

# Создание графического интерфейса

Модуль tkinter. Создание главного окна и отображение.  
Виджеты и их свойства. Кнопки, метки, радио-кнопки и т.д.  
Определение событий и их обработчиков.

[Введение](#)

[Стандартная библиотека tkinter](#)

[Создание главного окна и отображение](#)

[Виджеты и их свойства](#)

[Button](#)

[Text](#)

[Checkbutton](#)

[Scrollbar](#)

[Упаковщики](#)

[Grid](#)

[Pack](#)

[Place](#)

[Определение событий и их обработчиков](#)

[Command](#)

[Bind](#)

[Домашнее задание](#)

[Дополнительные материалы](#)

## Используемая литература

# Введение

На этом уроке мы познакомимся со стандартной библиотекой tkinter. С помощью данной библиотеки можно создавать графические интерфейсы для различных приложений. Мы рассмотрим основные её возможности: главное окно, виджеты и их свойства, события и их обработчик.

## Стандартная библиотека tkinter

Tkinter (от англ. tk interface) - это графическая библиотека, позволяющая создавать приложения с оконным интерфейсом tcl/tk. Tkinter, как и tcl/tk, является кроссплатформенной библиотекой и используется в большинстве распространённых операционных систем (Windows, Linux, Mac OS X и др.).

Для создания графического интерфейса приложения необходимо осуществить следующие шаги:

- импорт библиотеки;
- создание главного окна;
- создание виджетов и определение их расположения;
- установка их свойств;
- определение событий;
- определение обработчиков событий;
- отображение главного окна.

Как и любой модуль, tkinter в Python можно импортировать двумя способами: командами `import tkinter` или `from tkinter import *`. В дальнейшем мы будем пользоваться только вторым способом, т. к. это позволит не указывать каждый раз имя модуля при обращении к объектам, которые в нём содержатся.

Для подключения библиотеки первой строкой программы, мы с вами пропишем:

```
from tkinter import *
```

## Создание главного окна и отображение

В современных операционных системах любое пользовательское приложение заключено в окно, которое можно назвать главным, т.к. в нём располагаются все остальные виджеты. Объект окна верхнего уровня создаётся при обращении к классу Tk модуля tkinter. Переменную, являющуюся главным окном, принято называть root (хотя понятно, что можно назвать как угодно, но так уж принято). Таким образом, вторая строчка кода будет выглядеть так:

```
root = Tk()
```

У объекта-окно есть различные методы для работы с ним, мы с вами рассмотрим несколько. Первое, давайте зададим заголовок окна, чтобы было понятно, что у нас за приложение, для этого мы используем метод `title`, параметром метода является строка. Так как мы будем разрабатывать графический интерфейс приложения, разработанного в предыдущем уроке, то давайте установим наименование окна как "Информационная система БИБЛИОТЕКА".

```
root.title("Информационная система БИБЛИОТЕКА")
```

Теперь давайте установить начальные размеры главного окна, с помощью метода `minsize()`, входными параметрами он принимает ширину и высоту окна соответственно.

```
root.minsize(500,300)
```

Также мы можем либо разрешить пользователю менять размеры открывшегося окна, либо запретить с помощью метода `resizable()`. Входными параметрами он принимает значения `true` и `false` для изменения ширины и высоты окна. Мы выключим возможность изменять размер окна:

```
root.resizable(width=False, height=False)
```

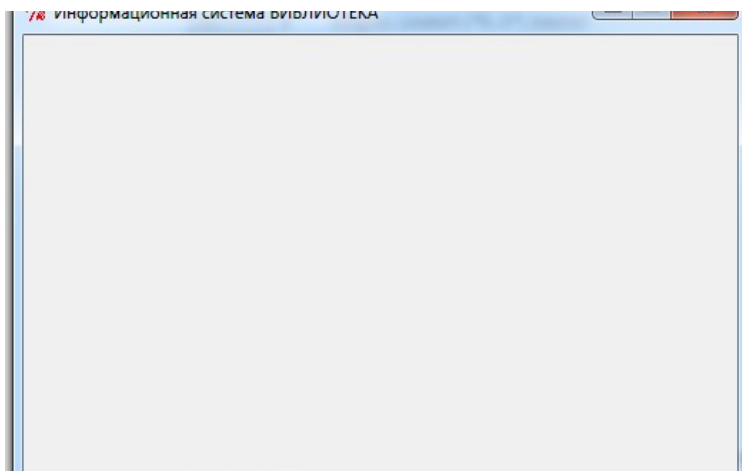
После того, как мы определили все основные составляющие главного окна, мы можем его отобразить. Отобразить объект-окно можно с помощью метода `mainloop()`.

```
root.mainloop()
```

Теперь давайте соберем всё, что написали, воедино.

```
root = Tk()
root.title("Информационная система БИБЛИОТЕКА")
root.minsize(500,300)
root.resizable(width=False, height=False)
root.mainloop()
```

И в итоге мы должны увидеть следующее окно:



## Виджеты и их свойства

После того, как мы с вами создали главное окно, необходимо определить, какие виджеты на нём будут располагаться. Мы будем разрабатывать графический интерфейс для приложения, которое разработали на предыдущем уроке. Наше приложение управляет информацией о книгах и журналах, содержащихся в библиотеке. Соответственно, мы должны создать кнопки для вызова методов, которые мы описали в классе `Library`. Давайте вспомним, какие методы мы с Вами написали:

- Распечатка всего списка авторов, содержащихся в библиотеке.
- Распечатка всего списка книг, содержащихся в библиотеке.
- Распечатка всего списка журналов, содержащихся в библиотеке.
- Добавление нового автора.
- Добавление новой книги.
- Добавление нового журнала.
- Удаление автора.
- Удаление книги.
- Удаление журнала.

