Тема 10. Создание GUI на Python с помощью библиотеки Tkinter

Введение в tkinter

В многообразии программ, которые пишут программисты, выделяют приложения с графическим пользовательским интерфейсом (GUI). При создании таких программ становятся важными не только алгоритмы обработки данных, но и разработка для пользователя программы удобного интерфейса, взаимодействуя с которым, он будет определять поведение приложения.

Современный пользователь в основном взаимодействует с программой с помощью различных кнопок, меню, значков, вводя информацию в специальные поля, выбирая определенные значения в списках и т. д. Эти "изображения" в определенном смысле и формируют GUI, в дальнейшем мы их будем называть виджетами (от англ. widget - "штучка").

Для языка программирования Python такие виджеты включены в специальную библиотеку — tkinter. Если ее импортировать в программу (скрипт), то можно пользоваться ее компонентами, создавая графический интерфейс.

Последовательность шагов при создании графического приложения имеет свои особенности. Программа должна выполнять свое основное назначение, быть удобной для пользователя, реагировать на его действия. Мы не будем вдаваться в подробности разработки, а рассмотрим какие этапы приблизительно нужно пройти при программировании, чтобы получить программу с GUI:

1. Импорт библиотеки
2. Создание главного окна
3. Создание виджет
4. Установка их свойств
5. Определение событий
6. Определение обработчиков событий
7. Расположение виджет на главном окне
8. Отображение главного окна

Импорт модуля tkinter

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 нем содержатся. Следует обратить внимание, что в версии Python 3 имя модуля пишется со строчной буквы (`tkinter`), хотя в более ранних

версиях использовалась прописная (Tkinter). Итак, первая строчка программы должна выглядеть так:

```
from tkinter import *
```

Создание главного окна

В современных операционных системах любое пользовательское приложение заключено в окно, которое можно назвать главным, т.к. в нем располагаются все остальные виджеты. Объект окна верхнего уровня создается при обращении к классу Tk модуля tkinter. Переменную связанную с объектом-окном принято называть root (хотя понятно, что можно назвать как угодно, но так уж принято). Вторая строчка кода:

```
root = Tk()
```

Создание виджет

Допустим в окне будет располагаться всего одна кнопка. Кнопка создается при обращении к классу Button модуля 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:

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

Установка свойств виджет

У кнопки много свойств: размер, цвет фона и надписи и др. Мы рассмотрим их на следующем уроке. Пока же установим всего одно свойство — текст надписи (text):

```
but["text"] = "Печать"
```

Определение событий и их обработчиков

Многообразие событий и способов их обработки будет рассмотрено на следующих уроках. Здесь же просто коснемся данного вопроса в связи с потребностью.

Что же будет делать кнопка и в какой момент она это будет делать? Предположим, что задача кнопки вывести какое-нибудь сообщение в поток вывода, используя функцию print. Делать она это будет при нажатии на нее левой кнопкой мыши.

Действия (алгоритм), которые происходят при том или ином событии, могут быть достаточно сложным. Поэтому часто их оформляют в виде функции, а затем вызывают, когда они понадобятся. Пусть у нас печать на экран будет оформлена в виде функции printer:

```
def printer(event):
    print ("Как всегда очередной 'Hello World!'")
```

Не забывайте, что функцию желательно (почти обязательно) размещать в начале кода. **Параметр `event` – это какое-либо событие.**

Событие нажатия левой кнопкой мыши выглядит так: `<Button-1>`. Требуется связать это событие с обработчиком (функцией `printer`). Для связи предназначен метод `bind`. Синтаксис связывания события с обработчиком выглядит так:

```
but.bind("<Button-1>",printer)
```

Событие – именился размер окна:

```
# обработчик res  закрытия окна
root.bind('<Configure>',res)
```

В обработчике можно получить новый размер графического окна:

```
    str=root.geometry()
    k1=str.index('x')
    xmax1=str[:k1]
    k2=str.index('+',k1+1)
    ymax1=str[k1+1:k2]
    xmax=int(xmax1)
    ymax=int(ymax1)
```

Событие – закрыть окно:

```
# обработчик window_deleted  закрытия окна
root.protocol('WM_DELETE_WINDOW', window_deleted)
```

Размещение виджет

Если вы заметите, то в любом приложении виджеты не разбросаны по окну как попало, а хорошо организованы, интерфейс продуман до мелочей и обычно подчинен определенным стандартам. До стандартов нам далеко, нужно просто кнопку как-то отобразить в окне. Самый простой способ — это использование метода `pack`.

```
but.pack()
```

Если не вставить эту строчку кода, то кнопка в окне так и не появится, хотя она есть в программе.

