# ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON

Особенности создания оконных приложений в Python (ч.1)


**Крашенинников Роман Сергеевич**
Главный специалист отдела системного администрирования РХТУ им. Д.И. Менделеева, ведущий программист кафедры информационных компьютерных технологий

# ТЕМЫ

- Библиотека **Tkinter**

# ПОЛЕЗНЫЕ РЕСУРСЫ

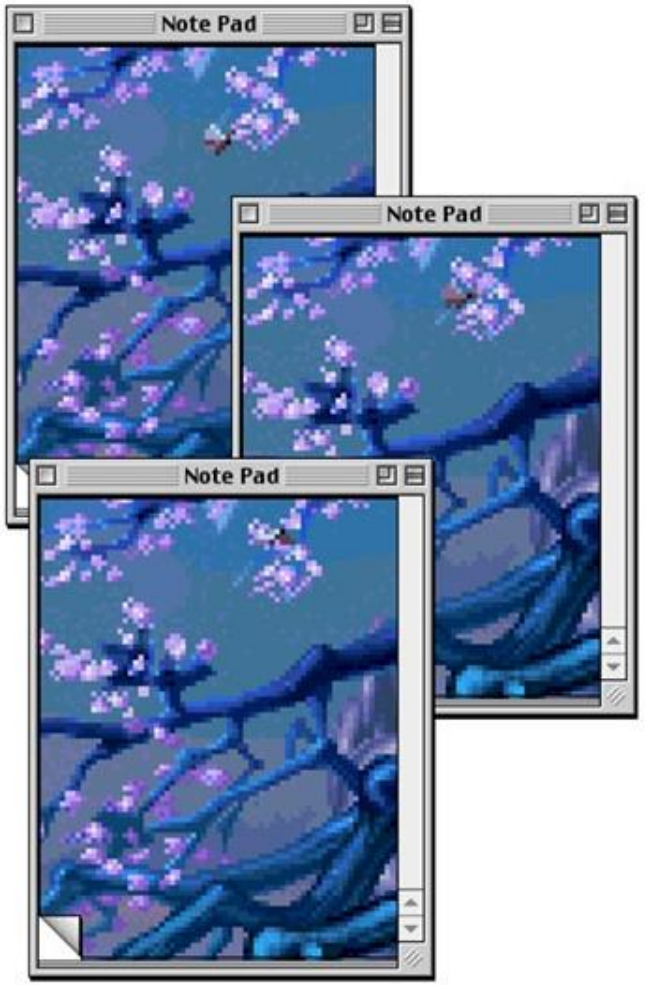
- Курс по Tkinter #1
- Курс по Tkinter #2

# БИБЛИОТЕКА TKINTER

В Python есть довольно много GUI фреймворков (graphical user interface), однако только Tkinter встроен в стандартную библиотеку языка. У Tkinter есть несколько преимуществ. Он кроссплатформенный, поэтому один и тот же код можно использовать на Windows, macOS и Linux.

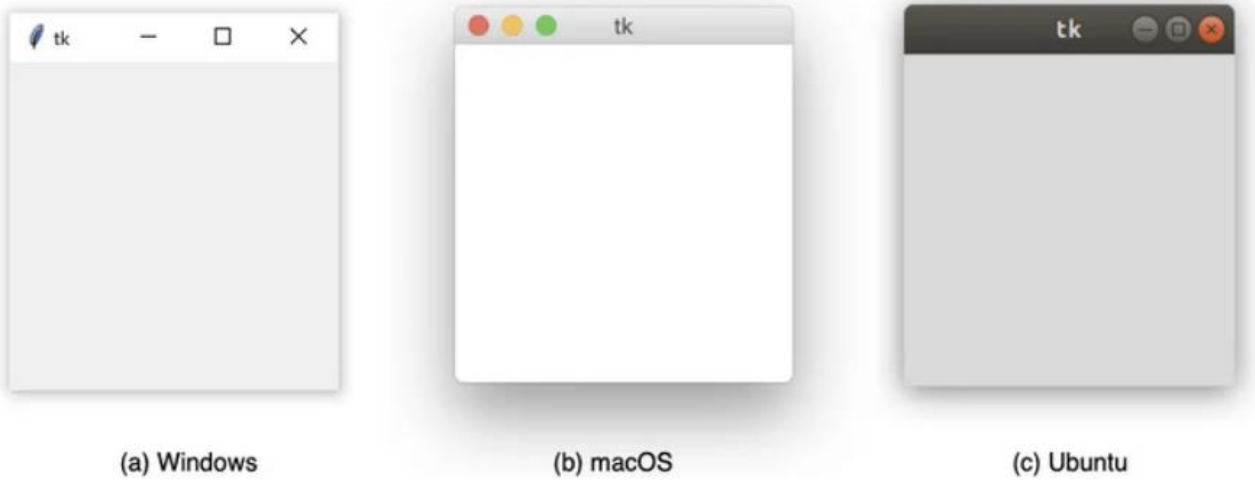
Визуальные элементы отображаются через собственные элементы текущей операционной системы, поэтому приложения, созданные с помощью Tkinter, выглядят так, как будто они принадлежат той платформе, на которой они работают.

