2. ОБРАБОТКА СОБЫТИЙ

2.1. СВЯЗЫВАНИЕ ВИДЖЕТОВ С СОБЫТИЯМИ И ДЕЙСТВИЯМИ. МЕТОД BIND()

Схема создания любого приложения с графическим интерфейсом пользователя заключается в связывании между собой виджета, события и действия.

Например:

• виджет — кнопка, событие — щелчок левой кнопкой мыши по кнопке, действие — запуск вычислительного процесса, открытие диалогового окна, вывод изображения и т. д.;

• виджет — текстовое поле, событие — нажатие клавиши **Enter** на клавиатуре, действие — получение текста из поля с помощью метода **get()** для последующей обработки программой.

За выполняемое действие отвечает функция-обработчик (или метод), которая вызывается при наступлении события. При этом один и тот же виджет может быть связан с несколькими событиями.

Для вызова функции-обработчика для виджета можно использовать аргумент **command**, который надо связать с именем этой функции. В качестве события, запускающего в этом случае функцию на выполнение, будет выступать щелчок левой кнопкой мыши.

Однако существует и другой способ связывания между собой виджета, события и действия — это метод **bind()**. Функциообработчики, вызываемые методом **bind()**, должны иметь обязательный аргумент **event**, через который передается событие. В данном случае имя **event** — это просто некое соглашение. В принципе, идентификатор может иметь любое другое имя. Обязательное условие одно — он должен быть первым в списке аргументов функции, или стоять на втором месте в списке аргументов метода в объектноориентированных программах.

Пример задачи: написать объектно-ориентированную программу, в которой одна и та же функция-обработчик используется для разных событий. В результате работы программы должен изменяться цвет текста на кнопке и цвет фона кнопки как при щелчке по ней мышью, так и при нажатии клавиши **Enter**.

Решение:

```
from tkinter import *
window = Tk()
window.title("Обработка событий")
#Класс-кнопка
class MyButton:
 #Конструктор
 def __init__(self):
     self.btn    =    Button(text='Обработка    событий',width=30,
height=3,font = ("Comic Sans MS", 14, "bold" ))
     self.btn.bind('<Button-1>', self.change)
     self.btn.bind('<Return>', self.change)
     self.btn.pack()
 #Метод-обработчик
 def change(self, event):
     self.btn['fg'] = "chartreuse"
     self.btn['bg'] = "ivory"

#Создаем кнопку как объект класса MyButton
MyButton()
window.mainloop()
```

Результат работы программы представлен на рисунке 1.

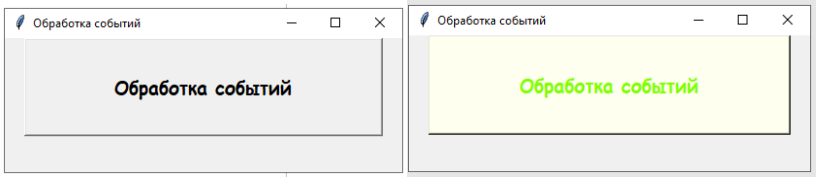

Рис.1

Однако такой подход вряд ли можно назвать оптимальным, поскольку код обоих функций-обработчиков в данном случае фактически идентичен. Другим решением этой задачи могло бы быть создание одной функции-обработчика, например, с именем **change_color()**, которая в качестве дополнительного аргумента получала бы цвет шрифта. Однако при этом возникает проблема передачи этого аргумента в метод **bind()**, поскольку указание списка аргументов для функции **change_color()** в вызове метода **bind()** означало бы вызов самой этой функции.

```
def change_color(event, col):
    lb['fg'] = col
b.bind('<Button-1>', change_color(event, 'blue'))
```

При этом надо иметь в виду, что все функции в языке Python возвращают значение. Даже в том случае, когда оператор **return** отсутствует в функции, функция вернет значение **None**. Поэтому в данном случае даже при правильно переданных аргументах функция **change_color()** метода **bind()** получит в качестве второго аргумента значение **None**, а не объект-функцию.

Для решения этой проблемы можно использовать **лямбда функции**.

Пример задачи: реализовать предыдущую программу с использованием лямбда-функций при работе с кнопками.

Решение:

```
from tkinter import *
window = Tk()
window.title("Обработка разных событий")
#Функция обработчик (одна функция для нескольких событий)
def change_color(col):
    lb['fg'] = col
#Создаем метку
lb = Label(text = "Обработка разных событий")
lb.pack()
#Создаем кнопку, окрашивающую текст в чайный цвет,и сразу размещаем
ее
Button(command = lambda col = 'teal':
change_color(col)).pack()
#Создаем кнопку, окрашивающую текст в темно-фиолетовый цвет, и сразу
размещаем ее
Button(command = lambda col = 'darkviolet':
change_color(col)).pack()
window.mainloop()
```

Результат работы программы представлен на рисунке 2.