Отображение главного окна

Ну и наконец, главное окно тоже не появится, пока не будет вызван специальный метод `mainloop`:

```
root.mainloop()
```

Данная строчка кода должна быть всегда в конце скрипта!

В итоге, код программы может выглядеть таким образом:

```
from tkinter import *

def printer(event):
     print ("Как всегда очередной 'Hello World!'")
```

```
root = Tk()
but = Button(root)
but["text"] = "Печать"
but.bind("<Button-1>",printer)

but.pack()
root.mainloop()
```

При программировании графического интерфейса пользователя более эффективным оказывается объектно-ориентированный подход. Поэтому многие «вещи» оформляются в виде классов. В нашем примере также можно использовать класс:

```
from tkinter import *

class But_print:
     def __init__(self):
          self.but = Button(root)
          self.but["text"] = "Печать"
          self.but.bind("<Button-1>",self.printer)
          self.but.pack()
     def printer(self,event):
          print ("Как всегда очередной 'Hello World!'")

root = Tk()
obj = But_print()
root.mainloop()
```

Разметка виджетов в Tkinter — pack, grid и place

Познакомимся с **менеджерами разметки**. Когда мы **создаем графический интерфейс** нашего приложения, мы определяем, какие виджеты будем использовать, и как они будут расположены в приложении. Для того, чтобы организовать виджеты в приложении, используются специальные невидимые объекты – **менеджеры разметки**.

Существует два вида виджетов:

- контейнеры;
- дочерние виджеты.

Контейнеры объединяют виджеты для формирования разметки. У Tkinter есть три встроенных менеджера разметки: **pack**, **grid** и **place**.

- **Place** – это менеджер геометрии, который размещает виджеты, используя абсолютное позиционирование.
- **Pack** – это менеджер геометрии, который размещает виджеты по горизонтали и вертикали.
- **Grid** – это менеджер геометрии, который размещает виджеты в двухмерной сетке.

Метод place() в Tkinter — Абсолютное позиционирование

В большинстве случаев разработчикам необходимо использовать **менеджеры разметки**. Есть несколько ситуаций, в которых следует использовать именно **абсолютное позиционирование**. В рамках абсолютного позиционирования разработчик определяет позицию и размер каждого виджета в пикселях. Во время изменения размеров окна размер и позиция виджетов не меняются.

Изображения для примера: bardejov.jpg rotunda.jpg mincol.jpg сохраните в папке рядом с файлом `absolute.py` код для которого будет ниже.

Для работы этого скрипта необходимо установить пакет Image:

```
python -m pip install Image
```

Таким образом, на разных платформах приложения выглядят по-разному. То, что выглядит нормально на **Linux**, может отображаться некорректно на **Mac OS**. Изменение шрифтов в нашем приложении также может испортить разметку. Если мы переведем наше приложение на другой язык, мы должны доработать и разметку.

absolute.py

```
from PIL import Image, ImageTk
from tkinter import Tk, BOTH
from tkinter.ttk import Frame, Label, Style


class Example(Frame):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.master.title("Absolute positioning")
```

```
            self.pack(fill=BOTH, expand=1)

            Style().configure("TFrame",
background="#333")

            bard = Image.open("bardejov.jpg")
            bardejov = ImageTk.PhotoImage(bard)
            label1 = Label(self, image=bardejov)
            label1.image = bardejov
            label1.place(x=20, y=20)

            rot = Image.open("rotunda.jpg")
            rotunda = ImageTk.PhotoImage(rot)
            label2 = Label(self, image=rotunda)
            label2.image = rotunda
            label2.place(x=40, y=160)

            minc = Image.open("mincol.jpg")
            mincol = ImageTk.PhotoImage(minc)
            label3 = Label(self, image=mincol)
            label3.image = mincol
            label3.place(x=170, y=50)


    def main():

        root = Tk()
        root.geometry("300x280+300+300")
        app = Example()
        root.mainloop()


    if __name__ == '__main__':
        main()
```