Следовательно, для каждого из методов должна быть реализована кнопка. Сейчас мы реализуем три кнопки для работы с информацией об авторе, остальные для закрепления вы разработаете самостоятельно.

С кнопками мы разобрались, но нам необходимо где-то отображать результат работы методов для этого можно использовать виджет text.

## Button

После того, как мы разобрались, какие кнопки необходимо реализовать, давайте приступим к ознакомлению виджета, осуществляющего работу кнопки – button.

Кнопка создаётся при обращении к классу Button библиотеки tkinter. Объект “кнопка” связывается с какой-нибудь переменной. У класса Button (как и всех остальных классов, за исключением Tk) есть обязательный параметр — объект, которому кнопка принадлежит (кнопка не может быть “ничейной”). Пока у нас есть единственное окно (root), оно и будет аргументом, передаваемым в класс при создании объекта-кнопки. Теперь давайте создадим первую кнопку:

```
but = Button(root)
```

У объекта-кнопки много различных свойств, мы рассмотрим основные, а именно:

- Текст надписи – text.
- Размеры кнопки: ширина и длина - width,height.
- Цвет кнопки – bg
- Цвет текста – fg..
- Шрифт текста – font.

Текст надписи должен соответствовать назначению метода, который будет вызываться по нажатию, цвета кнопки и текста может установить по усмотрению, в примере мы рассмотрим: цвет кнопки – белый, цвет текста – черный. Шрифт возьмем Arial.

```
but = Button(root, text="Посмотреть всех авторов", bg="white", fg="black", font="Arial")
```

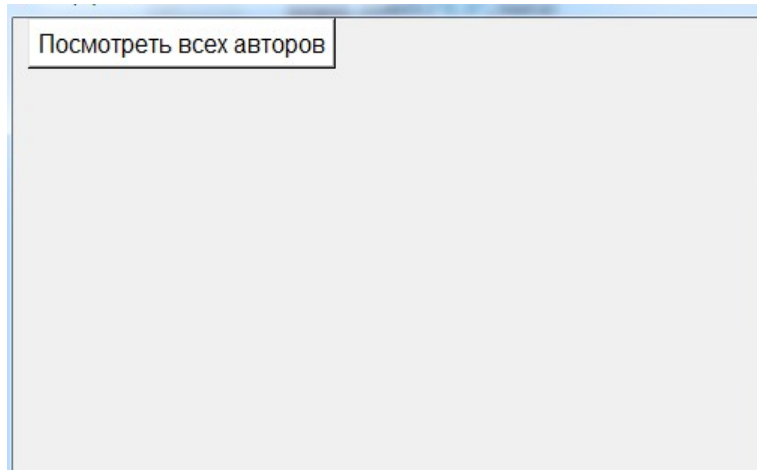
Теперь нам надо отобразить кнопку в главном окне, для этого используется метод grid(), он принимает следующие аргументы:

- row - номер строки, в который помещаем виджет;
- rowspan - сколько строк занимает виджет;
- column - номер столбца, в который помещаем виджет;
- columnspan - сколько столбцов занимает виджет;
- padx / pady - размер внешней границы (бордюра) по горизонтали и вертикали.

```
but = Button(root, text="Посмотреть всех авторов", bg="white", fg="black", font="Arial").grid(row=1,
```

```
column=1, padx=(10, 10))
```

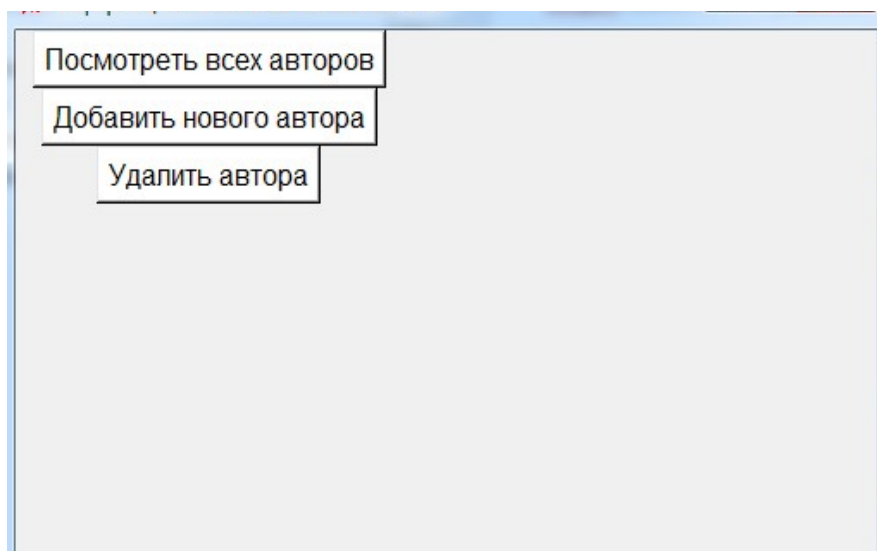
В итоге мы должны увидеть окно, содержащее одну кнопку:



Таким же образом создаём остальные две кнопки для методов: добавление автора и удаление автора.

```
but3 = Button(root, text="Добавить нового автора",bg="white", fg="black", font="Arial").grid(row=2,
column=1,                                padx=(10, 10)),
but5 = Button(root, text="Удалить автора", bg="white", fg="black", font="Arial").grid(row=3, column=1,
padx=(10, 10))
```

Получаем окно с тремя кнопками:



Но кнопки разного размера выглядят не очень красиво, поэтому давайте определим одинаковые размеры для каждой кнопки.

```
but = Button(root, text="Посмотреть всех авторов", bg="white", fg="black", font="Arial",
width=22,height=1).grid(row=1, column=1, padx=(10, 10))
```

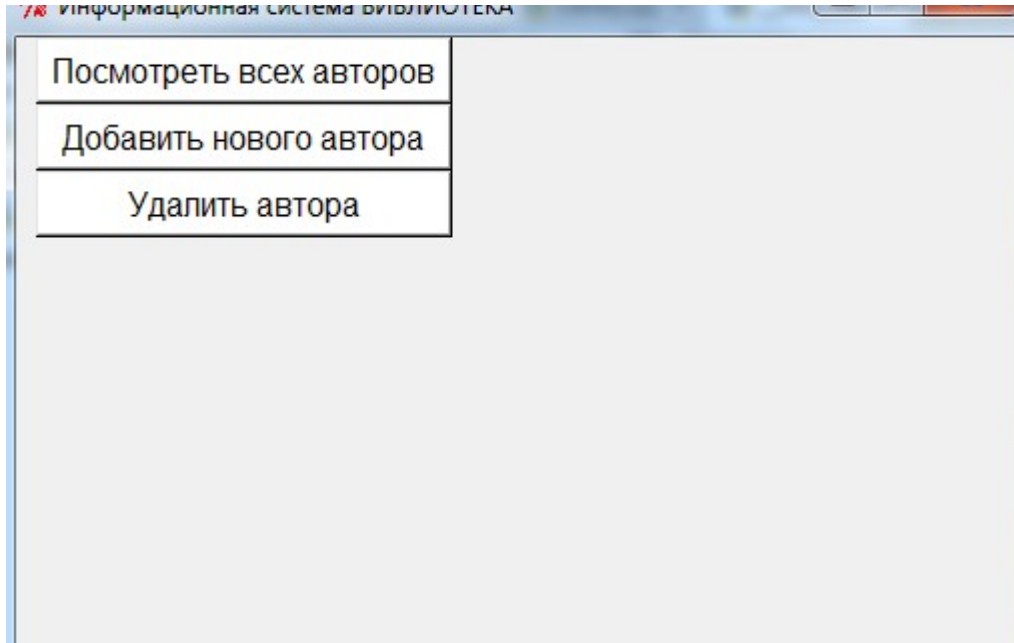
```

but3 = Button(root, text="Добавить нового автора",bg="white", fg="black", font="Arial",
width=22,height=1).grid(row=2, column=1, padx=(10, 10))

but5 = Button(root, text="Удалить автора", bg="white", fg="black", font="Arial",
width=22,height=1).grid(row=3, column=1, padx=(10, 10))

```

Теперь мы получим аккуратно расположенные одинаковые кнопки:



## Text

После того, как мы реализовали кнопки, нам необходимо создать область для отображения результатов выполнения методов, которые мы вызовем при нажатии на кнопки. Для этого мы можем воспользоваться виджетом text.

Виджет текст создаётся при обращении к классу Text библиотеки tkinter. Объект "текст" связывается с какой-нибудь переменной. У класса Text так же есть обязательный параметр — объект, которому текст принадлежит. В нашем случае это окно (root). Теперь давайте создадим объект-текст:

```
output = Text(root)
```

У объекта-текст, также как и у любого объекта библиотеки tkinter, есть различные свойства. Мы рассмотрим цвет фона, шрифт выводимого текста и ширину, и высоту области вывода результата.

```
output = Text(root, bg="white", font="Arial 12", width=60, height=10)
```

Соответственно:

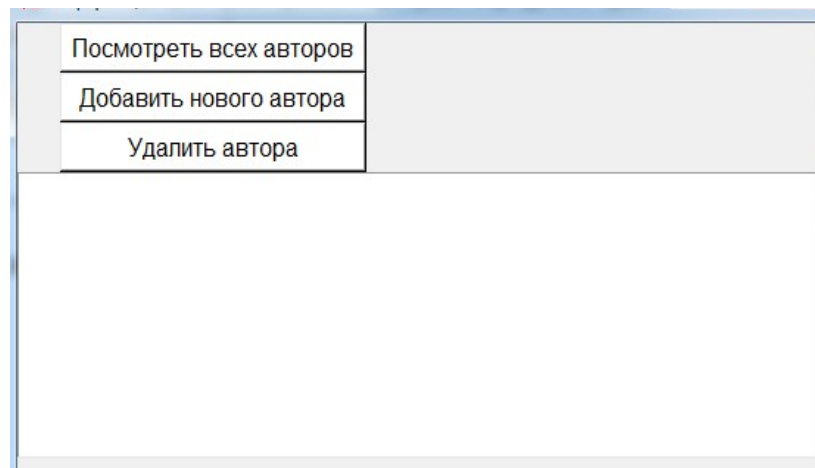
- bg – цвет фона;
- font – шрифт выводимого текста;
- width – ширина поля вывода;

- height – высота поля вывода.

