ОГЛАВЛЕНИЕ

ГРАФИЧЕСКИЙ ИНТЕРФЕЙС

Современный пользователь в основном использует приложения с графическим пользовательским интерфейсом (GUI), взаимодействуя с программой с помощью различных кнопок, меню, значков, вводя информацию в специальные поля, выбирая определенные значения в списках и т. д. Для внедрения этих и многих других графических компонент в приложение используют графические библиотеки. Для языка Python наиболее известными являются Tkinter, PyQt, wxPython, PyGTK. Рассмотрим Tkinter, как небольшой и простой модуль, который входит в состав стандартных библиотек языка.

Для создания интерфейса, первым делом, необходимо импортировать Tkinter и создать графическое окно. Для этого создаем экземпляр класса Tk с именем window. Далее последовательно вызываем методы класса:

– title: устанавливает заголовок окна;

– geometry: устанавливает размеры окна. В качестве аргумента может передаваться строка в формате «100x200» или «100x200+300+400», где 100 – ширина окна, 200 – высота окна, 300 – отступ от правого края экрана, 400 – отступ от верхнего края экрана.

– mainloop: запускает цикл обработки событий окна для взаимодействия с пользователем. Необходимо вызвать в конце для отображения окна.

```
1.  from tkinter import *
2.  window = Tk()
3.  window.title("Название приложения")
4.  window.geometry("300x200")
5.
6.  # В этой части добавляются элементы
7.
8.  window.mainloop()
```

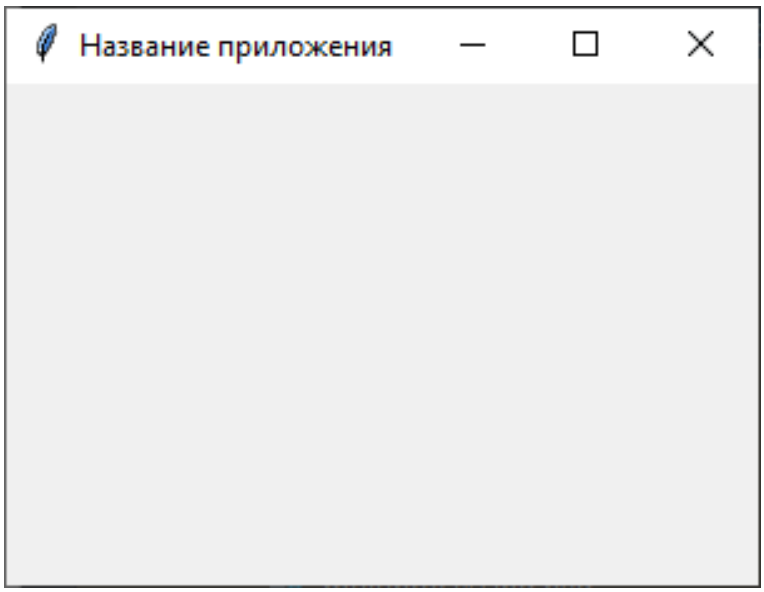

Рисунок 1 – Пустое графическое приложение

После создания графического окна можно перейти к его наполнению необходимыми элементами. Разместим на форме кнопку, для этого создадим экземпляр класса Button.

```
1.   btn = Button(window, text="Button")
```

Этот пример создает кнопку с соответствующей надписью, которая будет принадлежать нашему окну. Однако, при выполнении программы ничего не возникнет на экране, так как не было указано, где разместить элемент.

Размещение элементов

Существует три способа размещения элементов: методы grid, pack и place. *Метод grid* позволяет разместить элемент в некоторую ячейку условной, невидимой сетки. Основными параметрами данного метода являются row и column – номер строки и столбца в данной условной сетке.

```
1.   label1 = Label(window, text="Виджет Label #1")
2.   label1.grid(column=0, row=0)
3.   label1 = Label(window, text="Виджет Label #2")
4.   label1.grid(column=0, row=1)
5.   label1 = Label(window, text="Виджет Label #2")
6.   label1.grid(column=1, row=1)
```

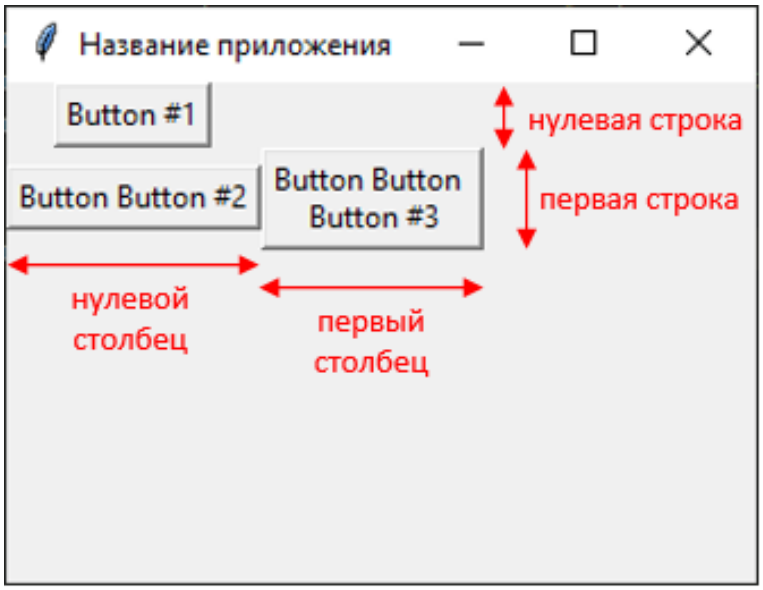

Рисунок 2 – Размещение элементов с использованием grid

Дополнительно могут указываться следующие параметры: rowspan и columnspan (количество строк и столбцов, занимаемых элементом), ipadx и ipady (отступы по горизонтали и вертикали от границ элемента до его текста), padx и pady (отступы от границ ячейки сетки до границ элемента).