В этом примере мы расположили три изображения при помощи абсолютного позиционирования. ***Мы использовали менеджер геометрии place.***

```
from PIL import Image, ImageTk
```

Мы использовали **Image** и **ImageTk** из модуля **PIL** (Python Imaging Library).

```
style = Style()
style.configure("TFrame", background="#333")
```

При помощи стилей, мы изменили фон нашего окна на темно-серый.

```
bard = Image.open("bardejov.jpg")
bardejov = ImageTk.PhotoImage(bard)
```

Мы *создали объект изображения* и объект фото изображения из сохраненных ранее изображений в текущей рабочей директории.

```
label1 = Label(self, image=bardejov)
```

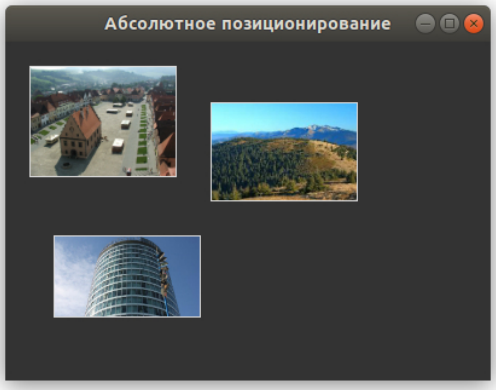
Мы создали `Label` (ярлык) с изображением. Данные ярлыки могут содержать как изображения, так и текст.

```
label1.image = bardejov
```

Нам нужно сохранить ссылку на изображение, чтобы не потерять его если сборщик мусора (Garbage collector) его не закроет.

```
label1.place(x=20, y=20)
```

Ярлык размещен в рамке по координатам `x=20` и `y=20`.



Tkinter pack() — размещение виджетов по горизонтали и вертикали

Менеджер геометрии `pack()` упорядочивает виджеты в горизонтальные и вертикальные блоки. Макетом можно управлять с помощью параметров `fill`, `expand` и `side`.

Пример создания кнопок в Tkinter

В следующем примере мы разместим две кнопки в нижнем правом углу нашего окна. Для этого мы воспользуемся менеджером **pack**.

```
from tkinter import Tk, RIGHT, BOTH, RAISED
from tkinter.ttk import Frame, Button, Style


class Example(Frame):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.master.title("Кнопки в kinter")
        self.style = Style()
        self.style.theme_use("default")

        frame = Frame(self, relief=RAISED, borderwidth=1)
        frame.pack(fill=BOTH, expand=True)

        self.pack(fill=BOTH, expand=True)

        closeButton = Button(self, text="Закрыть")
        closeButton.pack(side=RIGHT, padx=5, pady=5)
        okButton = Button(self, text="Готово")
        okButton.pack(side=RIGHT)


def main():

    root = Tk()
    root.geometry("300x200+300+300")
    app = Example()
    root.mainloop()


if __name__ == '__main__':
    main()
```

У нас есть две рамки. **Первая рамка** – основная, а также вторая – дополнительная, которая растягивается в обе стороны и сдвигает две кнопки в нижнюю часть основной рамки. **Кнопки** находятся в горизонтальном контейнере и размещены в ее правой части.

```
frame = Frame(self, relief=RAISED, borderwidth=1)
frame.pack(fill=BOTH, expand=True)
```

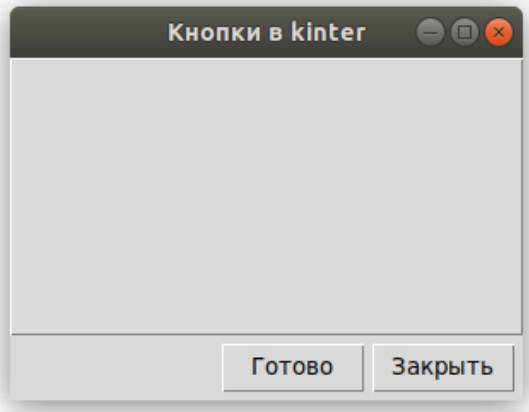
Мы создали еще один **виджет Frame**. Этот виджет занимает практически все пространство окна. Мы изменяем границы рамки, чтобы сама рамка была видна. По умолчанию она плоская.

```
closeButton = Button(self, text="Закрыть")
closeButton.pack(side=RIGHT, padx=5, pady=5)
```

Кнопка **closeButton** создана. Она расположена в горизонтальном контейнере. Параметр **side** позволяет поместить кнопку в правой части горизонтальной полосы. Параметры `padx` и `pady` позволяют **установить отступ** между виджетами. Параметр `padx` устанавливает пространство между виджетами кнопки `closeButton` и правой границей корневого окна.

```
okButton.pack(side=RIGHT)
```

Кнопка **okButton** размещена возле **closeButton** с установленным отступом (padding) в 5 пикселей.