Теперь давайте реализуем отображение данного виджета опять же с помощью метода grid().

```
output = Text(root, bg="white", font="Arial 12", width=60, height=10).grid(row=6, column=1,
columnspan=8)
```

В итоге мы получим следующее главное окно:



## Checkbutton

Рассмотрим пару часто используемых виджетов.

Объект checkbutton предназначен для выбора не взаимоисключающих пунктов в окне (в группе можно активировать один, два или более флажков или не один). Значение каждого флажка привязывается к своей переменной, значение которой определяется опциями onvalue (включено) и offvalue (выключено) в описании флажка.

Давайте рассмотрим небольшой пример использования checkbutton.

```
c1 = IntVar()
c2 = IntVar()
che1 = Checkbutton(root, text="Первый флажок", variable=c1, onvalue=1, offvalue=0).grid(row=13,
column=1)
che2 = Checkbutton(root, text="Второй флажок", variable=c2, onvalue=2, offvalue=0).grid(row=14,
column=1)
```

В данном примере мы создали два флажка – checkbutton. Значение для флажка che1 привязывается к переменной c1, а che2, соответственно, привязывается к c2. Для того, чтобы включить значение флажка, необходимо задать какое-либо значение, в нашем случае значения должны быть целочисленными, опции onvalue.

## C scrollbar

Данный виджет позволяет прокручивать содержимое другого виджета (например, текстового поля или списка). Прокрутка может быть как по горизонтали, так и по вертикали.

Рассмотрим пример использования виджета scrollbar для прокрутки по вертикали текстового поля.



```
output = Text(root, bg="white", font="Arial 12", width=50, height=10)
output.grid(row=6, column=1, columnspan=7)
scr = Scrollbar(root, command=output.yview)
output.configure(yscrollcommand=scr.set)
scr.grid(row=6, column=8)
```

Мы создаём переменную `scr`, отвечающую за объект прокрутки текстового поля `output`. Чтобы привязать объект прокрутки с полем `output`, используем опцию `command`, а для того, чтобы определить, что прокрутка будет по вертикали, используем `yview`. Далее поле `output` конфигурируется с помощью метода `configure`: устанавливается значение опции `yscrollcommand`.

## Упаковщики

В библиотеке `tkinter` существует несколько способов отобразить виджеты на интерфейсном окне. Для этого используются такие функции, как упаковщики, к ним относятся:

- `grid()` – воспринимает интерфейсное окно как сетку или таблицу и располагается виджеты на пересечении строк и столбцов.
- `pack()` - автоматически размещает виджет в родительском окне.
- `place()` - размещает виджет в фиксированном месте с фиксированным размером.

В течение всего урока мы с вами использовали метод `grid`, поэтому сначала рассмотрим его.

### Grid

Функция `Grid` размещает виджеты на сетке, то есть рабочее окно он представляет в виде сетки (если понятнее, матрицы) из строк и столбцов. Основными параметрами метода являются: `row/column` – строка/столбец в сетке и `rowspan/columnspan` – сколько строк/столбцов занимает виджет.

Например, мы использовали функцию `grid()` для размещения кнопок. Давайте рассмотрим их повнимательнее:

```
but = Button(root, text="Посмотреть всех автопов", bg="white", fg="black", font="Arial",
width=22,height=1).grid(row=1, column=1, padx=(1, 1))

but3 = Button(root, text="Добавить нового автора", bg="white", fg="black", font="Arial",
width=22,height=1).grid(row=2, column=1, padx=(1, 1))

but5 = Button(root, text="Удалить автора", bg="white", fg="black", font="Arial",
width=22,height=1).grid(row=3, column=1, padx=(1, 1))
```

Функция `grid()` для первой кнопки определяет, что кнопка будет располагаться на пересечении первой строки и первого столбца сетки/матрицы, а дополнительное свойство `padx` определяет отступ от виджета по оси `x` и оси `y`.

Соответственно, чтобы расположить кнопки друг под другом, мы должны определить у кнопок было одинаковое значение столбца (`column=1`).

### Pack

Функция `pack()` — автоматически размещает виджет в родительском окне. Упаковщик `pack()` является самым интеллектуальным (и самым непредсказуемым). При использовании этого упаковщика, с помощью

свойства `side`, нужно указать, к какой стороне родительского виджета он должен примыкать. Как правило, этот упаковщик размещает виджеты друг за другом (слева направо или сверху вниз).

