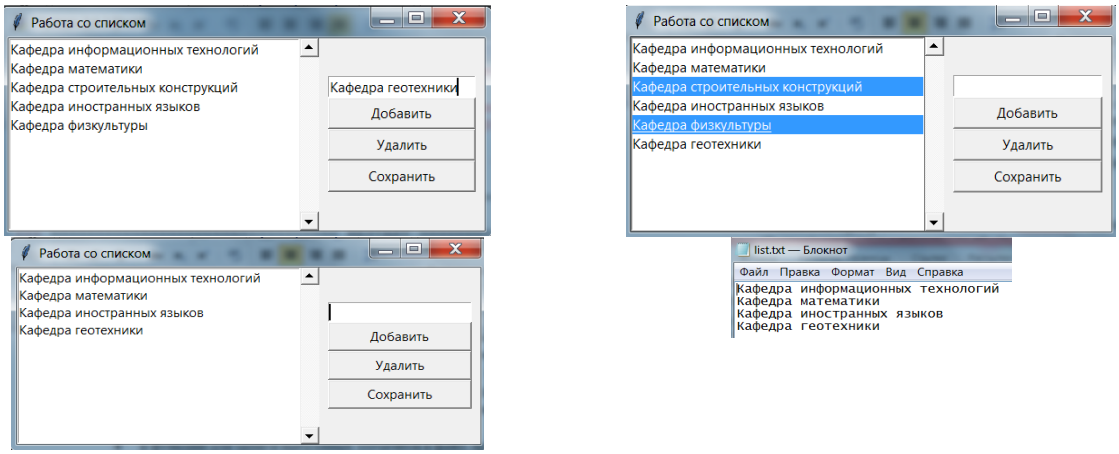
3) для каждой из кнопок напишите обработчики событий;

4) при нажатии кнопки «Добавление» введенная пользователем строка должна добавляться **в конец списка** и исчезать из поля ввода;

5) в функции по удалению выбранных элементов вначале преобразуйте кортеж выбранных элементов в список, а затем с помощью метода **revers()** измените порядок следования элементов в нем на противоположный для того, чтобы удаление элементов списка происходило с конца списка (в противном случае удаление элемента будет приводить к изменению индексов всех следующих за ним элементов — в результате программа будет работать неправильно);

6) в функции для записи выбранных элементов в файл, преобразуйте кортеж строк-элементов списка, который вернет метод **get()**, в одну строку с помощью метода **join()** через разделитель “\n”.

Возможный результат работы программы представлен на рисунке:



4. Составьте программу, имитирующую интерфейс добавления товаров в корзину.

Комментарии к программе:

1) создайте в программе два списка (один с полосой прокрутки, другой без нее) и три кнопки;

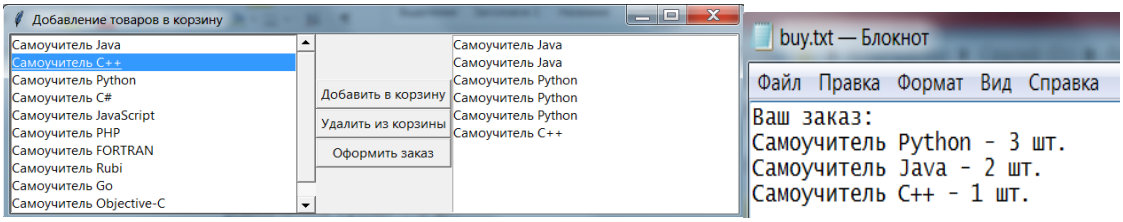
2) в первом списке отобразите список товаров, заданный программно, а второй список должен быть изначально пустым;

3) при нажатии кнопки «Добавить в корзину» товар должен перемещаться из одного списка в другой, при этом каждый новый товар должен добавляться в конец списка;

4) при нажатии на кнопку «Удалить из корзины» товар должен удаляться из второго списка;

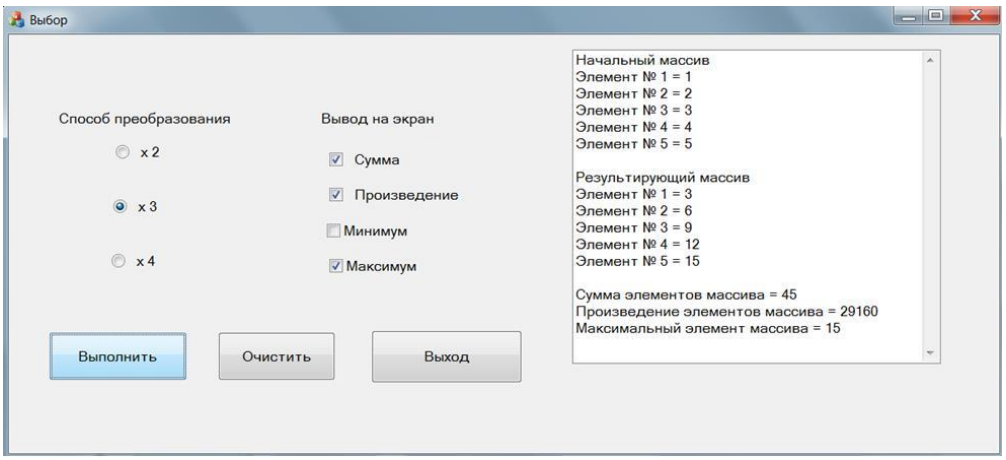
5) при нажатии кнопки «Оформить заказ» информация о выбранных товарах и их количестве должна записываться в файл.

Возможный результат работы программы представлен на рисунке:



5. Составьте программу, которая будет преобразовывать заданный целочисленный массив из пяти элементов разными способами **в зависимости от выбранного переключателя**, а именно, увеличивать каждый элемент исходного массива в 2, 3 или 4 раза. **В зависимости от выбранного флажка** на экран должны выводиться различные итоговые результаты, а именно, сумма, произведение всех эле-ентов нового массива, минимальный и максимальный его элемент.

Возможный результат работы программы представлен на рисунке:



6. Составьте программу с графическим пользовательским интерфейсом для работы с внешними файлами.

Комментарии к программе:

1) в программе создайте две кнопки и одно многотекстовое поле;

2) при нажатии кнопки «Открыть» на экране должно появляться диалоговое окно для открытия файла;

3) после выбора файла его содержимое должно отображаться в текстовом поле;

4) при нажатии кнопки «Сохранить» должно открываться диалоговое окно для сохранения информации в файл; после задания имени файла информация из текстового поля должна записываться в указанный файл;

5) добавьте в программу код обработки исключений, которые могут генерироваться в случае, если диалоговые окна были закрыты без выбора или указания имени файла; при этом в случае генерации исключения на экран должно выводиться диалоговое окно с соответствующим сообщением о том, что файл не загружен или не сохранен;

6) добавьте в интерфейс кнопку «Очистить», при нажатии на которую будет удаляться вся информация из текстового поля; при этом перед удалением пользователь должен подтвердить свои намерения через соответствующее диалоговое окно.

7. Реализуйте еще одну версию предыдущей программы для случая выполнения всей функциональности с помощью меню. Команду очистки текстового поля поместите в контекстное меню