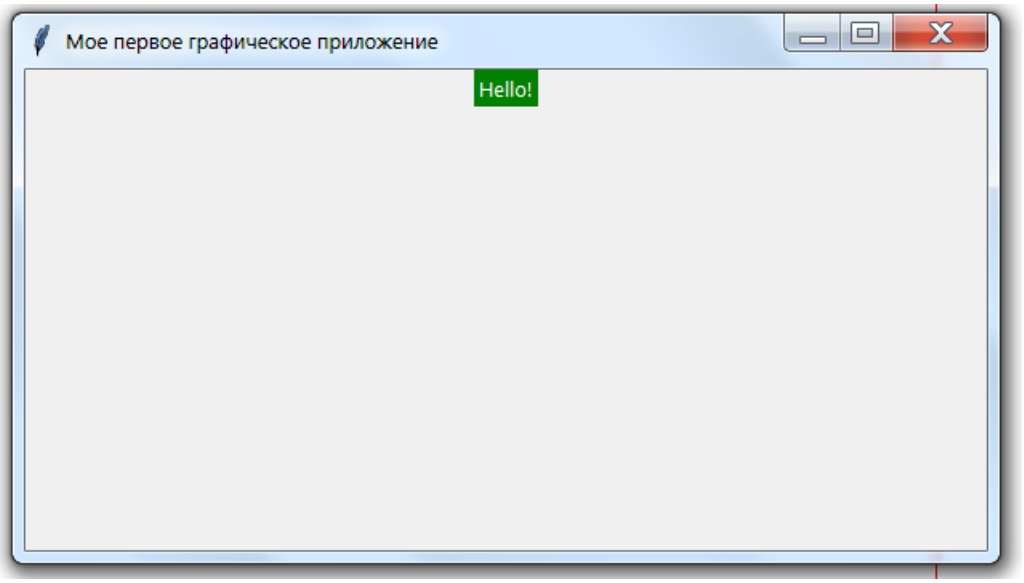
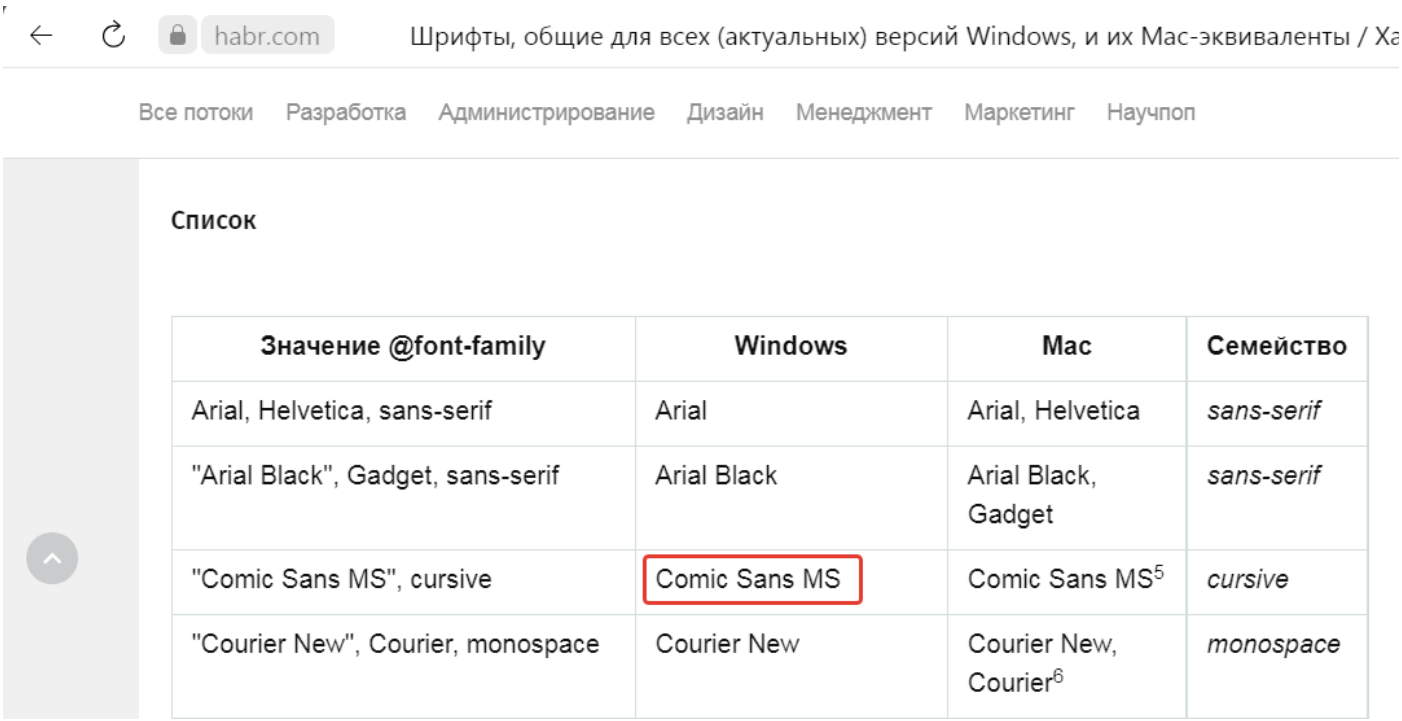


Далее мы можем изменить шрифт, размер шрифта, тип шрифта.

Название основных шрифтов можно посмотреть в интернете: **Шрифты для Windows.**



К примеру, возьмем шрифт: **Comic Sans MS.**

У шрифтов бывают некоторые свойства:

Насыщенность шрифта: свойство **font-weight** взято из HTML

normal - Значение по умолчанию, устанавливает нормальную насыщенность шрифта. Эквивалентно значению насыщенности, равной 400.

bold - Делает шрифт текста полужирным. Эквивалентно значению насыщенности, равной 700.

Начертание шрифта: свойство font-style взято из HTML

normal - Значение по умолчанию, устанавливает для текста обычное начертание шрифта.