При применении этого упаковщика можно указать следующие аргументы:

- `side ("left"/"right"/"top"/"bottom")` - к какой стороне должен примыкать размещаемый виджет.
- `fill (None/"x"/"y"/"both")` - необходимо ли расширять пространство, предоставляемое виджету.
- `expand (True/False)` - необходимо ли расширять сам виджет, чтобы он занял всё предоставляемое ему пространство.
- `in_` - явное указание в какой родительский виджет должен быть помещён.

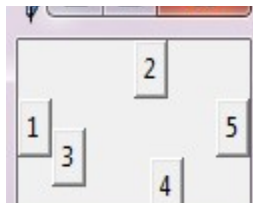
Рассмотрим небольшой пример:

```

from tkinter import *
root=Tk()
button1 = Button(text="1")
button2 = Button(text="2")
button3 = Button(text="3")
button4 = Button(text="4")
button5 = Button(text="5")
button1.pack(side='left')
button2.pack(side='top')
button3.pack(side='left')
button4.pack(side='bottom')
button5.pack(side='right')
root.mainloop()

```

В результате получим:



Как видите, получается действительно непредсказуемо. Поэтому обычно в связке с этим упаковщиком используют Frame, вложенные друг в друга.

## Place

Функция `place` представляет собой простой упаковщик, позволяющий размещать виджет в фиксированном месте с фиксированным размером. Также он позволяет указывать координаты размещения в относительных единицах для реализации "резинового" размещения. При использовании этого упаковщика нам необходимо указывать координаты каждого виджета.

Например:

```

button1 = Button(text="1")
button1.place(x=10, y=10, width=30)

```

Этот упаковщик предоставляет полную свободу в размещении виджетов в окне.

Аргументами упаковщика `place` являются:

- `anchor` ("n", "s", "e", "w", "ne", "nw", "se", "sw" или "center") - какой угол или сторона размещаемого виджета будет указана в аргументах `x/y/relx/rely`. По умолчанию "nw" - левый верхний.
- `bordermode` ("inside", "outside", "ignore") - определяет, в какой степени будут учитываться границы при размещении виджета.
- `in_` - явное указание, в какой родительский виджет должен быть помещён.
- `x` и `y` - абсолютные координаты (в пикселях) размещения виджета.
- `width` и `height` - абсолютные ширина и высота виджета.
- `relx` и `rely` - относительные координаты (от 0.0 до 1.0) размещения виджета.

- relwidth и relheight - относительные ширина и высота виджета.

Относительные и абсолютные координаты (а также ширину и высоту) можно комбинировать. Так, например, relx=0.5, x=-2 означает размещение виджета в двух пикселях слева от центра родительского виджета, relheight=1.0, height=-2 - высота виджета на два пикселя меньше высоты родительского виджета.

## Определение событий и их обработчиков

После того, как мы с вами нарисовали сам графический интерфейс, мы можем приступить к определению событий по нажатию кнопок.

Для начала определим, что такое событие. Событие – это какое-либо действие выполняемое устройствами ввода-вывода при работе приложения, а именно, например, нажатие мышкой кнопки, нажатие кнопки клавиатуры и т.д.

### Command

Мы с вами рассмотрим событие по нажатию кнопки мышью. Для создания такого события используется свойство объекта-кнопки command. Свойству command приравнивается результат работы функции/метода, которые будут вызываться при нажатии соответствующей кнопки. Давайте рассмотрим пример связывания события нажатия кнопки but (отвечает за распечатку списка авторов) с вызовом метода класса Library addAuthor().

Для этого сначала давайте напишем функцию, которая будет вызываться при нажатии кнопки but:

```
def butt():  
    texts = Library.printerAuthor()  
    for t in texts:  
        output.insert("0.0",str(t)+"\n")
```

В результате работы этой функции в переменную text запишется список авторов, содержащихся в библиотеке, и с помощью метода insert данный текст отобразится в поле вывода, которое мы ранее создали (output).

Теперь мы можем связать вызов функции butt и событие по нажатию кнопки but.

```
but = Button(root, text="Посмотреть всех авторов", bg="white", fg="black", font="Arial",  
width=22,height=1,command=butt).grid(row=1, column=1, padx=(10, 10))
```

В результате по нажатию кнопки «Посмотреть всех авторов» мы должны получить:

Посмотреть всех авторов	
Добавить нового автора	
Удалить автора	

(4, 'Джоан', 'Роупинг')  
(3, 'Антон', 'Чехов')  
(2, 'Александр', 'Пушкин')

Теперь давайте разработаем событие для добавления нового автора. Для того, чтобы добавить нового автора, нам необходимо создать дополнительное окно с полями для ввода информации об авторе, которая будет записываться в базу данных. Соответственно, по нажатию кнопки «Добавить нового автора» должно открываться новое окно.