Хотя Tkinter является популярным GUI фреймворком на Python, у него есть свои недостатки. Один из них заключается в том, что графические интерфейсы, созданные с использованием Tkinter, выглядят устаревшими. Если вам нужен современный, броский интерфейс, то Tkinter может оказаться не совсем тем, для этого есть PyQt5 который развивается сильнее в данном плане.



# TKINTER СОЗДАНИЕ ОКНА

Главным элементом GUI Tkinter является окно. Окнами называют контейнеры, в которых находятся все GUI элементы. Данные GUI элементы, к числу которых относятся текстовые боксы, ярлыки и кнопки, называются виджетами. Виджеты помещаются внутри окон.




(a) Windows (b) macOS (c) Ubuntu

Пример создания окна

```
import tkinter as tk

#Создание нового окна
window = tk.Tk()

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

# TKINTER ВИДЖЕТЫ

Виджеты являются основой GUI фреймворка Tkinter в Python. Это элементы, через которые пользователи взаимодействуют с программой. В Tkinter каждый виджет определен классом.

| Класс виджета | Описание |
|---|---|
| Label | Используется для отображения текста или вставки изображения на окне приложения. |
| Button | Кнопка, на которой может быть текст, совершает определенные действия при нажатии на нее. |
| Entry | Виджет для ввода одной строчки текста. Эквивалент `<input type="text">` в HTML. |
| Text | Виджет для ввода большого текста. Эквивалент `<textarea>` в HTML. |
| Frame | Прямоугольная область, что используется для группировки виджетов или для добавления расстояния между виджетами. |

# ВИДЖЕТ LABEL

Виджеты Label используется для отображения текста или картинок. Текст на виджете Label, не может редактироваться пользователем. Он только показывается. Настроить цвет текста и фона, а так же ширину, высоту и другое, можно при помощи соответствующих параметров.

Пример создания **Label**

```
root = tk.Tk()
label = tk.Label(
    text="Привет, Tkinter!",
    foreground="white",  # Устанавливает белый текст
    background="black"  # Устанавливает черный фон
)
label.pack()
root.mainloop()
```

# ВИДЖЕТ **BUTTON**

Виджеты Button нужны для создания кликабельных кнопок. Их можно настроить таким образом, чтобы при нажатии вызывалась определенная функция. Для присвоения кнопке функции используется аргумент **command**.

```
def fun(i):
    if i == 1:
        btn1.pack_forget()
        btn2.pack()
    else:
        btn2.pack_forget()
        btn1.pack()