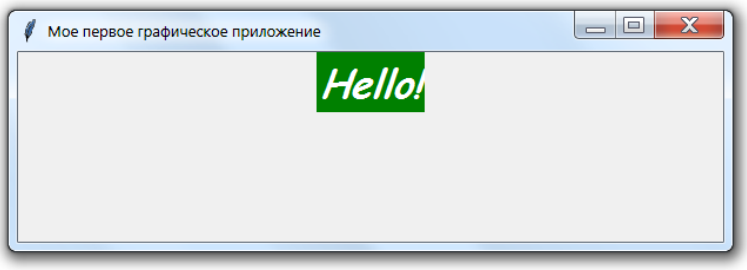
italic - Выделяет текст курсивом.

Используем некоторые свойства для настройки шрифта нашего виджета label:

```
label_1 = tk.Label(win, text = 'Hello!',
                   bg = 'green',
                   fg = 'white',
                   font = ('Comic Sans MS', 20, 'bold', 'italic')
                   )
```

Тестируем:

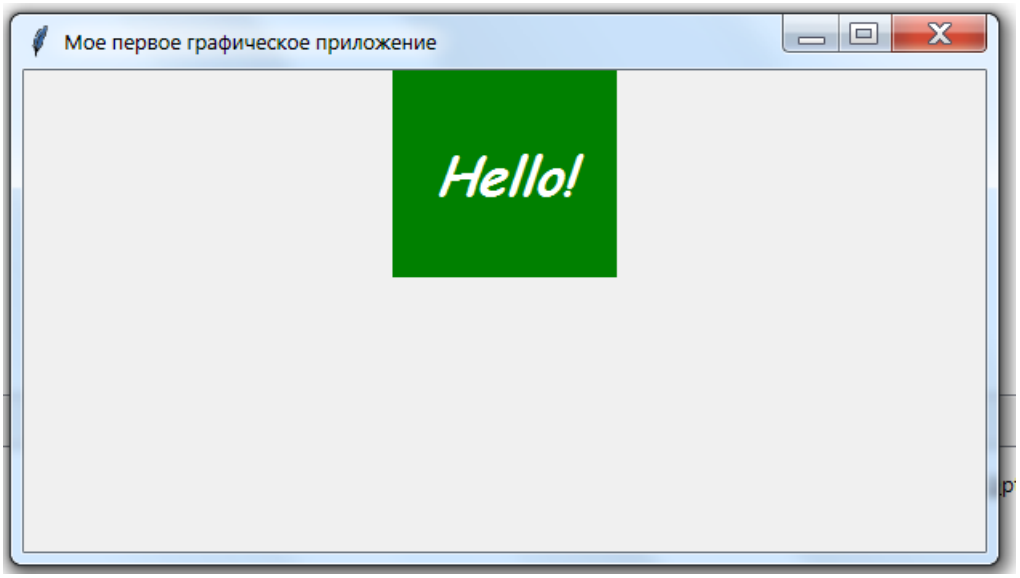
```
 8 label_1 = tk.Label(win, text = 'Hello!',
 9                    bg = 'green',
10                    fg = 'white',
11                    font = ('Comic Sans MS', 20, 'bold', 'italic')
12                    )
13 label_1.pack()
14
15 win.mainloop()
```



Внутри виджета мы можем добавить отступы для текста по x и по y:

```
label_1 = tk.Label(win, text = 'Hello!',
                   bg = 'green',
```

```
fg = 'white',
font = ('Comic Sans MS', 20, 'bold', 'italic'),
padx = 25,
pady = 40
)
```



Теперь давайте закомментируем строки: padx, pady

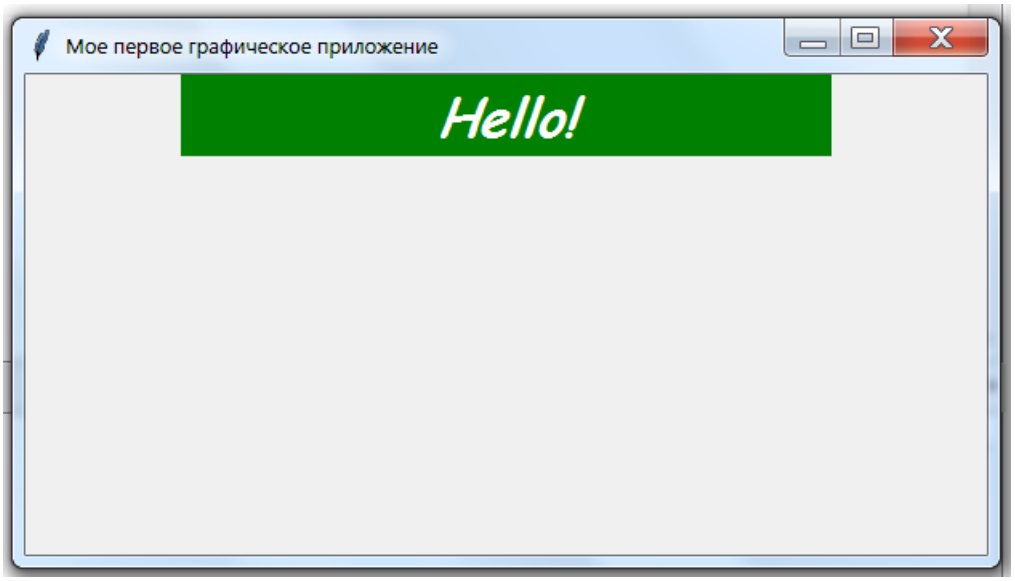
```
 8  label_1 = tk.Label(win, text = 'Hello!',
 9                     bg = 'green',
10                     fg = 'white',
11                     font = ('Comic Sans MS', 20, 'bold', 'italic'),
12                     #padx = 25,
13                     #pady = 40,
14
15                     )
```

Мы можем изменить размеры самого виджета:

Изменим ширину: width

```
label_1 = tk.Label(win, text = 'Hello!',
        bg = 'green',
        fg = 'white',
        font = ('Comic Sans MS', 20, 'bold', 'italic'),
        #padx = 25,
        #pady = 40,
        width = 20
        )
```