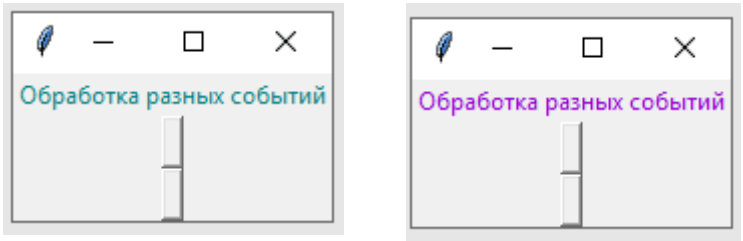

Рис.2

2.2. ВИДЫ СОБЫТИЙ

Под событием в приложениях с графическим интерфейсом пользователя, как правило, подразумевается воздействие пользователя на элементы интерфейса. Принято выделять три основных типа событий:

- события, связанные с мышью;
- события, связанные с клавиатурой;

• события, связанные с изменением виджетов.

Иногда событие может представлять собой комбинацию указанных событий (например, нажатие мыши при нажатой клавише на клавиатуре).

К наиболее часто используемым событиям, связанным с мышью, относятся следующие события:

• <Button-1> — щелчок левой кнопкой мыши;
• <Button-2> — щелчок средней кнопкой мыши;
• <Button-3> — щелчок правой кнопкой мыши;
• <Double-Button-1> — двойной щелчок левой кнопкой мыши;
• <Motion> — движение мышью;
• и др.

Пример задачи: написать программу, в которой необходимо реализовать возможность изменения заголовка главного окна в зависимости от действий пользователя (щелчок левой кнопкой, щелчок правой кнопкой, движение мышью).

Решение:

```
from tkinter import *
#Функции-обработчики различных событий
def mouse_left(event):
    window.title("Левая кнопка мыши")
def mouse_right(event):
    window.title("Правая кнопка мыши")
def mouse_move(event):
    x = event.x
    y = event.y
    xy = "Движение мышью {0}x{1}".format(x, y)
    window.title(xy)
window = Tk()
window.minsize(width = 600, height = 300)
window.bind('<Button-1>', mouse_left)
window.bind('<Button-3>', mouse_right)
window.bind('<Motion>', mouse_move)
window.mainloop()
```

Результат работы программы представлен на рисунке 3.

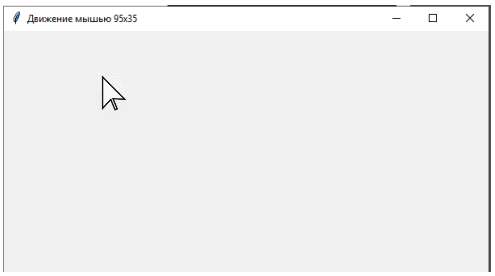

Рис.3

Событие (**event**) в tkinter — это объект со своими атрибутами.

В предыдущей программе в качестве виджета выступает главное окно приложения, а в качестве события — перемещение мыши. В функции **mouse_move()** происходит извлечение значений атрибутов x и y объекта **event**, в которых хранятся координаты местоположения курсора мыши в системе координат главного окна приложения.

При обработке событий, связанных с клавиатурой, следует иметь в виду, что:

- буквенные клавиши можно записывать и без угловых скобок, например ‘f’, ‘k’ и т. д.;
- для небуквенных клавиш существуют специальные зарезервированные слова, например:
  - ▪ <**Return**> — нажатие клавиши <**Enter**>;
  - ▪ <**space**> — пробел;
  - сочетания пишутся через тире; в случае использования так называемого модификатора он указывается первым, например:

- ▪ <**Shift-Up**> — одновременное нажатие клавиш <**Shift**> и стрелки вверх;
- ▪ <**Control-B1-Motion**> — движение мышью с нажатой левой кнопкой и клавишей <**Ctrl**>.

При работе с фокусом следует иметь в виду, что:

• событие получения фокуса обозначается как <**FocusIn**>;

• событие снятия фокуса обозначается как <**FocusOut**>;

• фокус перемещается по виджетам при нажатии клавиш <**Tab**>, <**Ctrl+Tab**>, <**Shift+Tab**>, а также при щелчке по ним мышью (за исключением кнопок).

Пример задачи: написать программу, в которой:

• при нажатии клавиши <**Enter**> текст, введенный пользователем в текстовое поле, копируется в метку;

• при нажатии комбинации клавиш <**Ctrl+t**> этот текст выделяется;

• при нажатии комбинации клавиш <**Ctrl+q**> происходит выход из приложения.