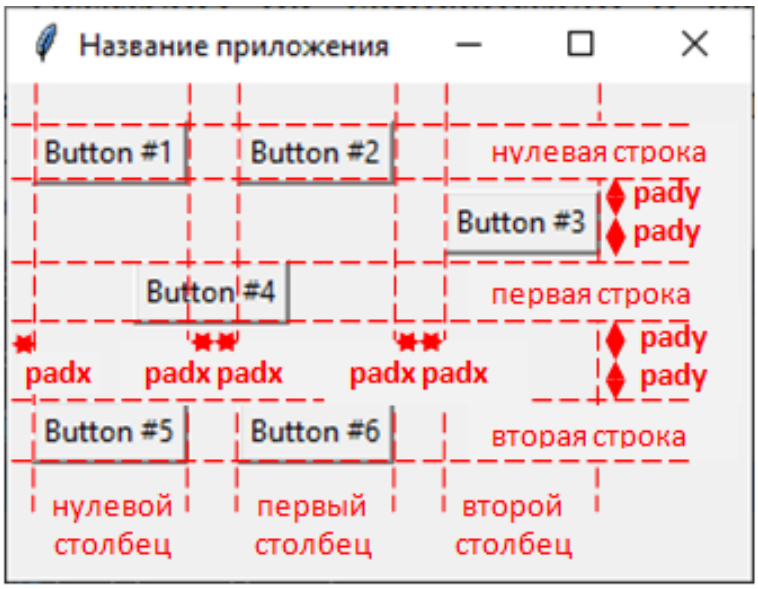

Рисунок 3 – Дополнительная настройка метода grid

```
1.   btn1 = Button(window, text="Button #1")
2.   btn1.grid(column=0, row=0, padx=10, pady=15)
3.   btn2 = Button(window, text="Button #2")
4.   btn2.grid(column=1, row=0, padx=10, pady=15)
```

```
5.   btn3 = Button(window, text="Button #3")
6.   btn3.grid(column=2, row=0, rowspan=2, padx=5, pady=15)
7.   btn4 = Button(window, text="Button #4")
8.   btn4.grid(column=0, row=1, columnspan=2, padx=5,
pady=15)
9.   btn5 = Button(window, text="Button #5")
10.  btn5.grid(column=0, row=2, padx=10, pady=15)
11.  btn6 = Button(window, text="Button #6")
12.  btn6.grid(column=1, row=2, padx=10, pady=15)
```

Метод pack позиционирует элемент внутри родительского контейнера по одной из сторон. Это наиболее простой сполсоб размещения. Этот метод принимает три стандартных аргумента:

– expand: логичекая переменная, определяющая заполнит ли элемент все пространство контейнера;

– fill: принимает одно из значений (NONE, X, Y, BOTH) в зависимости от того по какой оси будет растягиваться элемент;

– side: принимает одно из значений (TOP, LEFT, RIGHT, BOTTOM) в зависимости от того, по какой стороне контейнера будет выровнен элемент.

Также дополнительно могут указываться парметры ipadx, ipady, padx и pady, определяющие то же, что и в предыдущем методе.

```
1.   btn1 = Button(window, text="Button #1")
2.   btn1.pack(side=TOP, padx=10, pady=15)
3.   btn2 = Button(window, text="Button #2")
4.   btn2.pack(side=BOTTOM, padx=10, pady=15)
5.   btn3 = Button(window, text="Button #3")
6.   btn3.pack(side=RIGHT, padx=10, pady=15)
7.   btn4 = Button(window, text="Button #4")
8.   btn4.pack(side=LEFT, padx=10, pady=15)
```