root = tk.Tk()
btn1 = tk.Button(
    command=lambda: fun(1),
    text='Кнопка 1'
)
btn2 = tk.Button(
    command=lambda: fun(2),
    text='Кнопка 2'
)
btn1.pack()
btn2.pack()
root.mainloop()
```

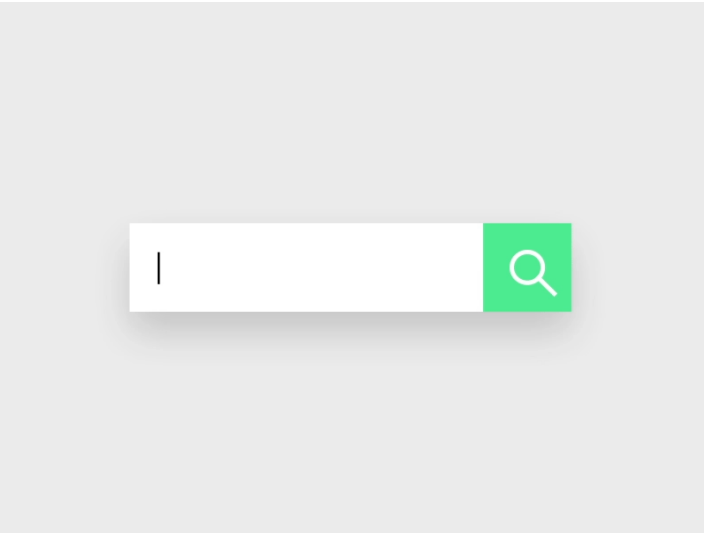
Пример создания **Button**

# ВИДЖЕТ ENTRY

В случаях, когда требуется получить текстовую информацию от пользователя, используется виджет Entry. Он отображает небольшой текстовый бокс, куда пользователь может ввести текст. Для получения текста и присваивания его значения переменной name используется метод .get().

Пример создания **Entry**

```
root = tk.Tk()
entry = tk.Entry(fg="yellow", bg="blue")
entry.pack()
btn = tk.Button(command=lambda: print(entry.get()))
btn.pack()
root.mainloop()
```



# ВИДЖЕТ **TEXT**

Виджеты Text используются для ввода текста, как и виджеты Entry. Разница в том, что Text может содержать несколько строчек текста. С виджетом Text пользователь может вводить целые параграфы или страницы текста. Для использования метода get требуется передать начальный и конечный индексы.

Пример создания **Text**

```
root = tk.Tk()
text = tk.Text(fg="yellow", bg="blue")
text.pack()
btn = tk.Button(command=lambda: print(text.get("1.0", tk.END)))
btn.pack()
root.mainloop()
```

# ВИДЖЕТ **FRAME**

Виджет **Frame** необходим для группирования элементов в окне. **Frame** играет важную роль для организации макета виджетов. Рамки могут менять свой стиль с помощью атрибута relief, который создает границу вокруг рамки. Для применения эффекта рамки нужно установить значение атрибута borderwidth выше, чем 1. Данный атрибут корректирует ширину рамки в пикселях.

Пример создания **Frame**

```
window = tk.Tk()

frame_a = tk.Frame()
frame_b = tk.Frame()

label_a = tk.Label(master=frame_a, text="I'm in Frame A")
label_a.pack()

label_b = tk.Label(master=frame_b, text="I'm in Frame B")
label_b.pack()

frame_a.pack()
frame_b.pack()

window.mainloop()
```

`tk.FLAT`: Никакого эффекта рамки (по умолчанию);

`tk.SUNKEN`: Создается эффект углубления элемента;

`tk.RAISED`: Создается эффект выпуклости элемента;

`tk.GROOVE`: Создается эффект врезанной в текстуру рамки, своего рода выемки;

`tk.RIDGE`: Создается эффект выпуклой выемки.

# МЕНЕДЖЕР ГЕОМЕТРИИ PACK

Менеджер .pack() используется для размещения виджетов на рамку или на окне приложения в определенном порядке использует алгоритм упаковки. Для заданного виджета у алгоритма упаковки есть два основных этапа:

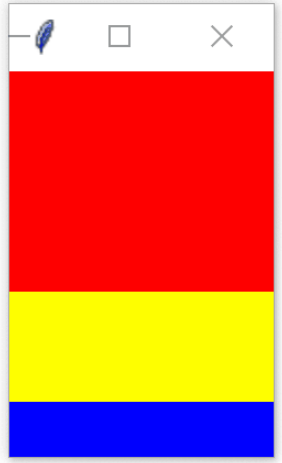
- Расчет прямоугольной области, называемой участком. Размер области подходит для вмещения виджета, а оставшаяся ширина (или высота) окна заполняется пустым пространством;
- Центрирование виджета в участке, если не указано другое местоположение в окне.

# МЕНЕДЖЕР ГЕОМЕТРИИ PACK

Менеджер .pack() принимает некоторые ключевые аргументы для более точной настройки размещения виджетов. К примеру, можно установить ключевой аргумент fill для уточнения в какое направление рамки будут пополняться. Доступная опция tk.X для горизонтального направления, tk.Y для вертикального направления и tk.BOTH для обоих. Далее представлен код для горизонтального заполнения окна тремя рамками. Аргумент side от метода .pack() уточняет, на какую сторону окна виджет должен размещаться.

# МЕНЕДЖЕР ГЕОМЕТРИИ PACK

