Компьютерные технологии

Лекция № 9.

Разработка графического интерфейса: Tkinter

Нижний
Новгород
2022 г.

Tkinter

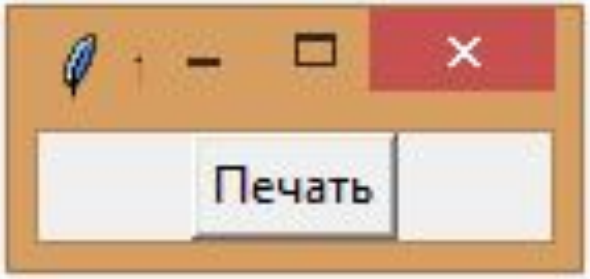
Это кроссплатформенная библиотека для разработки графического интерфейса на языке Python (**начиная с Python 3.0 переименована в tkinter**). Tkinter расшифровывается как Tk interface, и является интерфейсом к Tcl/Tk. Tkinter входит в стандартный дистрибутив Python.

Последовательность шагов при создании графического приложения имеет свои особенности. Этапы, которые нужно пройти, чтобы получить программу с GUI:

- Импорт библиотеки
- Создание главного окна
- Создание виджет
- Установка их свойств
- Определение событий
- Определение обработчиков событий
- Расположение виджет на главном окне
- Отображение главного окна

Tkinter

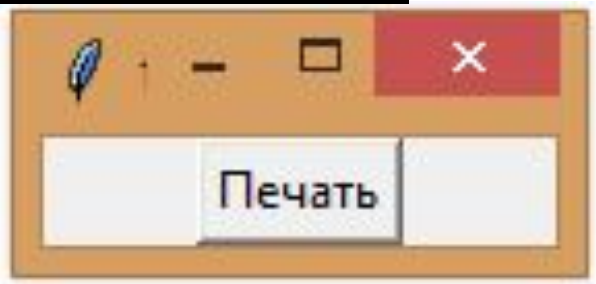
```
from tkinter import *    #Импорт модуля tkinter

#Определение событий и их обработчиков
def printer(event):
    print ("Как всегда очередной 'Hello World!'")

root = Tk()     #Создание главного окна
but = Button(root)    #Создание виджет
but["text"] = "Печать"    #Установка свойств виджет
but.bind("<Button-1>",printer)

but.pack()    #Размещение виджет
root.mainloop()  #Отображение главного окна
```



Tkinter

```
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()
```

```
C:\Users\Olga>python C:\Users\Olga\Downloads\simple_plot.py
Как всегда очередной 'Hello World!'
```



Разметка в Tkinter

Для того, чтобы организовать виджеты в приложении, используются специальные невидимые объекты – менеджеры разметки.

Существует два вида виджетов: **контейнеры** и **их дочерние виджеты**. Контейнеры объединяют их дочерние 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.

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.

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.

Абсолютное позиционирование

Используем менеджер геометрии **place.**

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

from PIL import Image, ImageTk

При помощи стилей, изменим фон окна на темно-серый.

style = Style()

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

Создаем объект изображения и объект фото изображения из изображения в текущей рабочей директории.

img = Image.open(“img_1.jpg")

photo = ImageTk.PhotoImage(bard)

Создаем ***Label*** с изображением. Данные ярлыки могут содержать как изображения, так и текст: ***label1 = Label(self, image= photo)***

Сохраняем ссылку на изображение: ***label1.image = photo***

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

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

Абсолютное позиционирование

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

class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Absolute positioning")
        self.pack(fill=BOTH, expand=1)

        self.style = Style()
        self.style.theme_use("default")

        img = Image.open("1.JPG")
        photo = ImageTk.PhotoImage(img)
        label1 = Label(self, image=photo)
        label1.image = photo
        label1.place(x=20, y=20)

        rot = Image.open("2.JPG")
        rotunda = ImageTk.PhotoImage(rot)
        label2 = Label(self, image=rotunda)
        label2.image = rotunda
        label2.place(x=40, y=160)

        minc = Image.open("3.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)
    root.mainloop()

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

Примеры Кнопок

Создадим две рамки. Первая рамка – основная, 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. Виджет Frame занимает практически все пространство окна. ***frame = Frame(self, relief=RAISED, borderwidth=1)***