Рассмотрим, как должна выглядеть функция, вызываемая при нажатии на кнопку «Добавить нового автора».

```
#метод создания диалогового окна для добавления автора
def butt3():
    # окно для добавления нового автора
    root1 = Tk()
    root1.title("Добавление нового автора")
    root1.minsize(270, 100)
    #поля для ввода данных о новом авторе
    a = Entry(root1, width=15)
    a.grid(row=1, column=2, padx=(10, 0))
    l = Label(root1, text = "Идентификатор автора")
    l.grid(row=1, column=0, padx=(10, 0))
    a1 = Entry(root1, width=15)
    a1.grid(row=2, column=2, padx=(10, 0))
    l1 = Label(root1, text = "Имя автора")
    l1.grid(row=2, column=0, padx=(10, 0))
    a2 = Entry(root1, width=15)
    a2.grid(row=3, column=2, padx=(10, 0))
    l2 = Label(root1, text = "Фамилия автора")
    l2.grid(row=3, column=0, padx=(10, 0))
    #метод для добавления нового автора
    def button1():
        t = Library.addAuthor(a.get(), a1.get(), a2.get())
        output.insert("0.0", str(t) + "\n")
    #кнопка для добавления нового автора
    butt = tkinter.Button(root1, text = "Добавить", bg="white", fg="black",
width=15,height=1,command=button1)
    butt.grid(row=4, column=0,padx=(10, 10))
    root1.mainloop()
```

Данная функция создаёт новое окно root1, на котором будут располагаться поля ввода и кнопка, инициирующая добавление нового автора.

Для ввода информации об авторе нам необходимо создать три поля ввода, так как мы должны ввести уникальный идентификатор автора, имя автора и фамилию автора. Поля ввода создаются с помощью виджета Entry. Поля ввода a, a1 и a2 привязывается к окну root1 и определяется ширина поля ввода.

Напротив полей ввода для удобства расположим текст, подсказывающий какую информацию необходимо ввести в поле. Для этого используется виджет Label. Label также необходимо привязать к окну, а текст, который должен содержаться в подсказке, задаётся в свойстве text.

Далее пишем функцию button1, реализующую вызов метода класса Library для добавления нового автора. Для вывода результата отработки функции используем метод insert для определенного ранее поля вывода output.

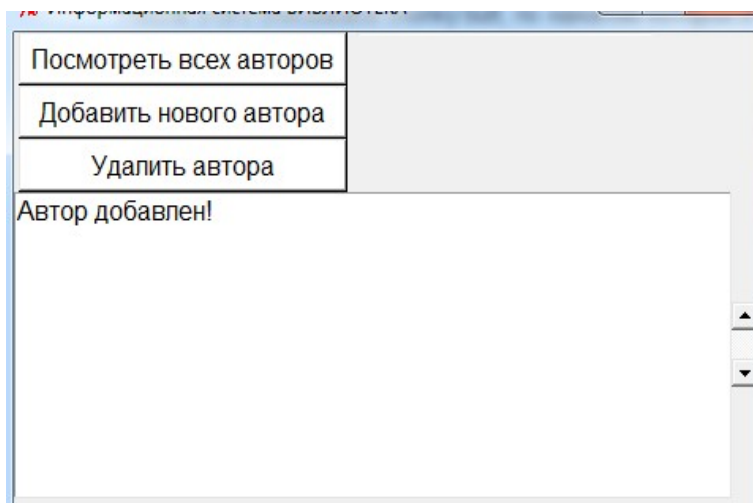
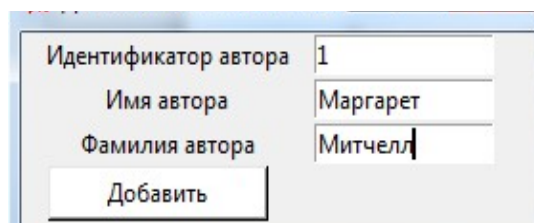
После этого описываем кнопку butt, по нажатию которой и будет вызываться функция button1.

Все виджеты отображаются с помощью метода grid().

После этого давайте привяжем реализованную функцию butt3 к кнопке «Добавление нового автора».

```
but3 = Button(root, text="Добавить нового автора",bg="white", fg="black", font="Arial",
width=22,height=1, command=butt3).grid(row=2, column=1, padx=(1, 1))
```

В результате работы функции добавления нового автора мы должны в итоге получить либо сообщение об ошибке, либо сообщение об успешном добавлении нового автора. Результат должен выглядеть следующим образом:



Аналогичным образом напомним функцию для удаления автора. С единственным отличием для удаления автора мы должны ввести только уникальный идентификатор удаляемого автора.

```
#метод создания диалогового окна для удаления автора
def butt4():
```

```

#окно для удаления автора
root2 = Tk()
root2.title("Удаление автора")
root2.minsize(250, 50)
# поле для ввода идентификатора удаляемого автора
a = Entry(root2, width=17)
a.grid(row=1, column=0, padx=(10, 10))
l = Label(root2, text = "Идентификатор автора")
l.grid(row=1, column=1, padx=(10, 0))
# метод для добавления нового автора
def button2():
    output.edit_reset()
    t = Library.deleteAuthor(a.get())
    output.insert("0.0", str(t) + "\n")
#кнопка для добавления нового автора
butt = tkinter.Button(root2, text = "Удалить", bg="white", fg="black",
width=15,height=1,command=button2)
butt.grid(row=4, column=0,padx=(10, 0))
root2.mainloop()