```
window = tk.Tk()
frame1 = tk.Frame(master=window, height=100, bg="red")
frame1.pack(fill=tk.X)
frame2 = tk.Frame(master=window, height=50, bg="yellow")
frame2.pack(fill=tk.X)
frame3 = tk.Frame(master=window, height=25, bg="blue")
frame3.pack(fill=tk.X)
window.mainloop()
```

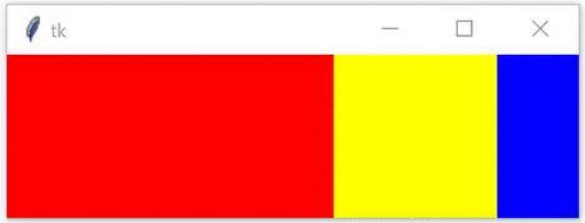


# МЕНЕДЖЕР ГЕОМЕТРИИ PACK

```
window = tk.Tk()
frame1 = tk.Frame(master=window, width=200, height=100, bg="red")
frame1.pack(fill=tk.Y, side=tk.LEFT)
frame2 = tk.Frame(master=window, width=100, bg="yellow")
frame2.pack(fill=tk.Y, side=tk.LEFT)
frame3 = tk.Frame(master=window, width=50, bg="blue")
frame3.pack(fill=tk.Y, side=tk.LEFT)
window.mainloop()
```

# МЕНЕДЖЕР ГЕОМЕТРИИ PACK

```
window = tk.Tk()
frame1 = tk.Frame(master=window, width=200, height=100, bg="red")
frame1.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)
frame2 = tk.Frame(master=window, width=100, bg="yellow")
frame2.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)
frame3 = tk.Frame(master=window, width=50, bg="blue")
frame3.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)

window.mainloop()
```



# МЕНЕДЖЕР ГЕОМЕТРИИ PLACE

Для управления точным местом расположения виджета в окне или в рамке используется менеджер .place(). Вы должны указать два ключевых аргумента x и y, которые определяют координаты x и y для верхнего левого угла виджета. Аргументы x и y измеряются в пикселях.

```
window = tk.Tk()
frame = tk.Frame(master=window, width=150, height=150)
frame.pack()
label1 = tk.Label(master=frame, text="I'm at (0, 0)", bg="red")
label1.place(x=0, y=0)
label2 = tk.Label(master=frame, text="I'm at (75, 75)", bg="yellow")
label2.place(x=75, y=75)
window.mainloop()
```

# МЕНЕДЖЕР ГЕОМЕТРИИ GRID

Менеджер .grid() работает путем разделения окна или рамки на строки и столбцы (на сетку). Вы указываете местоположение виджета, вызывая метод .grid() и передавая индексы row и column (строки и столбца) в ключевые аргументы строки и столбца соответственно.