Решение:

```
from tkinter import *
window = Tk()
window.title("Вывод текстовой строки в окно")
#Функции-обработчики различных событий
def close_win(event):
    window.destroy()

def text_to_Label(event):
    s = t.get()
    lbl.configure(text = s)
def select_All(event):
    window.after(10, select_all, event.widget)
def select_all(widget):
    widget.selection_range(0, END)
    widget.icursor(END) #Установка курсора в конец
#Создаем текстовое поле
t = Entry(width = 50)
t.focus_set() #Устанавливаем фокус в текстовое поле
t.pack()
#Создаем метку
lbl = Label(height = 4, fg = 'teal', bg = 'darkviolet', font =
('Comic Sans MS', 16,'bold'))
lbl.pack(fill = X)
t.bind('<Return>', text_to_Label)
t.bind('<Control-t>', select_All)
window.bind('<Control-q>', close_win)
window.mainloop()
```

Результат работы программы представлен на рисунке 4.

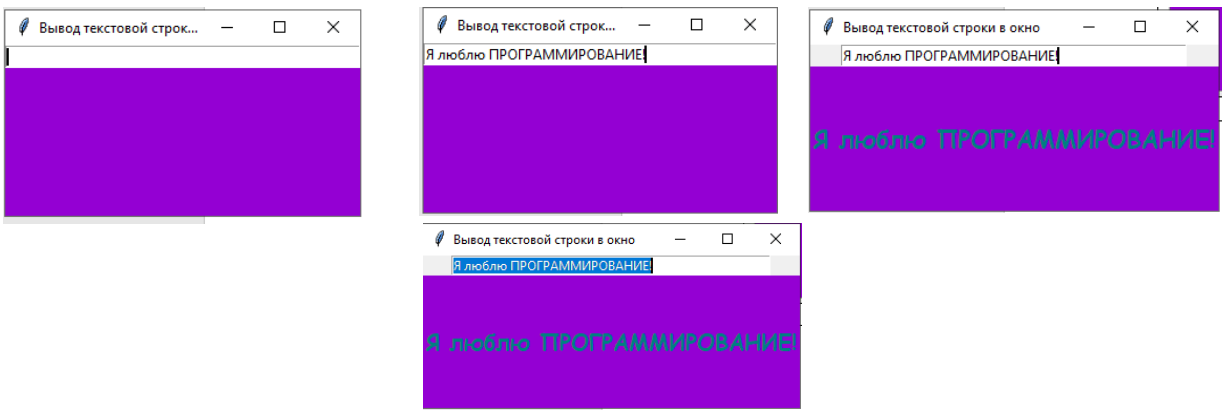

Рис.4

В данной программе метод **after()** выполняет функцию, указанную во втором аргументе (функция **select_all()**), через промежуток времени, указанный в первом аргументе. В третьем аргументе передается значение атрибута **widget** объекта **event**. В качестве виджета используется текстовое поле t, которое передается в качестве аргумента в функцию **select_all().**

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

1. Составьте программу, которая будет позволять изменять размеры многострочного текстового поля.

Комментарии к программе:

1) размеры многострочного текстового поля (Text) должны определяться значениями, вводимыми пользователем в однострочные текстовые поля (Entry);

2) изменение размера должно происходить при совершении следующих событий:

− нажатии соответствующей кнопки;

− нажатии на клавиатуре клавиши <Enter>;

3) цвет фона экземпляров Entry должен быть светло-серым (lightgrey), когда поле не в фокусе, и белым (white), когда поле в фокусе.

Возможный результат работы программы представлен на рисунке 5.

Рис. 5

2. Составьте программу с графическим пользовательским интерфейсом, представляющую собой системную утилиту, позволяющую проводить манипуляции с файлами.

Комментарии к программе:

1) программа должна предоставлять пользователю следующие возможности:

- выводить на экран список доступных директорий;
- выводить на экран список файлов из выбранной директории;
- давать пользователю производить определенные действия с файлами, например:

− дублировать все файлы в текущей директории;

− дублировать конкретный файл;

− удалять дубликаты файлов из директории;

− удалять пустые директории;

− удалять из конкретной директории файлы определенного типа (с определенным расширением);

− переименовывать какие-либо файлы по выбранному признаку;

− переименовывать конкретный файл;

− перемещать файлы из одной директории в другую;

2) набор возможных действий с файлами может быть произвольным, но не менее трех;

3) в программе необходимо использовать как можно большее количество функций и переменных из модулей **os, sys, shutil, psutil** (изучить самостоятельно);

4) внешний вид интерфейса — произвольный.

Возможный результат работы программы представлен на рисунках 6–8.