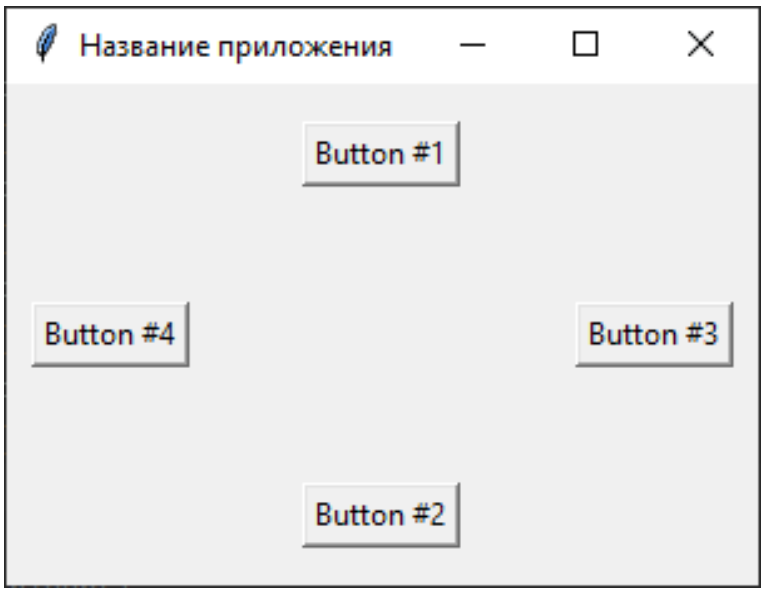

Рисунок 4 – Размещение элементов с использованием метода pack

Однако, применяя данный метод над элементами, невозможно добиться более сложного размещения. Например, при желании разместить элементы квадратом (два сверху и ровно над ними два снизу) необходимо объединять элементы в группы – Frame.

```
1.   btn1 = Button(top_frame, text="Button #1")
2.   btn1.pack(side=LEFT)
3.   btn2 = Button(top_frame, text="Button #2")
4.   btn2.pack(side=RIGHT)
5.   btn3 = Button(bottom_frame, text="Button #3")
6.   btn3.pack(side=RIGHT)
7.   btn4 = Button(bottom_frame, text="Button #4")
8.   btn4.pack(side=LEFT)
```

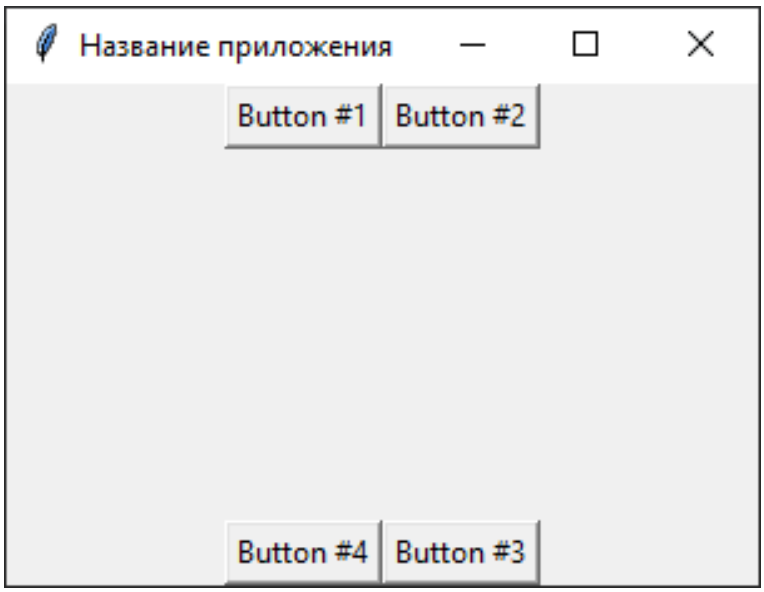

Рисунок 5 – Размещение элементов с использованием группировки

И, наконец, третий *метод place* позволяет применять точное позиционирование элементов. Он принимает следующие параметры:

– height и width (или relheight и relwidth): высота и ширина элемента в пикселях или доля от высоты и ширины родительского контейнера в промежутке от 0.0 до 1.0;

– x и y (или relx и rely): смещение элемента по горизонтали и вертикали относительно верхнего левого угла в пикселях или в доле от высоты и ширины родительского контейнера;

– anchor: растяжение элемента (n, e, s, w, ne, nw, se, sw, c – сокращения от сторон света, где, например, nw (северозапад) – верхний левый угол, а c – центр)).

```
1.   btn1 = Button(window, text="Button #1")
2.   btn1.place(relheight=0.2, relwidth=0.4, relx=0.5,
rely=0.5)
3.   btn2 = Button(window, text="Button #2")
4.   btn2.place(relheight=0.1, relwidth=0.2, relx=0.1,
rely=0.1)
```

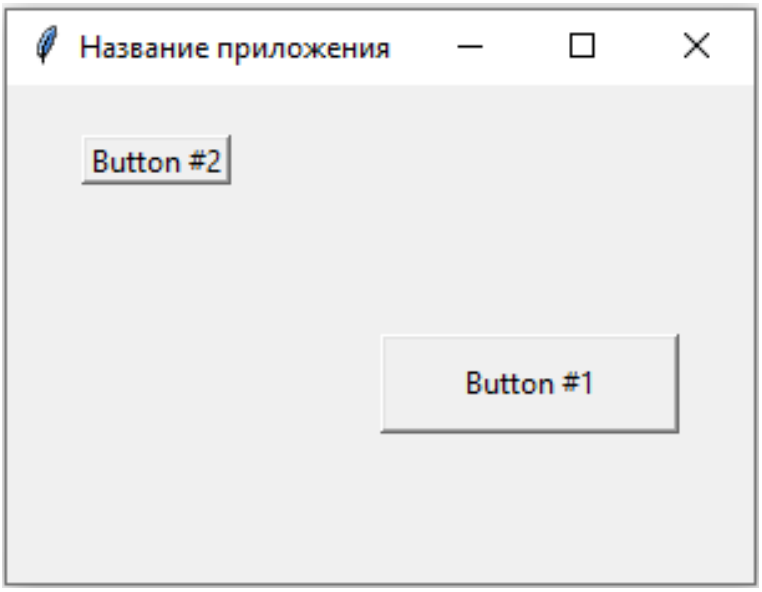

Рисунок 6 – Размещение элементов с использованием метода place

Текстовая метка

Самым простым из элементов является Label, который позволяет выводить статический текст, без возможности ручного редактирования.