```

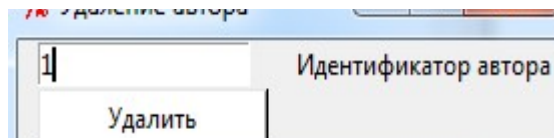
После этого давайте привяжем реализованную функцию butt4 к кнопке «Удаление автора».

```

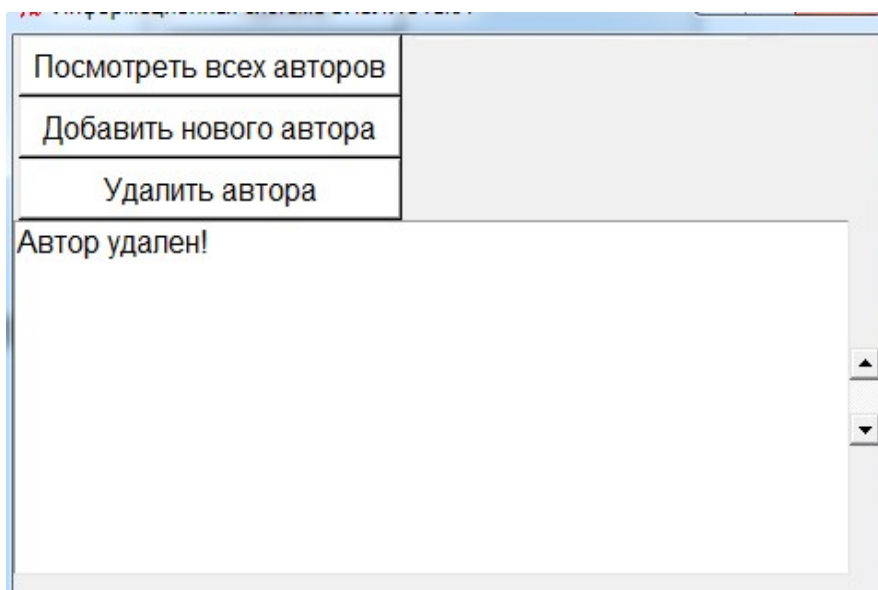
but4 = Button(root, text="Удалить автора", bg="white", fg="black", font="Arial", width=22,height=1,
command=butt4).grid(row=3, column=1, padx=(1, 1))

```

В результате работы функции удаления автора мы должны в итоге получить либо сообщение об ошибке, либо сообщение об успешном удалении автора. Результат должен выглядеть следующим образом:



Удалим автора, которого только что добавили.



## Bind

Метод bind привязывает событие к какому-либо действию (нажатие кнопки мыши, нажатие клавиши на клавиатуре и т.д.). bind принимает три аргумента:

- название события;
- функцию, которая будет вызвана при наступлении события;
- третий аргумент (необязательный) - строка "+" - означает, что эта привязка добавляется к уже существующим. Если третий аргумент опущен или равен пустой строке - привязка замещает все другие привязки данного события к виджету.

Метод bind возвращает идентификатор привязки, который может быть использован в функции unbind.

Обратите внимание, что если bind привязан к окну верхнего уровня, то Tkinter будет обрабатывать события всех виджетов этого окна.

Функция, которая вызывается при наступлении события, должна принимать один аргумент. Это объект класса Event, в котором описано наступившее событие.

Эта функция может возвращать строки "continue" и "break". Если функция возвращает "continue" то Tkinter продолжит обработку других привязок этого события, если "break" - обработка этого события прекращается. Если функция ничего не возвращает (если возвращает None), то обработка событий продолжается (т.е. это эквивалентно возвращению "continue").

Есть три формы названия событий. Самый простой случай - это символ ASCII. Так описываются события нажатия клавиш на клавиатуре:

```
widget.bind("z", callback)
```

Callback вызывается каждый раз, когда будет нажата клавиша "z".

Второй способ длиннее, но позволяет описать больше событий. Название события заключено в угловые скобки. Внутри имеются ноль или более модификаторов, тип события и дополнительная информация (номер нажатой клавиши мыши или символ клавиатуры). Поля разделяются дефисом или пробелом. Пример (привязываем одновременное нажатие Ctrl+Shift+q):

```
widget.bind("<Control-Shift-KeyPress-q>", callback)
```

Третий способ позволяет привязывать виртуальные события - события, которые генерируются самим приложением. Такие события можно создавать самим, а потом привязывать их. Имена таких событий помещаются в двойные угловые скобки: <<Paste>>. Есть некоторое количество уже определённых виртуальных событий:

- Control;
- Alt;
- Shift;
- Lock;
- Extended;
- Prior - PgUp;
- Next - PgDown;
- Button1, B1 - нажата первая (левая) кнопка мыши;
- Button2, B2 - вторая (средняя) кнопка мыши;
- Button3, B3 - третья (правая);
- Button4, B4 - четвёртая;
- Button5, B5 - пятая;



- Mod1, M1, Command;
- Mod2, M2, Option;
- Mod3, M3;
- Mod4, M4;
- Mod5, M5;
- Meta, M4
- Double - двойной щелчок мыши (например, <Double-Button-1>);
- Triple - тройной;
- Quadruple - четверной.