Создаем приложение для отзывов на Tkinter

Менеджер pack – это простой **менеджер разметки**. Его можно использовать для простых задач разметки. Чтобы создать более сложную разметку, необходимо использовать больше рамок, каждая из которых имеет собственный менеджер разметки.

review.py

```
from tkinter import Tk, Text, BOTH, X, N, LEFT
from tkinter.ttk import Frame, Label, Entry


class Example(Frame):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.master.title("Оставить отзыв")
        self.pack(fill=BOTH, expand=True)

        frame1 = Frame(self)
        frame1.pack(fill=X)

        lbl1 = Label(frame1, text="Заголовок", width=10)
        lbl1.pack(side=LEFT, padx=5, pady=5)
```

```
        entry1 = Entry(frame1)
        entry1.pack(fill=X, padx=5, expand=True)

        frame2 = Frame(self)
        frame2.pack(fill=X)

        lbl2 = Label(frame2, text="Автор", width=10)
        lbl2.pack(side=LEFT, padx=5, pady=5)

        entry2 = Entry(frame2)
        entry2.pack(fill=X, padx=5, expand=True)

        frame3 = Frame(self)
        frame3.pack(fill=BOTH, expand=True)

        lbl3 = Label(frame3, text="Отзыв", width=10)
        lbl3.pack(side=LEFT, anchor=N, padx=5, pady=5)

        txt = Text(frame3)
        txt.pack(fill=BOTH, pady=5, padx=5, expand=True)


def main():

    root = Tk()
    root.geometry("300x300+300+300")
    app = Example()
    root.mainloop()


if __name__ == '__main__':
    main()
```

На этом примере видно, как можно создать **более сложную разметку** с многочисленными рамками и **менеджерами pack()**.

```
1
self.pack(fill=BOTH, expand=True)
```

Первая рамка является базовой. На ней располагаются все остальные рамки. Стоит отметить, что даже при организации дочерних виджетов в рамках, мы управляем ими на базовой рамке.

```
frame1 = Frame(self)
frame1.pack(fill=X)

lbl1 = Label(frame1, text="Заголовок", width=10)
lbl1.pack(side=LEFT, padx=5, pady=5)

entry1 = Entry(frame1)
entry1.pack(fill=X, padx=5, expand=True)
```

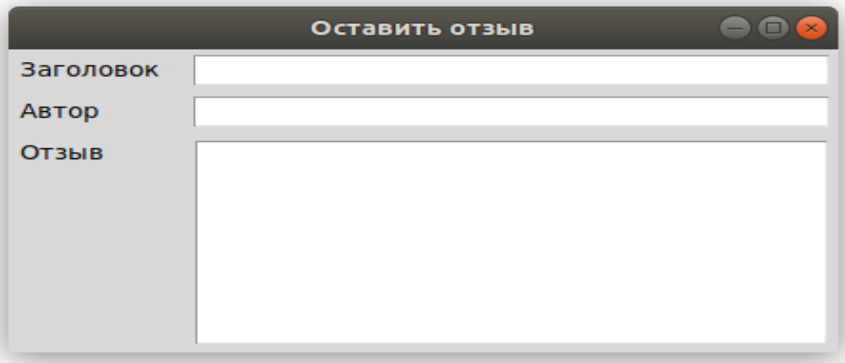
Первые два виджета размещены на первой рамке. Поле для ввода данных растянуто горизонтально с параметрами **fill** и **expand**.

```
frame3 = Frame(self)
frame3.pack(fill=BOTH, expand=True)

lbl3 = Label(frame3, text="Отзыв", width=10)
lbl3.pack(side=LEFT, anchor=N, padx=5, pady=5)

txt = Text(frame3)
txt.pack(fill=BOTH, pady=5, padx=5, expand=True)
```

В третьей рамке мы разместили ярлык и виджет для ввода текста. Ярлык закреплен по северной стороне `anchor=N`, а виджет текста занимает все остальное пространство.