Создаем экземпляр класса Label, конструктор которого принимает два аргумента. Первый – ссылка на родительский контейнер, а второй аргумент принимает необходимое количество именованных параметров. В наиболее простом случае достаточно одного опционального параметра text (текст метки).

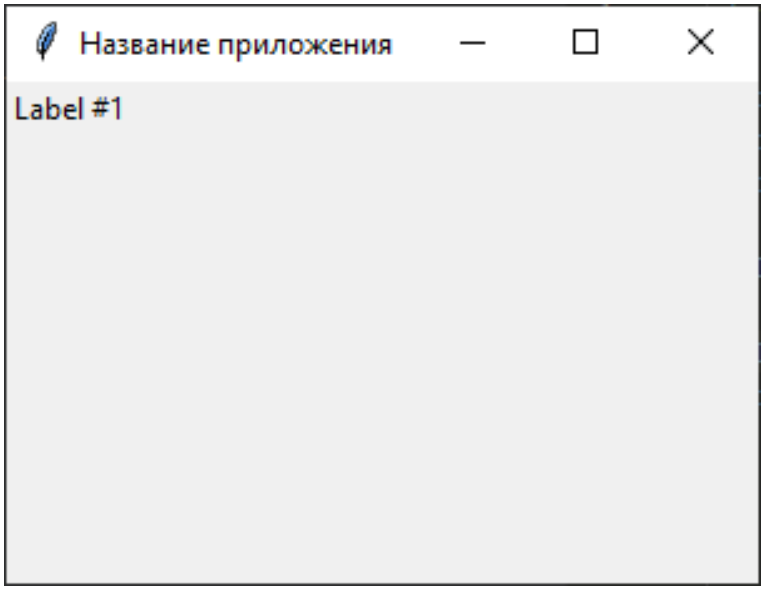

Рисунок 7 – Текстовая метка

Доступных опциональных элементов гораздо больше:

– anchor, height, width, padx, pady: рассмотрены ранее;

– bg: фоновый цвет;

– fg: цвет текста;

– font: шрифт текста;

– bd: толщина границы метки;

– justify: устанавливает выравнивание текста (LEFT, CENTER, RIGHT);

– wraplength: при положительном значении строки текста будут переносится для вмещения в пространство элемента.

```
1.  label1 = Label(window, text = "Label\n#1",
2.                 bg = "Indigo", fg = "White",
3.                 font = ("Courier New", 20, "bold"),
4.                 bd = 10, justify = RIGHT,
5.                 height = 2, width= 15)
6.  label1.grid(column=0, row=0)
```

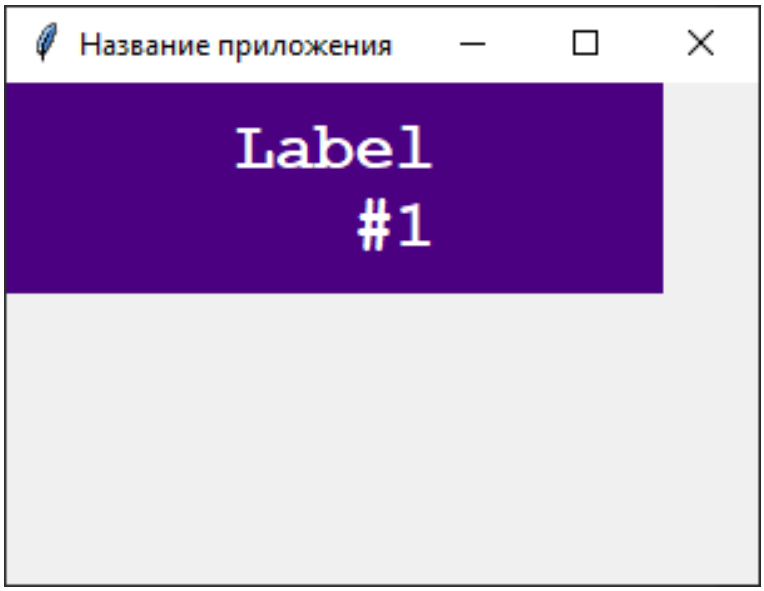

Рисунок 8 – Дополнительная настройка текстовой метки

Кнопка

Одним из наиболее важных элементов является кнопка – Button. Создается также, как, например, метка и обладает примерно тем же набором опциональных параметров. Но добавляется важный параметр:

– command: название функции-обработчика нажатия на кнопку.

```
1.  def btn1_click():
```

```
2.       window.title("Button #1")
3.   button1 = Button(window, text="Button #1", command =
btn1_click)
4.   button1.pack()
```

Например, указанный выше код, при нажатии на кнопку поменяет название окна.