Существует множество стандартных типов событий, к наиболее часто используемым относятся:

- Activate, Deactivate;
- MouseWheel - прокрутка колесом мыши;
- KeyPress, KeyRelease - нажатие и отпускание клавиши на клавиатуре;
- ButtonPress, ButtonRelease, Motion - нажатие, отпускание клавиши мыши, движение мышью;
- Configure - изменение положения или размера окна;
- Map, Unmap - показывание или сокрытие окна (например, в случае сворачивания/разворачивания окна пользователем);
- Visibility;
- Expose - событие генерируется, когда необходимо всё окно или его часть перерисовать;
- Destroy - закрытие окна;
- FocusIn, FocusOut - получение или лишение фокуса;
- Enter, Leave - Enter генерируется когда курсор мыши "входит" в окно, Leave - когда "уходит" из окна.

Также для определения события используются обозначения кнопок мыши и клавиатуры. Например, для обозначения нажатия левой кнопки мыши используется <Button-1> или <1>, а для обозначения правой используется <Button-3> или <3>. <Alt-Motion> - обозначает движение мышью с нажатой на клавиатуре клавишей Alt, а <Key> - нажатие любой клавиши на клавиатуре.

Рассмотрим простой пример:

```
from Tkinter import *
root=Tk()
def leftclick(event):
    print u'Вы нажали левую кнопку мыши'
def rightclick(event):
    print u'Вы нажали правую кнопку мыши'
button1=Button(root, text=u'Нажми')
button1.pack()
button1.bind('<Button-1>', leftclick)
button1.bind('<Button-3>', rightclick)
root.mainloop()
```

Таким образом, за урок мы с вами создали простой и понятный графический интерфейс для приложения, управляющего информацией в библиотеке, и познакомились с основными классами библиотеки tkinter. Весь программный код полностью вы можете посмотреть по ссылке в разделе [Дополнительные материалы](#).

## Домашнее задание

1. Доработать разработанный во время урока интерфейс следующими виджетами:
  - Кнопка «Посмотреть все книги» - по нажатию данной кнопки должен вызываться метод класса library, осуществляющий распечатку всех книг, содержащихся в библиотеке.

- Кнопка «Посмотреть все журналы» - по нажатию данной кнопки должен вызываться метод класса `library`, осуществляющий распечатку всех журналов, содержащихся в библиотеке.
  - Кнопка «Добавить новую книгу» - по нажатию данной кнопки должно создаваться новое диалоговое окно, посредством которого осуществляется добавление новой книги, и вызываться метод класса `library`, осуществляющий добавление новой книги.
  - Кнопка «Добавить новый журнал» - по нажатию данной кнопки должно создаваться новое диалоговое окно, посредством которого осуществляется добавление нового журнала, и вызываться метод класса `library`, осуществляющий добавление нового журнала.
  - Кнопка «Удалить книгу» - по нажатию данной кнопки должно создаваться новое диалоговое окно, посредством которого осуществляется удаление книги по её идентификационному номеру, и вызываться метод класса `library`, осуществляющий удаление книги.
  - Кнопка «Удалить журнал» - по нажатию данной кнопки должно создаваться новое диалоговое окно, посредством которого осуществляется удаление журнала по его идентификационному номеру, и вызываться метод класса `library`, осуществляющий удаление журнала.
2. \*Разработать графический интерфейс, обеспечивающую ввод и редактирование информации об объектах в соответствии с заданной предметной областью. Информация об объектах должна храниться в отдельной базе данных:
- Предметная область - предприятие. Разработать класс `Enterprise`, описывающий работу с предприятием. Разработать класс `People`, описывающий человека, человек характеризуется параметрами: фамилия, имя, отчество, дата рождения, телефон. Разработать класс `Employees` на базе класса `People`, сотрудник характеризуется следующими параметрами: уникальный идентификатор сотрудника, код отдела, заработная плата.
  - Предметная область – университет. Разработать класс `University`, описывающий работы университета. Разработать класс `People`, описывающий человека, человек характеризуется параметрами: фамилия, имя, отчество, дата рождения, телефон. Разработать класс `Students` на базе класса `People`, студент описывается следующими параметрами: уникальный идентификатор студента, ФИО студента, номер группы, признак старосты группы, код специальности.
  - Предметная область – магазин. Разработать класс `Shop`, описывающий работу магазина продуктов. Разработать класс `Products`, продукт описывается следующими параметрами: уникальный идентификатор, название продукта, стоимость, количество. Разработать класс `Fruit_product` на базе класса `Product`, фрукт характеризуется параметрами: страна изготовителя, срок годности.
  - Предметная область – автосалон. Разработать класс `Car_dealership`, описывающий работу автосалона. Разработать класс `Car`, автомобиль описывается следующими параметрами: уникальный идентификатор, марка автомобиля, страна-производитель, год выпуска, объём двигателя, стоимость. Разработать класс `Lorry` на базе класса `Car`, грузовик характеризуется: весовым ограничением перевозки.

## Дополнительные материалы

1. Ссылка на проект Библиотека. <https://drive.google.com/open?id=0B2HCiOJRtFFzUHNkX3EzUFRiNGs>

# Используемая литература

1. <https://www.python.org>
2. <http://pythonicway.com>
3. <https://habrahabr.ru/post/133337>
4. Марк Лутц. Изучаем Python, 4-е издание.