Разметка grid() в Tkinter для создания калькулятора

Менеджер геометрии grid() в Tkinter используется для создания сетки кнопок для калькулятора.

calculator.py

```
from tkinter import Tk, W, E
from tkinter.ttk import Frame, Button, Entry, Style


class Example(Frame):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.master.title("Калькулятор на Tkinter")

        Style().configure("TButton", padding=(0, 5, 0, 5),
                                            font='serif 10')

        self.columnconfigure(0, pad=3)
        self.columnconfigure(1, pad=3)
        self.columnconfigure(2, pad=3)
        self.columnconfigure(3, pad=3)

        self.rowconfigure(0, pad=3)
```

```
self.rowconfigure(1, pad=3)
self.rowconfigure(2, pad=3)
self.rowconfigure(3, pad=3)
self.rowconfigure(4, pad=3)

entry = Entry(self)
entry.grid(row=0, columnspan=4, sticky=W+E)
cls = Button(self, text="Очистить")
cls.grid(row=1, column=0)
bck = Button(self, text="Удалить")
bck.grid(row=1, column=1)
lbl = Button(self)
lbl.grid(row=1, column=2)
clo = Button(self, text="Закрыть")
clo.grid(row=1, column=3)
sev = Button(self, text="7")
sev.grid(row=2, column=0)
eig = Button(self, text="8")
eig.grid(row=2, column=1)
nin = Button(self, text="9")
nin.grid(row=2, column=2)
div = Button(self, text="/")
div.grid(row=2, column=3)

fou = Button(self, text="4")
fou.grid(row=3, column=0)
fiv = Button(self, text="5")
fiv.grid(row=3, column=1)
six = Button(self, text="6")
six.grid(row=3, column=2)
mul = Button(self, text="*")
mul.grid(row=3, column=3)

one = Button(self, text="1")
one.grid(row=4, column=0)
two = Button(self, text="2")
two.grid(row=4, column=1)
thr = Button(self, text="3")
thr.grid(row=4, column=2)
mns = Button(self, text="-")
mns.grid(row=4, column=3)

zer = Button(self, text="0")
zer.grid(row=5, column=0)
dot = Button(self, text=".")
dot.grid(row=5, column=1)
equ = Button(self, text="=")
equ.grid(row=5, column=2)
pls = Button(self, text="+")
pls.grid(row=5, column=3)

self.pack()
```

```
def main():
    root = Tk()
    app = Example()
    root.mainloop()


if __name__ == '__main__':
    main()
```

Менеджер **grid()** используется для организации кнопок в контейнере рамки.

```
Style().configure("TButton", padding=(0, 5, 0, 5),
    font='serif 10')
```

Мы настроили **виджет кнопки** так, чтобы отображался специфический шрифт и применялся отступ (padding) в 3 пикселя.

```
self.columnconfigure(0, pad=3)
...
self.rowconfigure(0, pad=3)
```

Мы использовали методы **columnconfigure**() и **rowconfigure**() чтобы создать определенное пространство в сетке строк и столбцов. Благодаря этому шагу мы разделяем кнопки определенным пустым пространством.

```
entry = Entry(self)
entry.grid(row=0, columnspan=4, sticky=W+E)
```

Виджет графы ввода – это место, где будут отображаться цифры. Данный виджет расположен в первом ряду и охватывает все четыре столбца. Виджеты могут не занимать все пространство, которое выделяется клетками в созданной сетке.

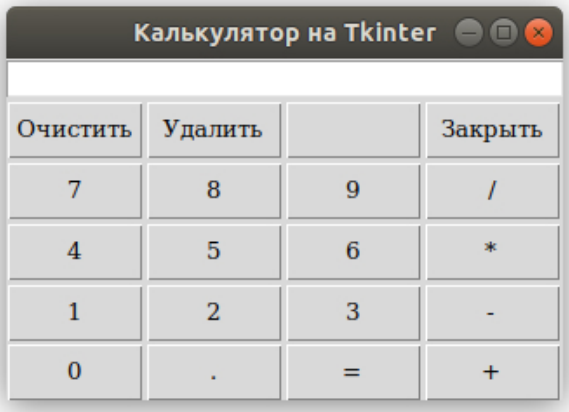
Параметр **sticky** расширяет виджет в указанном направлении. В нашем случае, мы можем убедиться, что наш виджет графы ввода был расширен слева направо **W+E** (восток-запад).

```
cls = Button(self, text="Очистить")
cls.grid(row=1, column=0)
```