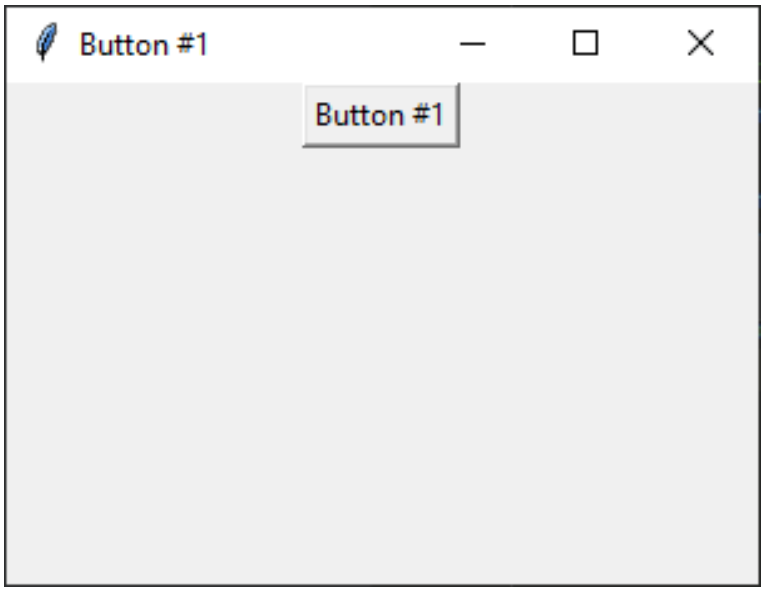

Рисунок 9 – Создание кнопки

Поле ввода

Кроме вывода информации в окне у пользователя должна быть возможность ввода информации для дальнейшей обработки. Для этого существует элемент Entry. Механизм создания и основные параметры опять же совпадают с таковыми у текстовой метки. Дополнительно:

– show: задает маску для вводимых символов (может применяться для скрытия символов пароля на экране, например, звездочками);

– state: состояние элемента (NORMAL – по умолчанию активный и DISABLED – неактивный);

– focus(): установка элемента в фокус, который позволяет сразу писать текст без необходимости ставить курсор на поле ввода.

```
1.   entry1 = Entry(window, show = "*")
2.   entry1.focus()
3.   entry1.pack()
```

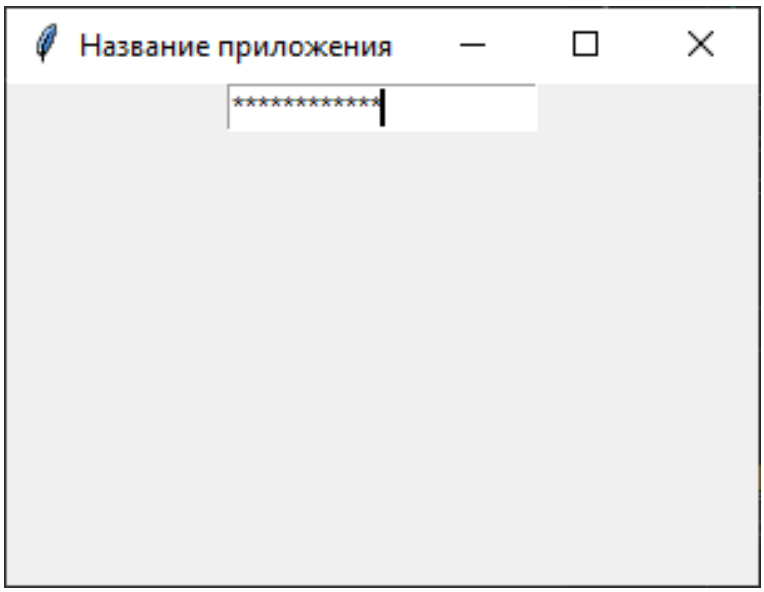

Рисунок 10 – Поле ввода со скрытием вводимых данных

Элемент Entry обладает рядом методов:

– get(): основной метод, возвращающий введенный в поле ввода текст;

– insert(index, str): вставляет в поле ввода по индексу index строку str;

– delete(first, last): удаляет символы с first до last (если он указан, также last может принимать значение END, тогда будут удалены все символы до конца).

Часто необходимо изменять какие-то свойства элементов, например, изменить текст метки, для этого можно воспользоваться следующим методом:

– configure(): параметрами являются изменяемые свойства элемента.

Так, в следующем примере при нажатии на кнопку происходит изменение текста метки на содержимое поля ввода.

```
1.   def button1_click():
2.       label1.configure(text=entry1.get())
3.
4.   label1 = Label(window, text="")
5.   label1.pack()
6.   entry1 = Entry(window, text="")
7.   entry1.pack()
8.   button1 = Button(window, text="Click!", command =
button1_click)
9.   button1.pack()
```