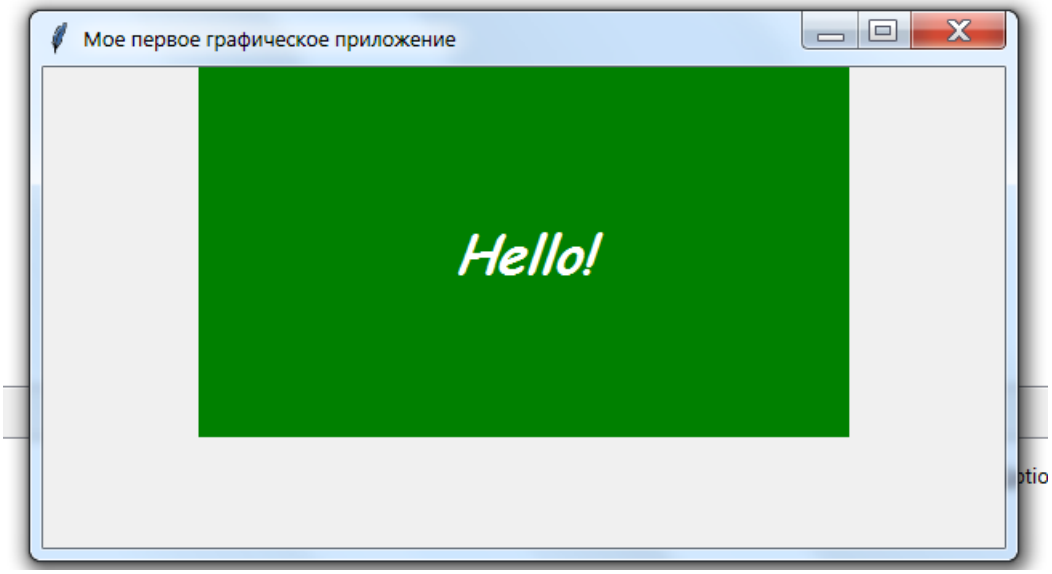
Тестируем:



Изменим высоту виджета:

```
label_1 = tk.Label(win, text = 'Hello!',
                   bg = 'green',
                   fg = 'white',
                   font = ('Comic Sans MS', 20, 'bold', 'italic'),
                   #padx = 25,
                   #pady = 40,
                   width = 20,
                   height = 5
                   )
```

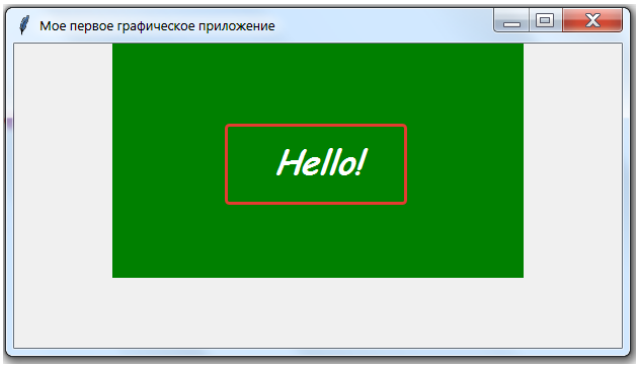
Тестируем:



Важно запомнить! Значения для ширины и высоты самого виджета label это не пиксели (не точки на экране) – это выделение места под символы текста!

Сколько символов уместится внутри виджета по ширине или высоте!

Можно обратить внимание еще на то, что текст сейчас при запуске нашей программы всегда расположен посередине внутри нашего виджета.



Мы можем повлиять на расположение текста внутри окна виджета.

Для этого существует свойство: **anchor** (якорь, привязка).

Внутри привязки мы должны будем указать сторону света для привязки текста внутри виджета **label**.

Какие варианты сторон света мы можем указать:

n, ne, e, se, s, sw, w, nw, или center

n – north север

e – east восток

s – south юг (салф произносится – странно, но факт)

w – west запад

ne – северо-восток

se – юго-восток

sw – юго-запад

nw – северо-запад

Для понимания позиции по сторонам света составим таблицу:

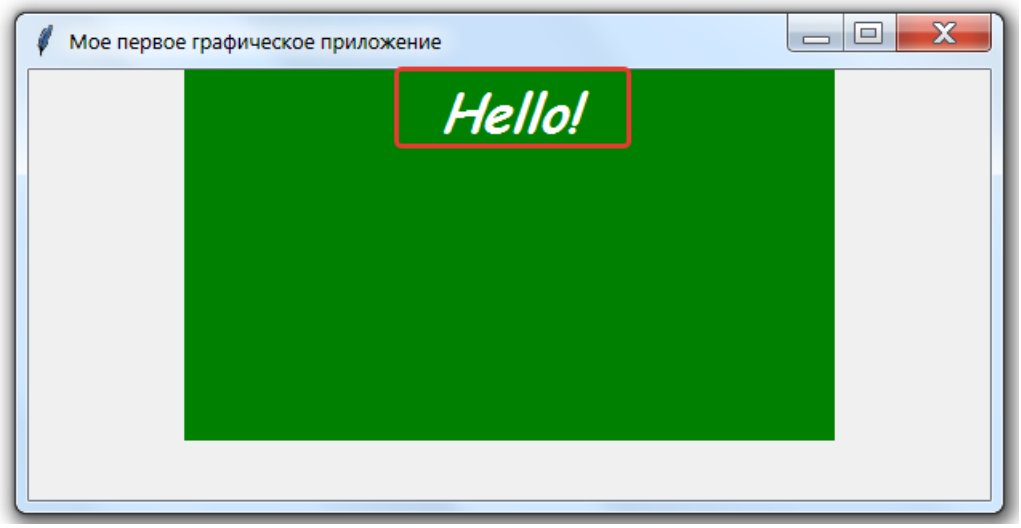
северо-запад	север	северо-восток
запад	центр	восток
юго-запад	юг	юго-восток