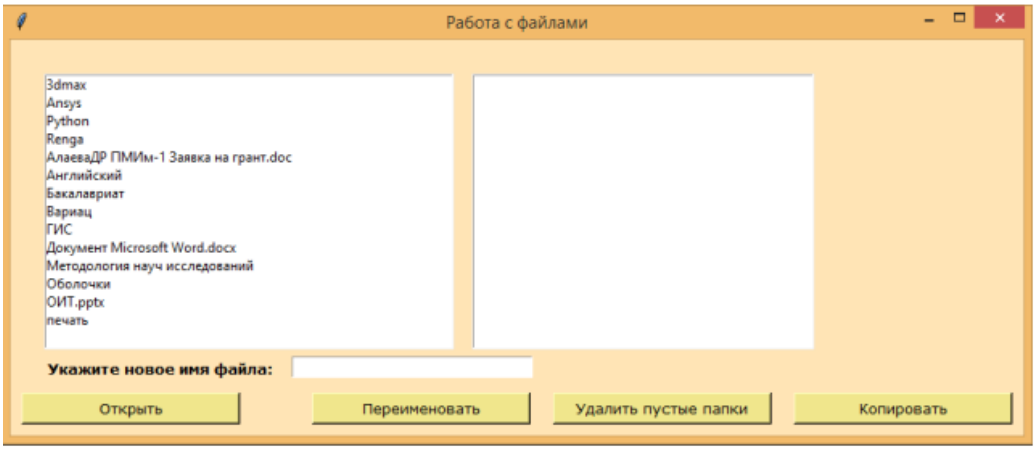

Рис. 6

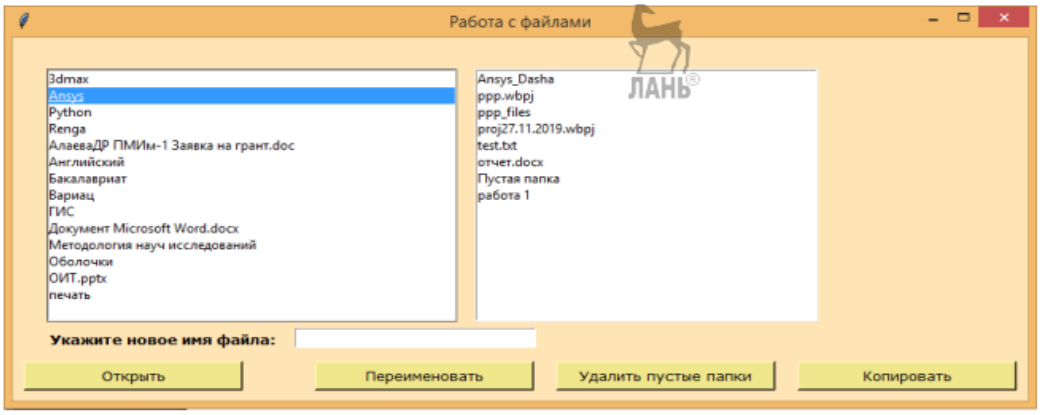

Рис. 7

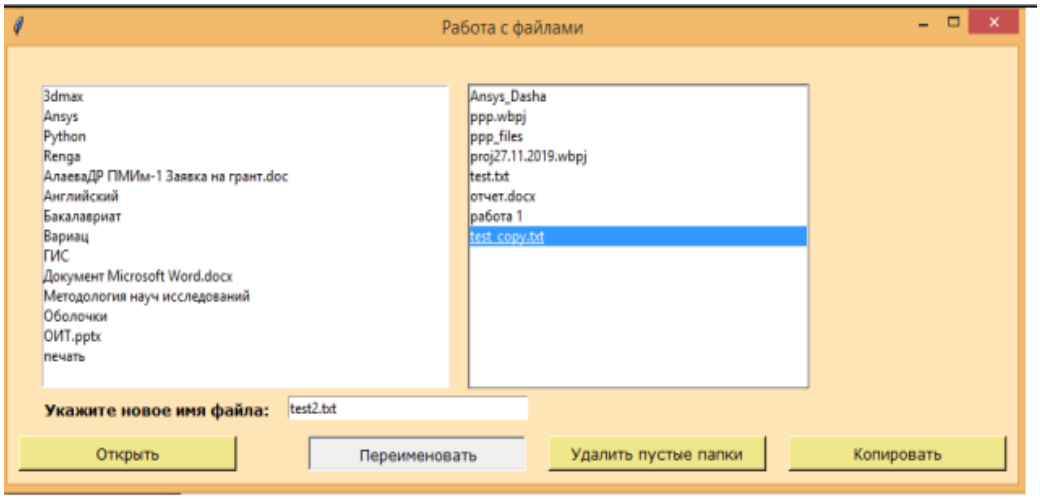

Рис. 8