```python
import tkinter as tk

window = tk.Tk()

for i in range(5):
    for j in range(5):
        frame = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1
        )
        frame.grid(row=i, column=j, padx=5, pady=5)
        label = tk.Label(master=frame, text=f"Строка {i}\nСтолбец {j}")
        label.pack()

window.mainloop()
```

# МЕНЕДЖЕР ГЕОМЕТРИИ GRID

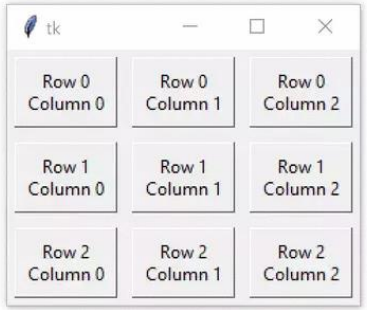
Можно настроить так, чтобы строки и столбы сетки будут реагировать на изменение размера окна с помощью метода .columnconfigure() и .rowconfigure() для объекта окна приложения. Помните, что сетка привязана к окну, даже если вы вызываете метод .grid() для каждой рамки. Метод .columnconfigure() и .rowconfigure() принимают три важных аргумента:

- Индекс столбца или строки сетки, которого нужно настроить (или список индексов для настройки нескольких строк или столбцов одновременно);
- Ключевой аргумент под названием weight, который определяет, как столбец или строка должны реагировать на изменение размера окна относительно других столбцов и строк;
- Ключевой аргумент под названием minsize, который устанавливает минимальный размер высоты строки или ширины столбца в пикселях.

# МЕНЕДЖЕР ГЕОМЕТРИИ GRID

Аргумент weight по умолчанию равен 0 . Это означает, что столбец или строка не расширяются при изменении размера окна. Если каждому столбцу и строке присвоен weight=1, то все они будут меняться одинаково. Если вес (weight) одного столбца 1, а у другого 2, то второй столбец расширяется вдвое быстрее первого.

```
window = tk.Tk()

for i in range(5):
    window.columnconfigure(i, weight=1, minsize=75)
    window.rowconfigure(i, weight=1, minsize=50)
    for j in range(5):
        frame = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1
        )
        frame.grid(row=i, column=j, padx=5, pady=5)
        label = tk.Label(master=frame, text=f"Строка {i}\nСтолбец {j}")
        label.pack()

window.mainloop()
```



# ОБРАБОТЧИК СОБЫТИЙ BIND

Для вызова обработчика событий во время возникновения события, связанного с виджетом, используется метод .bind(). Обработчик событий напрямую связан с событием. Обработчик события связан с виджетом, для которого вызывается метод .bind(). Когда вызывается обработчик события, объект события передается в функцию обработчика события.

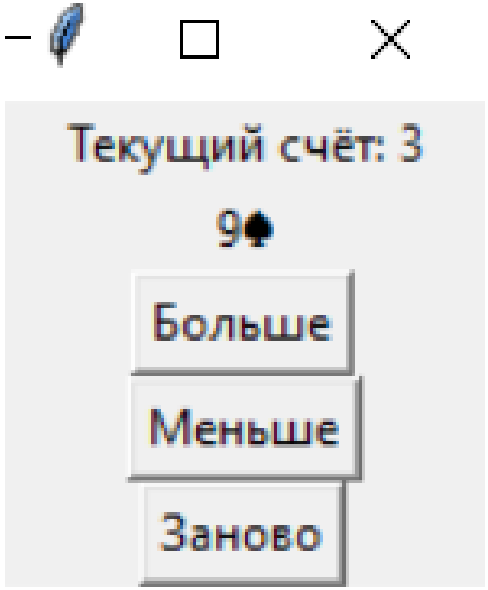
```
window = tk.Tk()

def handle_click(event):
    print(event)
    print("Нажата кнопка!")


label = tk.Label(text="Кликни дважды!")

#Обработка двойного нажатия левой кнопки мыши
label.bind("<Double-Button-1>", handle_click)
label.pack()
window.mainloop()
```

# ЗАДАНИЕ ДЛЯ САМОКОНТРОЛЯ

При помощи библиотеки **Tkinter** реализовать игру:

**Игра больше/меньше –** из колоды карт (52 карты) тянется карта и игрок пытается угадать будет ли следующая карта по значению больше или меньше. Если игрок угадывает, тянется следующая карта, если нет, игра заканчивается. Если колода кончилась, игрок победил и игра заканчивается. **Специальное правило** – считается, что туз меньше двойки, но больше всех остальных карт.

Примерный интерфейс



# СПАСИБО ЗА ВНИМАНИЕ!