nw	n	ne
w	center	e
sw	s	se

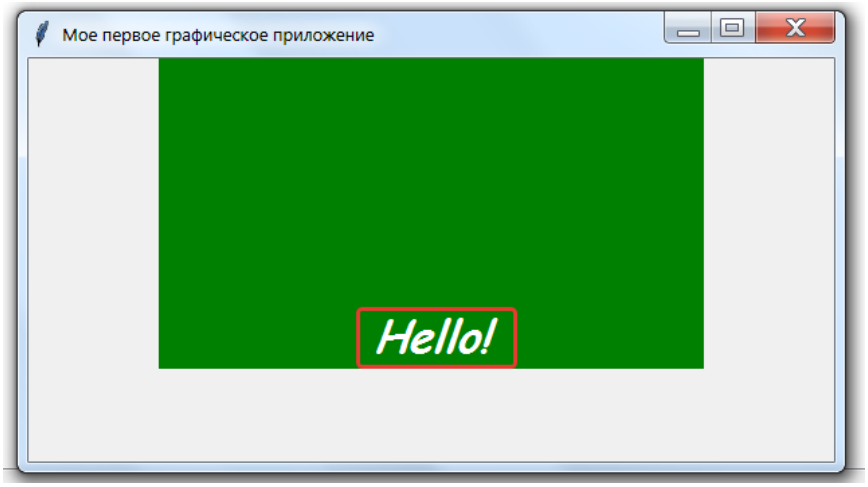
Теперь посмотрим на практике, как меняется положение текста внутри виджета label:

```
label_1 = tk.Label(win, text = 'Hello!',
                   bg = 'green',
                   fg = 'white',
                   font = ('Comic Sans MS', 20, 'bold', 'italic'),
                   #padx = 25,
                   #pady = 40,
                   width = 20,
                   height = 5,
                   anchor = 'n'
                   )
```

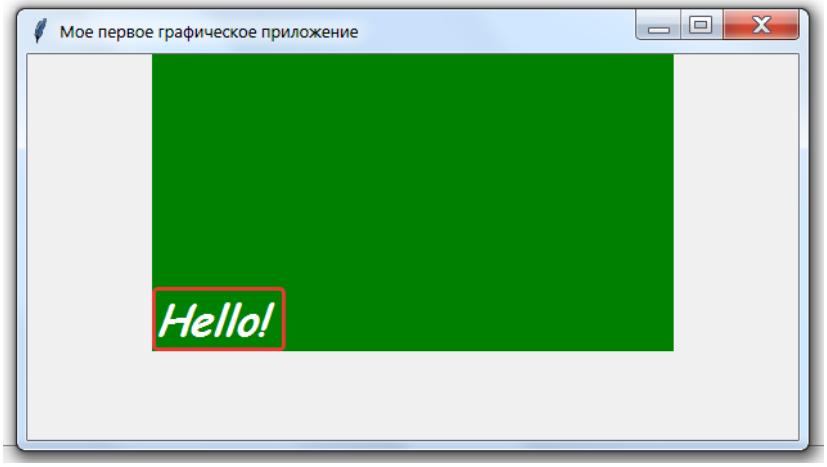
Тестируем:



Если: anchor = 's'



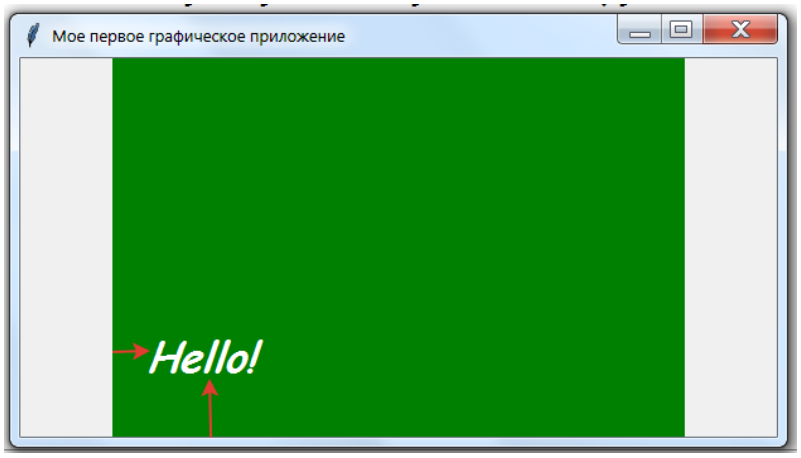
Если: anchor = 'sw'



Здесь понятно, что значение **center** для свойства **anchor** можно вводить, а можно не использовать вообще свойство **anchor**. Если положение текста нужно ровно по центру виджета, то можно и вообще не использовать настройку свойства **anchor** в программе.

Оставим пока в нижнем левом углу текст внутри виджета **label**.

Снимем комментарий с отступов: **padx, pady**.

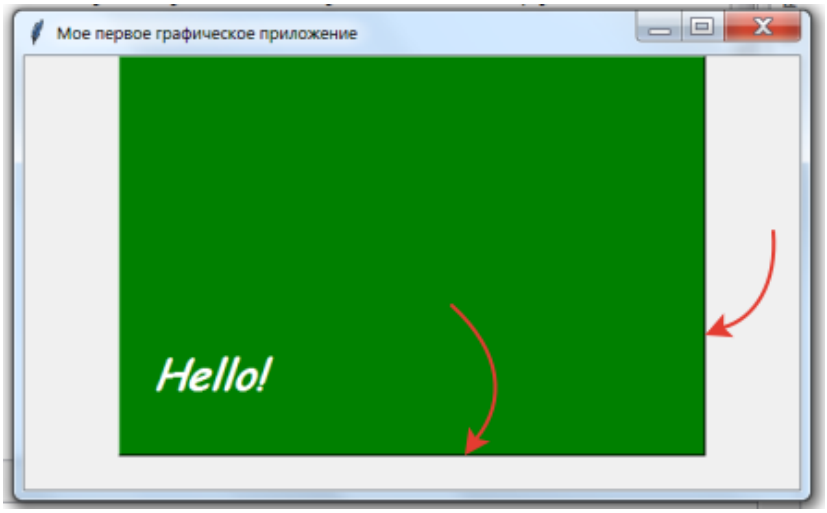


Когда мы включили отступы, то видим, как отодвинулся текст из угла.