Кнопка очистки установлена во второй строке и первом столбце. Стоит отметить, что строки и столбцы начинаются с нуля.

```
self.pack()
```

Метод pack() показывает виджет рамки и дает ей первоначальный размер. Если дополнительные параметры не указываются, размер будет таким, чтобы все дочерние виджеты могли поместиться. Этот метод компонует виджет рамки в верхнем корневом окне, которое также является контейнером. Менеджер **grid()** используется для организации кнопок в виджете рамки.



Пример создания диалогового окна в Tkinter

Следующий пример **создает диалоговое окно**, используя менеджер геометрии grid.

windows.py

```
from tkinter import Tk, Text, BOTH, W, N, E, S
from tkinter.ttk import Frame, Button, Label, Style


class Example(Frame):

    def __init__(self):
        super().__init__()
        self.initUI()


    def initUI(self):

        self.master.title("Диалоговое окно в Tkinter")
        self.pack(fill=BOTH, expand=True)

        self.columnconfigure(1, weight=1)
        self.columnconfigure(3, pad=7)
        self.rowconfigure(3, weight=1)
        self.rowconfigure(5, pad=7)

        lbl = Label(self, text="Окна")
        lbl.grid(sticky=W, pady=4, padx=5)

        area = Text(self)
        area.grid(row=1, column=0, columnspan=2, rowspan=4,
                                        padx=5, sticky=E+W+S+N)

        abtn = Button(self, text="Активир.")
        abtn.grid(row=1, column=3)

        cbtn = Button(self, text="Закрыть")
```

```
        cbtn.grid(row=2, column=3, pady=4)

        hbtn = Button(self, text="Помощь")
        hbtn.grid(row=5, column=0, padx=5)

        obtn = Button(self, text="Готово")
        obtn.grid(row=5, column=3)


def main():

    root = Tk()
    root.geometry("350x300+300+300")
    app = Example()
    root.mainloop()


if __name__ == '__main__':
    main()
```

В этом примере мы использовали **виджет ярлыка**, **текстовой виджет** и четыре кнопки.

```
self.columnconfigure(1, weight=1)
self.columnconfigure(3, pad=7)
self.rowconfigure(3, weight=1)
self.rowconfigure(5, pad=7)
```

Мы добавили небольшое пространство между виджетами в сетке. Параметр **weight** создает возможность расширения второго столбца и четвертого ряда. В этом ряду и столбце находится **текстовой виджет**, поэтому оставшееся пространство заполняет данный виджет.

```
lbl = Label(self, text="Окна")
lbl.grid(sticky=W, pady=4, padx=5)
```

Виджет ярлыка также создается и *помещается в сетку*. Если не указываются ряд и столбец, тогда он займет первый ряд и столбец. Ярлык закрепляется у западной части окна `sticky=W` и имеет определенные отступы вокруг своих границ.

```
area = Text(self)
area.grid(row=1, column=0, columnspan=2, rowspan=4,
    padx=5, sticky=E+W+S+N)
```

Создается **текстовый виджет** и помещается во второй ряд и первый столбец. Он охватывает два столбца и четыре строки.

Между виджетом и левым краем корневого окна присутствует пространство в 4 пикселя. Также, виджет закреплен около всех четырех

сторон. Поэтому, когда окно расширяется, виджеты текстов увеличиваются во всех направлениях.

```
abtn = Button(self, text="Активир.")
abtn.grid(row=1, column=3)

cbtn = Button(self, text="Закрыть")
cbtn.grid(row=2, column=3, pady=4)
```

Эти две кнопки находятся возле текстового виджета.

```
hbtn = Button(self, text="Помощь")
hbtn.grid(row=5, column=0, padx=5)

obtn = Button(self, text="Готово")
obtn.grid(row=5, column=3)
```

Эти две кнопки находятся под текстовым виджетом. Кнопка «Помощь» расположена в первом столбце, а кнопка «Готово» в последнем столбце.