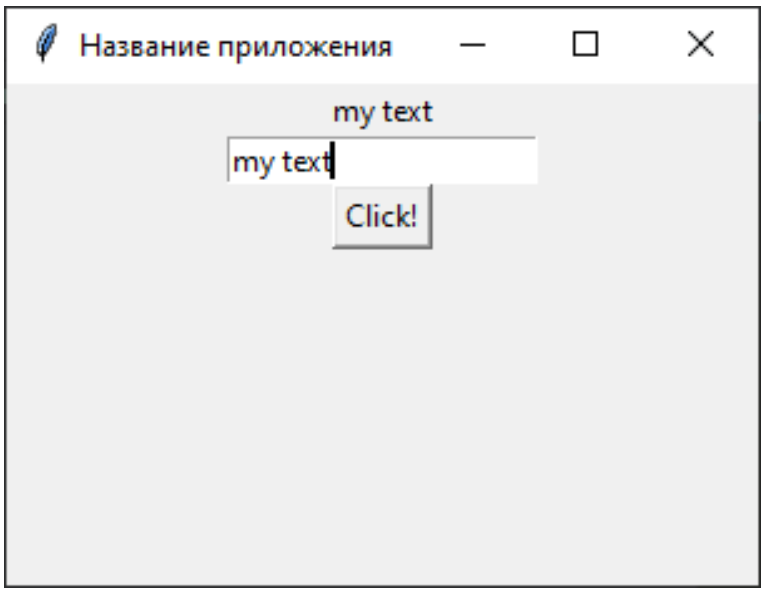

Рисунок 11 – Задание обработчика кнопки

Но есть и другой способ, можно использовать промежуточный объект StringVar (или IntVar, BooleanVar, DoubleVar для других типов данных). Для связи этих объектов с текстом графических элементов, при их создании нужно указывать параметр textvariable.

Например, в следующем примере при каждом нажатии на кнопку значение числа на кнопке увеличивается на единицу.

```
1.   def click ():
2.       clicks.set(clicks.get() + 1)
3.
4.   clicks = IntVar()
5.   clicks.set(0)
6.
7.   button1 = Button(window, textvariable=clicks, command =
click)
8.   button1.pack()
```

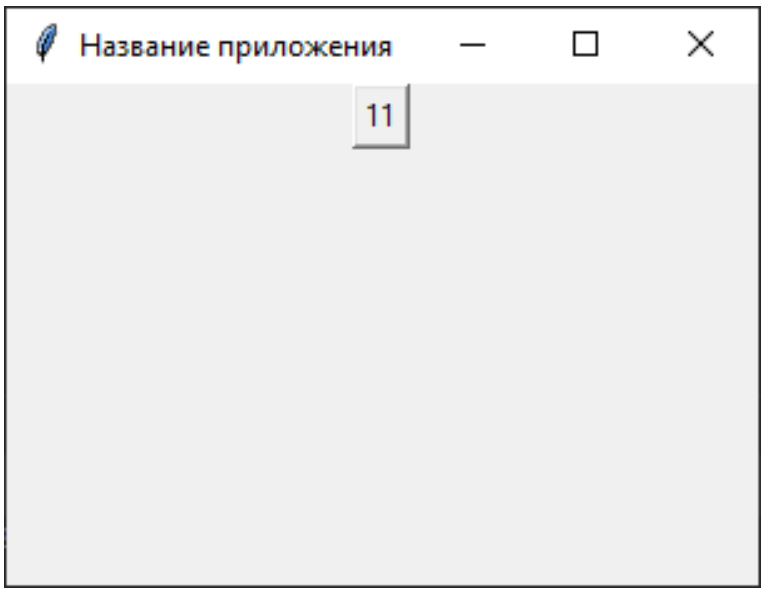

Рисунок 12 – Обработчик кнопки «Подсчет количества нажатий»

Messagebox

Для вывода на экран различных данных или информационных сообщений конечно же можно воспользоваться текстовыми метками, однако для большей наглядности или для более важных сообщений можно вызывать методы объекта как Messagebox:

- showinfo(title, text): обычное окно, в качестве параметров должны идти две строки для заголовка и сообщения;
- showwarning(title, text): окно для предупредительных сообщений;
- showerror(title, text): окно для ошибок.

```
1.  def show_message():
2.      messagebox.showinfo("Title", "Info")
3.      messagebox.showwarning("Title", "Warning")
4.      messagebox.showerror("Title", "Error")
5.
6.  button1 = Button(window, text="MessageBox",
command=show_message)
7.  button1.pack()
```