Рассмотрим еще одно свойство класса Label(): **relief (рельеф)**

```
width = 20,
height = 5,
anchor = 'sw',
relief = tk.RAISED
)
```

Тестируем:



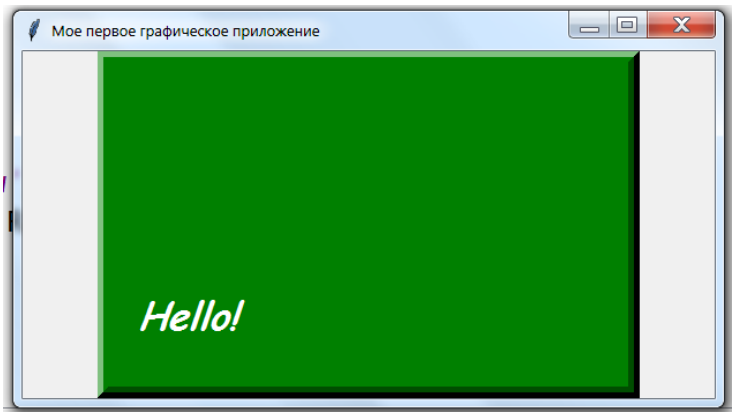
Появляется у виджета по краям небольшие выступы. Виджет как будто стал объемным (произошло выдавливание виджета)

По умолчанию высота выдавливания установлена в **2px**.

Мы можем изменить настройки выдавливания с помощью свойства: **bd (border depth)**

```
anchor = 'sw',
relief = tk.RAISED,
bd = 10
)
```

Тестируем:

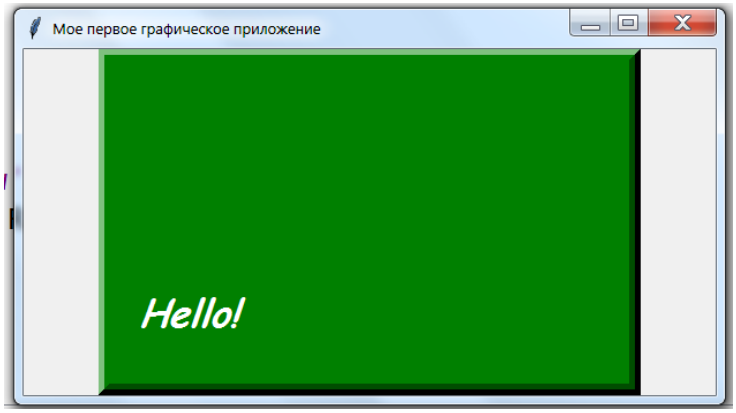


В завершении знакомства со свойствами класса **Label()** рассмотрим работу свойства: **justify**

Justify – выравнивание многострочного текста.

```
anchor = 'sw',
relief = tk.RAISED,
bd = 10,
justify = tk.CENTER
)
```

Если запустить программу сейчас, то мы не увидим изменений по выравниванию текста.



Во-первых, у нас только одна строка текста.

Во-вторых, значение **tk.CENTER** в свойстве **justify** – это значение по умолчанию.

Добавим к нашей строке – **Hello!** ещё пару строк.

Чтобы добавить несколько строк мы будем использовать специальный синтаксис с тремя кавычками.

Переходим к строке кода, где у нас передается строка **'Hello!'** в параметре text класса Label():

```
label_1 = tk.Label(win, text = 'Hello!',
        bg = 'green',
        fg = 'white',
        font = ('Comic Sans MS', 20, 'bold', 'italic'),
```

Вместо одинарных кавычек установим тройные кавычки:

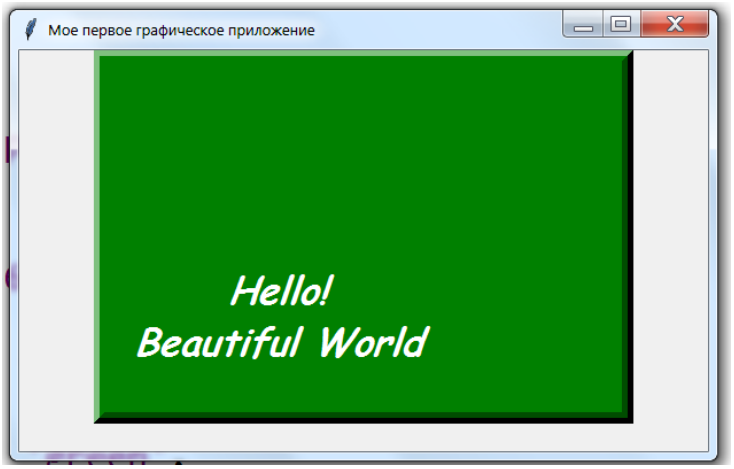
```
label_1 = tk.Label(win, text = '''Hello!''',
        bg = 'green',
        fg = 'white',
```

```
font = ('Comic Sans MS', 20, 'bold', 'italic'),
```

Теперь мы можем смело без опасения нарушить структуру и синтаксис языка Python написать внутри тройных кавычек текст в несколько строк.

```
label_1 = tk.Label(win, text = '''Hello!
Beautiful World''',
        bg = 'green',
        fg = 'white',
        font = ('Comic Sans MS', 20, 'bold', 'italic'),
```

Тестируем:



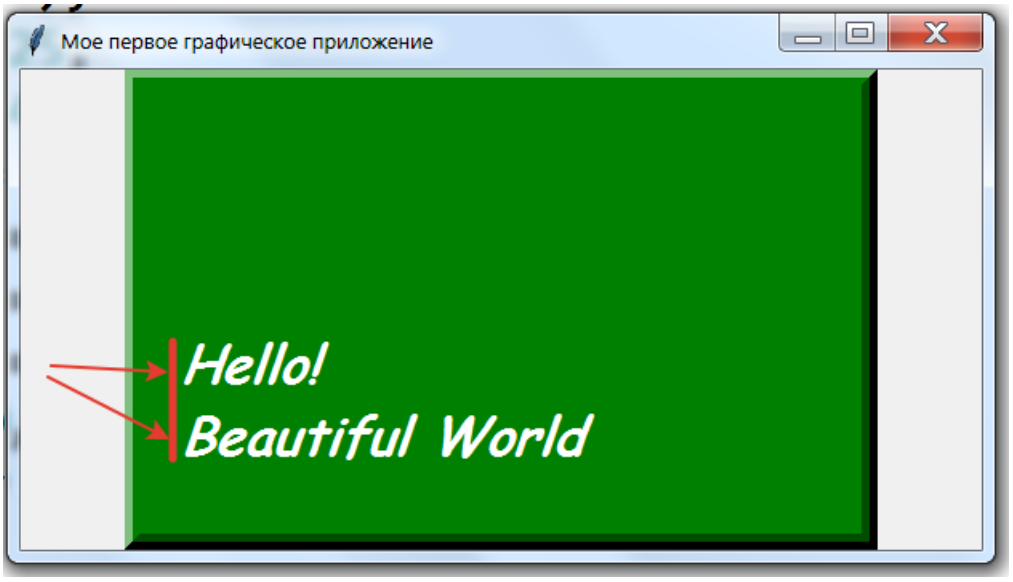
Видим, что сейчас у нас строки выравнены по центру только относительно друг друга.

Изменим значение свойства **justify**:

К примеру, настроим выравнивание по левому краю.

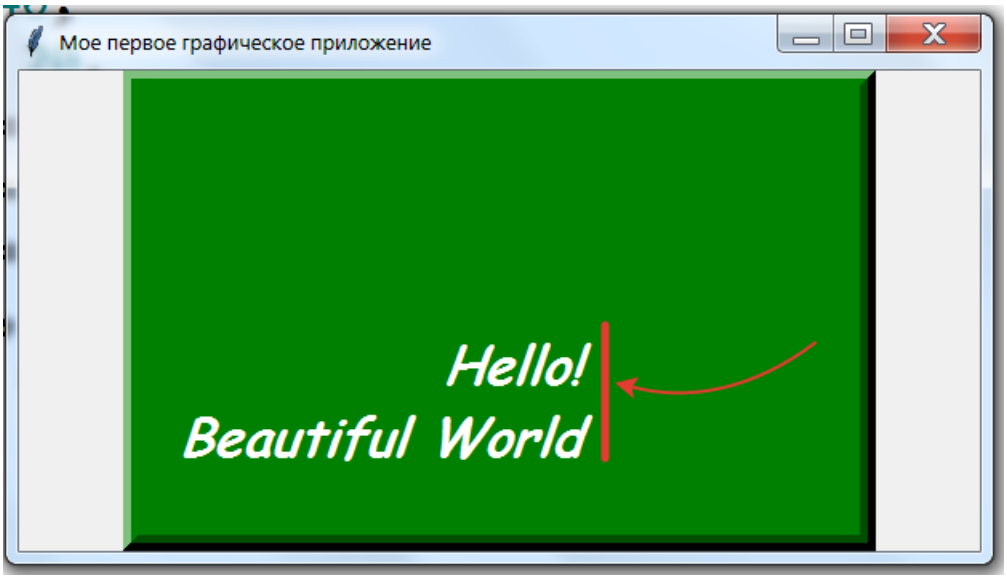
```
relief = tk.RAISED,
        bd = 10,
        justify = tk.LEFT
        )
```

Тестируем:



Если установить вместо LEFT → RIGHT?

justify = tk.RIGHT



Вот так работает свойство **justify**.

Свойство **justify** прижимает строчки текста к какой-то границе или размещает строчки текста по центру.

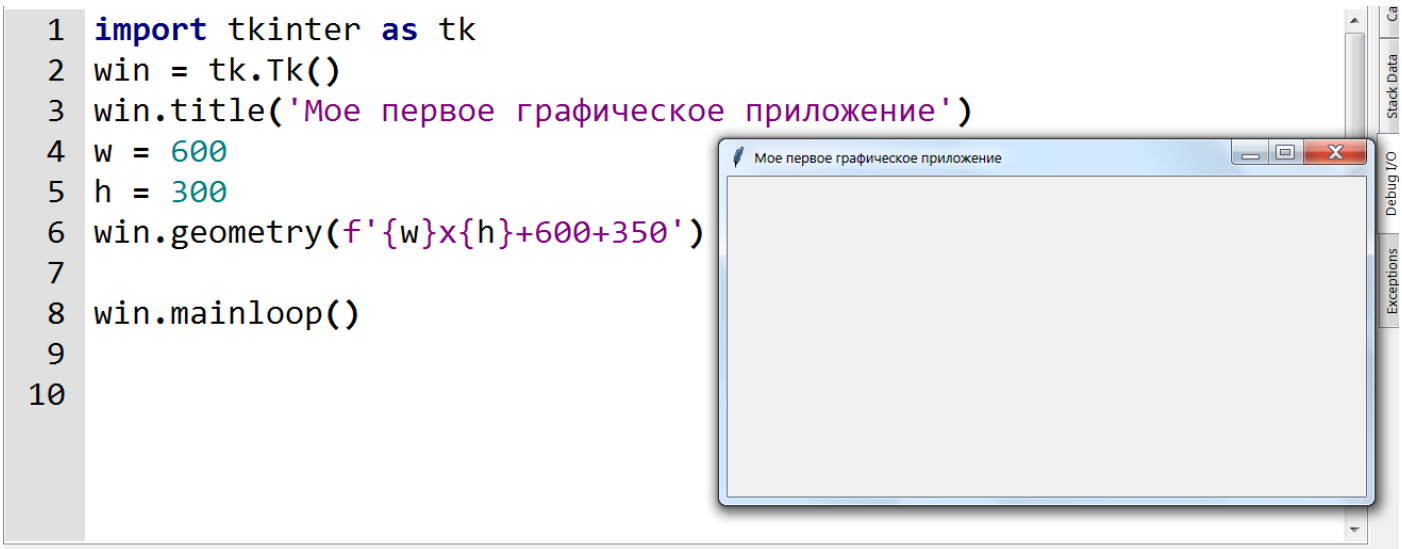
Ниже представлен весь код нашего урока:

```
import tkinter as tk
win = tk.Tk()
win.title('Мое первое графическое приложение')
w = 600
h = 300
win.geometry(f'{w}x{h}+600+350')

label_1 = tk.Label(win, text = '''Hello!
Beautiful World''',
            bg = 'green',
            fg = 'white',
            font = ('Comic Sans MS', 20, 'bold', 'italic'),
            padx = 25,
            pady = 40,
            width = 20,
            height = 5,
            anchor = 'sw',
            relief = tk.RAISED,
            bd = 10,
            justify = tk.RIGHT
            )
label_1.pack()

win.mainloop()
```

Виджет Button

Исходный код для знакомства с виджетом – **Button**:

```
import tkinter as tk
win = tk.Tk()
win.title('Мое первое графическое приложение')
w = 600
h = 300
win.geometry(f'{w}x{h}+600+350')

win.mainloop()
```



Перед строкой – **win.mainloop()** создаем виджет – **button**. Создаем имя переменной для кнопки: **btn1**. В этой переменной мы вызываем у модуля - tk класс Button(). В класс **Button()** мы передаем параметр – **win** (где мы располагаем нашу кнопку) и параметр – **text** (что будет написано на нашей кнопке).

```
w = 600
h = 300
win.geometry(f'{w}x{h}+600+350')
btn1 = tk.Button(win, text = 'Hello')
```

Запускаем программу и не видим кнопки. Это правильно.

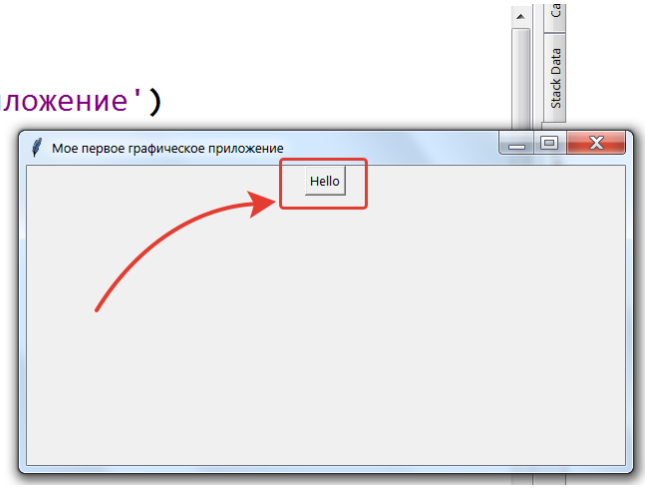
Мы не поместили нашу кнопку на экране программы.

Это выполняет метод pack().

```
btn1 = tk.Button(win, text = 'Hello')
btn1.pack()
```

Запускаем программу. Кнопка появилась.

```
1  import tkinter as tk
2  win = tk.Tk()
3  win.title('Мое первое графическое приложение')
4  w = 600
5  h = 300
6  win.geometry(f'{w}x{h}+600+350')
7  btn1 = tk.Button(win, text = 'Hello')
8  btn1.pack()
9
10 win.mainloop()
11
12
```



Мы можем нажимать на кнопку, но при нажатии на кнопку ничего пока не происходит.

А ведь события, которые происходят при нажатии на кнопку это, и есть самый главный функционал кнопки.

Другими словами, кнопка при нажатии должна что-то выполнять.

Кнопка ничего не выполняет, потому что мы не сделали обработчик событий.

Для создания обработчика мы добавляем еще один параметр в классе Button(): **command** (команда, приказ).

```
win.geometry(f'{w}x{h}+600+350')
btn1 = tk.Button(win, text = 'Hello',
            command = функция    # мы должны сюда передать название функции (обработчика)
            )
btn1.pack()
```

Функцию, которую мы должны передать в параметр – command мы должны написать (создать) сами.

Вспоминаем, что описание функции мы создаем в программе по коду выше места ее использования (вызова).

Поэтому переходим в начало кода и пишем функцию для параметра – **command**.

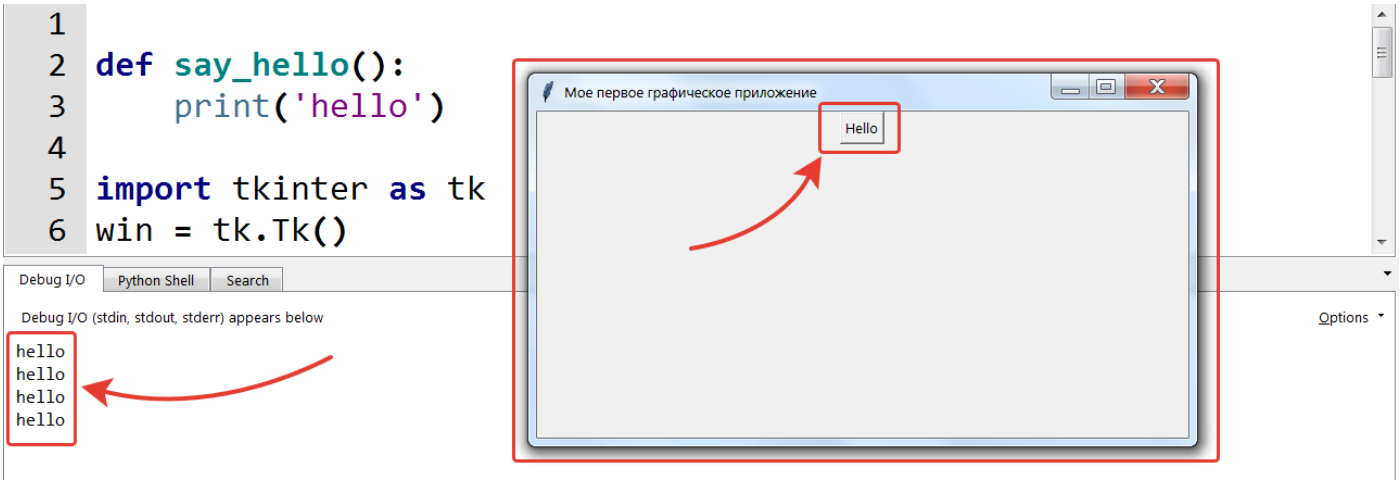
```
def say_hello():
   print('hello')

import tkinter as tk
win = tk.Tk()
```