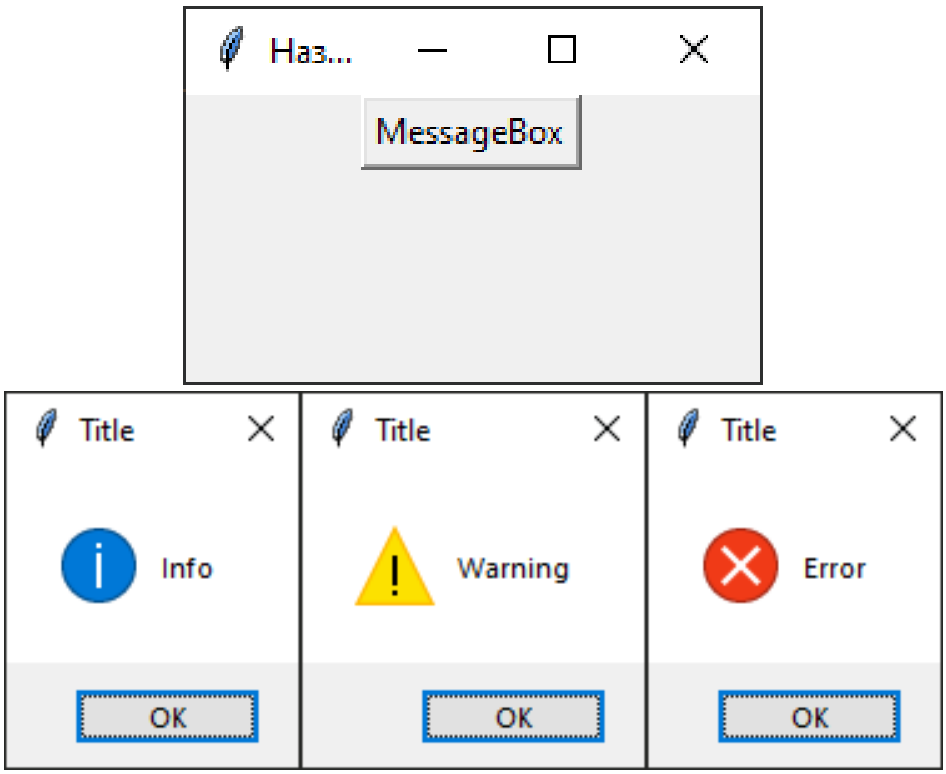

Рисунок 13 – Различные формы информационных сообщений

Для более сложных информационных сообщений, которые будут предоставлять пользователю выбор можно вызывать один из следующий методов: askquestion, askyesno, askyesnocancel, askokcancel, askretrycancel. Эти методы в зависимости от выбора пользователя возвращают одно из значений: True, False, None.

Дополнительные виджеты

Разобравшись с базовыми графическими элементами, перейдем к более сложными виджетам, таким как поле с выпадающим списком (Combobox), поле выбора (Checkbutton), переключатель (Radiobutton). Они определены в модуле tkinter.ttk.

Для создания *поля с выпадающим значением* создадим экземпляр класса *Combobox*. Предустановленные значения можно задать разных типов и предать в экземпляр класса кортежем. Также можно воспользоваться следующими методами:

– current(index): устанавливает выбранный элемент, в метод передается его индекс;

– get(): возвращает выбранное значение элемента.

Для установки *переключателя* создадим экземпляр класса *Checkbutton*, для него можно задать текст вывода, кроме того воспользуемся объектом BooleanVar или IntVar для определения состояния флажка.

Для установки значения в объекты BooleanVar и IntVar воспользуемся методом:

– set(index): в качестве индекса передается 0 (False, т.е. не выбрано,) или 1 (True, т.е. выбрано).

Для создания *radio-кнопок* воспользуемся классом *Radiobutton*, для данного объекта помимо текста необходимо задать уникальное значение для поля value. Работа с выбранным значением осуществляется также через дополнительные объекты, например, IntVar. А также для radio-кнопок можно устанавливать параметр command (функция-обработчик) по аналогии с обычными кнопками.

```
1.  from tkinter import *
2.  from tkinter.ttk import Combobox, Checkbutton,
Radiobutton
3.
4.  window = Tk()
5.  window.title("Название приложения")
6.  window.geometry("300x200")
7.
8.  combobox1 = Combobox(window)
9.  combobox1["values"] = (1, "two", 3, "four", 5)
10. combobox1.current(3)
11. combobox1.grid(row = 0, column = 0)
12.
13. check_state = BooleanVar()
14. check_state.set(True)
15.
16. checkbutton1 = Checkbutton(window, text = "Check",
variable = check_state)
17. checkbutton1.grid(row = 0, column = 1)
18.
19. radio_state = IntVar()
20. radio_state.set(2)
```

```
21.
22.  radiobutton1 = Radiobutton(window, text = "Radio 1",
value = 1, variable = radio_state)
23.  radiobutton1.grid(row = 1, column = 1)
24.  radiobutton2 = Radiobutton(window, text = "Radio 2",
value = 2, variable = radio_state)
25.  radiobutton2.grid(row = 2, column = 1)
26.  window.mainloop()
```

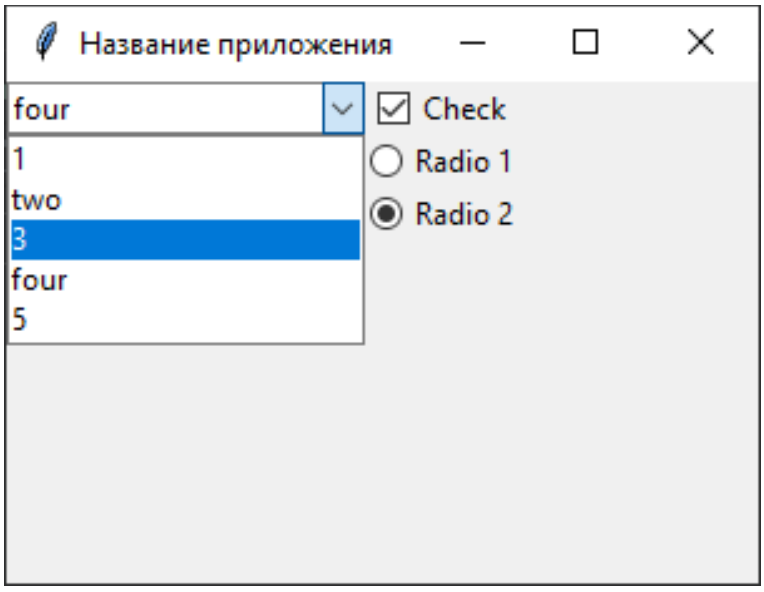

Рисунок 14 – Дополнительные элементы графического интерфейса tkinter

Часто перед чтением из файла или перед записью необходимо у пользователя попросить выбрать нужно. Для этих целей служит объект filedialog и воспользовавшись одним из его методов можно получить путь к нужному файлу:

– askopenfilename(): получить путь к файлу, можно передать парметр filetypes для определения типа файла и initialdir для указания начальной диркектории для поиска;

– askopenfilenames(): для выбора нескольких файлов;

– askdirectory(): для выбора папки.

```
1.   from tkinter import filedialog
2.   file = filedialog.askopenfilename(initialdir =
"C://Users",
```

```
3.            filetypes = (("txt files","*.txt"),("all
files","*.*")))
```

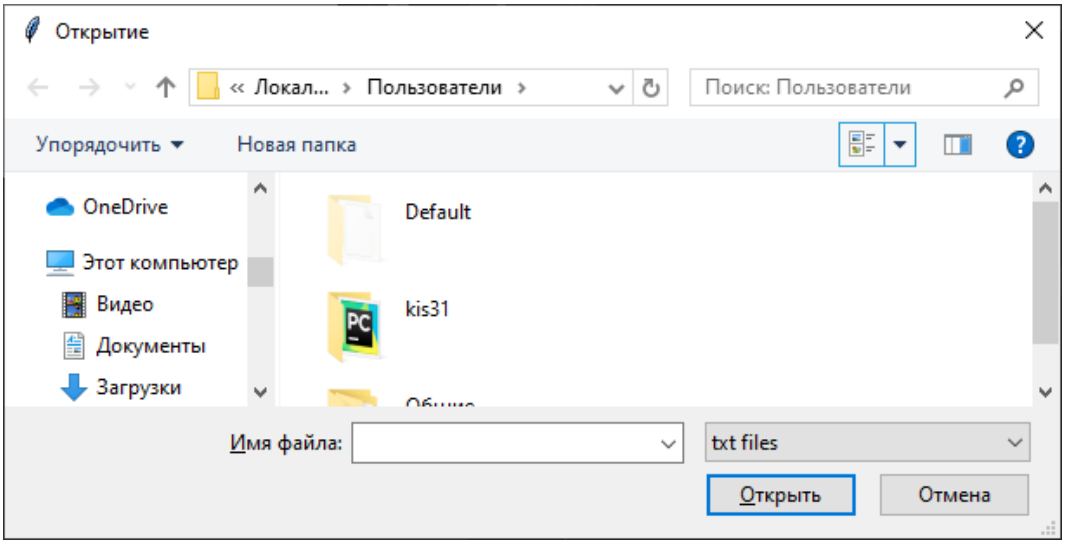

Рисунок 15 – Окно открытия файла

Иногда для удобства пользователя логичным является разделить функционал приложения на части и сгруппировать соответствующие элементы на разных вкладках. Для управления вкладками можно воспользоваться элементом Notebook, а вкладками будут являться экземпляры классов Frame.

```
1.  from tkinter import *
2.  from tkinter.ttk import Notebook, Frame
3.
4.  window = Tk()
5.  window.title("Название приложения")
6.  window.geometry("300x200")
7.
8.  tab_control = Notebook(window)
9.  tab1 = Frame(tab_control)
10. tab_control.add(tab1, text="Первая вкладка")
11. button1 = Button(tab1, text="Button #1")
12. button1.pack()
13.
14. tab2 = Frame(tab_control)
15. tab_control.add(tab2, text="Вторая вкладка")
```

```
16.  button2 = Button(tab2, text="Button #2")
17.  button2.pack()
18.
19.  tab_control.pack(expand = True, fill = BOTH)
20.  window.mainloop()
```

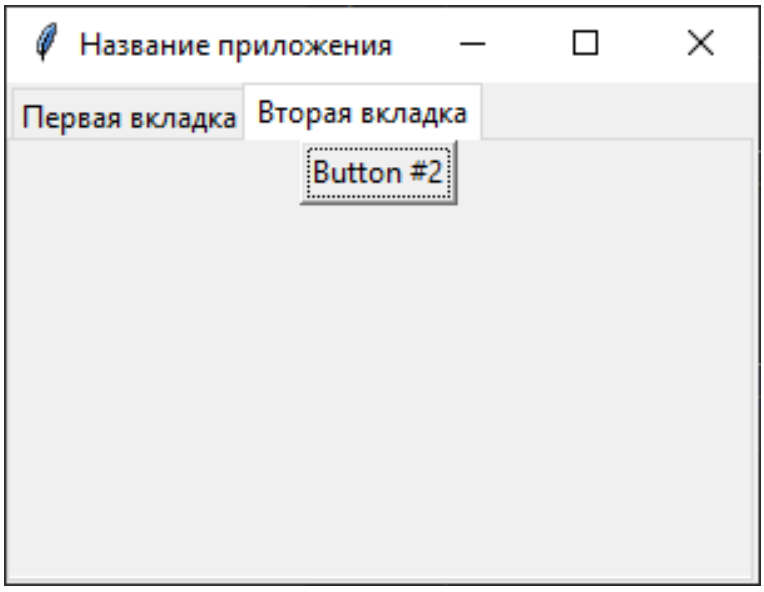

Рисунок 16 – Создание вкладок