Теперь добавим имя функции в параметр – **command**:

```
win.geometry(f'{w}x{h}+600+350')
btn1 = tk.Button(win, text = 'Hello',
            command = say_hello
            )
```

Важно! Перед запуском программы нам нужно запомнить, что мы в параметр – **command** передаем только имя функции, без ее вызова (без скобок!). Параметр – **command** сам вызовет нашу функцию – **say_hello()**!

Запускаем программу. Нажимаем на кнопку и смотрим, что отображается в консоли нашей среды разработки – WING.



Видим, что каждое нажатие на кнопку выполняет команду: **print('hello').**

Это самая простая кнопка, при нажатии которой результат ее работы мы видим в консоли среды разработки WING.

Нам нужно научиться программировать такую кнопку, при нажатии на которую будут выполняться действия с другими виджетами программы или действия с главным окном программы.

Давайте создадим такую кнопку: Копируем код первой кнопки **btn1**, вставляем код ниже и переименовываем кнопку в **btn2** и вводим новое имя функции **add_label,** переписываем текст внутри параметра кнопки **btn2**.

```
win.geometry(f'{w}x{h}+600+350')

btn1 = tk.Button(win, text = 'Hello',
            command = say_hello
            )

btn2 = tk.Button(win, text = 'Add Label',
            command = add_label
            )
```

Теперь переходим вверх программы, где у нас описывается функция **say_hello()** и ниже описания этой функции описываем функцию **add_label()**.

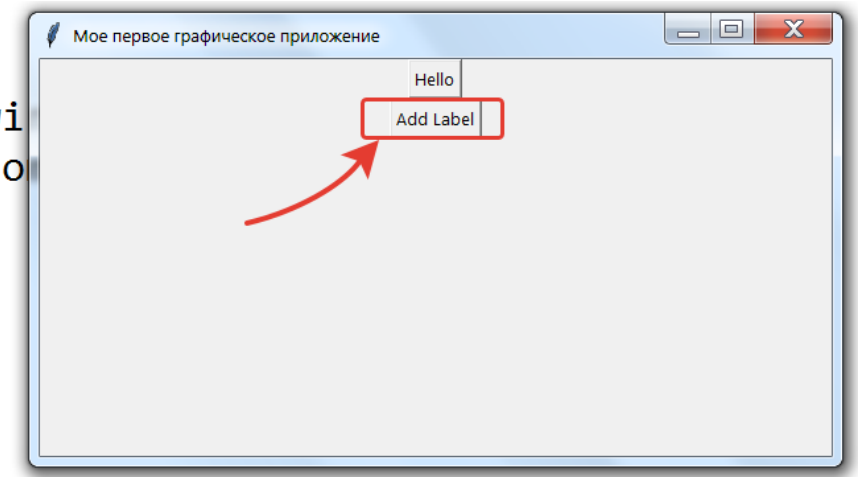
```
def say_hello():
    print('hello')

def add_label():
    label = tk.Label(win, text = 'New Label')
    label.pack()

import tkinter as tk
```

Теперь нужно отобразить нашу вторую кнопку **btn2** на главном экране программы.

```
btn1.pack()
btn2.pack()
```

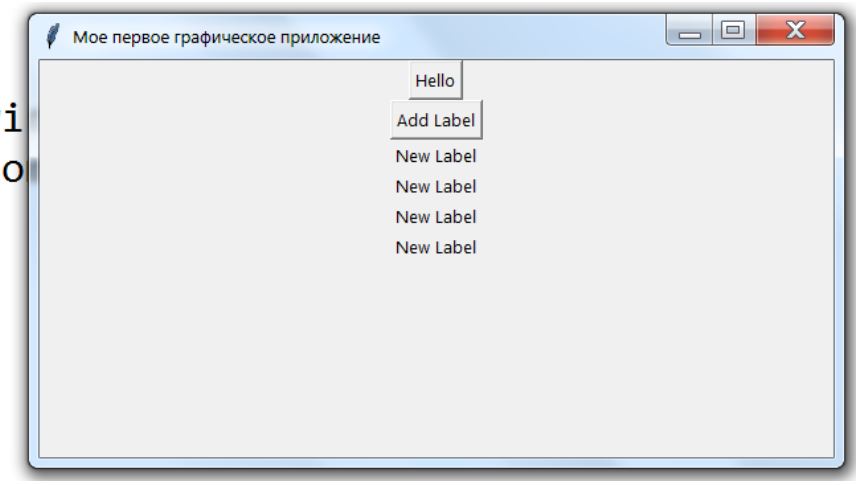
Запускаем программу:

```
18                      )
19
20  btn2 = tk.Button(wi
21                      co
22                      )
23  btn1.pack()
24  btn2.pack()
25
26  win.mainloop()
27
```



Теперь каждое нажатие на кнопку – **Add Label** будет создавать внутри окна программы виджет – **Label** с текстом – **New Label**.

```
18                      )
19
20  btn2 = tk.Button(wi
21                      co
22                      )
23  btn1.pack()
24  btn2.pack()
25
26  win.mainloop()
27
```



Существует еще один способ передать в параметр command функцию.

Этот способ называется – **lambda функция**.

Что такое лямбда функция в Python?

Лямбда функция в Python — это просто еще один способ определения функции.

Пример:

Способ определения функции, с которым мы уже знакомы:

```
def f(x):
    return x * x

print(f(5))
```

25

Способ использования лямбда функции:

```
f = lambda x: x * x
print(f(5))
```

25