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

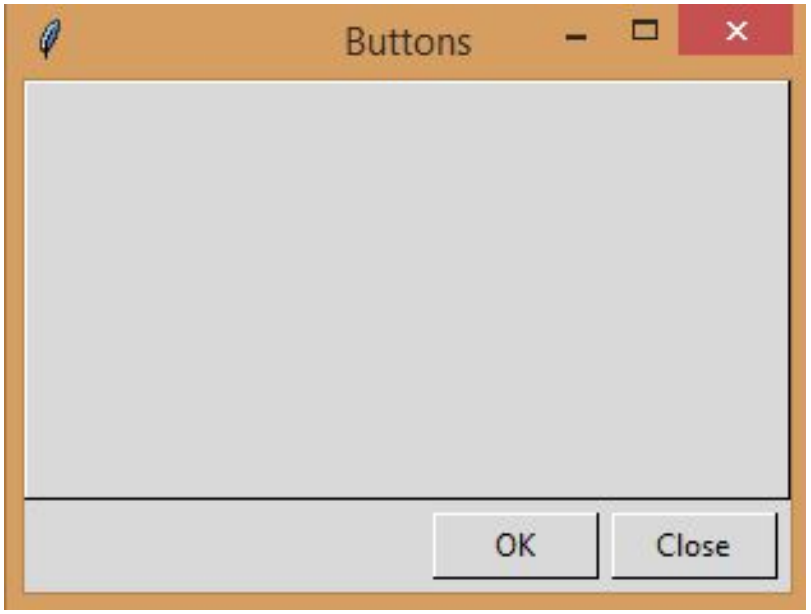
Кнопка closeButton расположена в горизонтальном контейнере. Параметр side позволяет поместить кнопку в правой части горизонтальной полосы. Параметры padx и pady позволяют установить небольшое пространство между виджетами. Параметр padx устанавливает пространство между виджетами кнопки, closeButton и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

closeButton = Button(self, text="Close")

closeButton.pack(side=RIGHT, padx=5, pady=5)

Примеры Кнопок

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

class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Buttons")
        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="Close")
        closeButton.pack(side=RIGHT, padx=5, pady=5)
        okButton = Button(self, text="OK")
        okButton.pack(side=RIGHT)

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

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



Виджеты в Tkinter

Виджеты – это базовые блоки для создания графического интерфейса программы. За годы развития программирования некоторые из виджетов стали стандартными во всех языках и на всех платформах.

Например, это виджеты **кнопок**, **флажки** или **полоса прокрутки**. Некоторые из виджетов могут иметь другие названия в Tkinter. Например, классические флажки (check box) называются check button. В Tkinter реализован небольшой набор виджетов, который покрывает базовые нужды программирования. Дополнительные виджеты могут быть созданы как пользовательские виджеты.

Checkbutton

Это виджет, который имеет два состояния: «*включен*» и «*выключен*». Состояние «включен» визуально символизируется соответствующей отметкой. Виджет используется для обозначения каких-либо логических свойств. Виджет Checkbutton имеет флажок и текстовый ярлык.

Создаем **BooleanVar** объект. Данные объекты позволяют хранить логические значения виджетов в **Tkinter**.

self.var = BooleanVar()

Создаем экземпляр **Checkbutton**. Объект, хранящий значение, соединяется с виджетом посредством параметра **variable**. При нажатии на флажок, вызывается метод ***onClick(***). Для этого также используется параметр **command**.

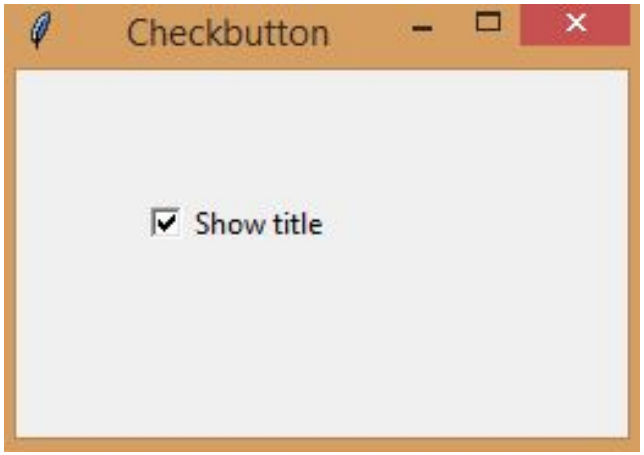
cb = Checkbutton(self, text="Show title", variable=self.var, command=self.onClick)

Изначально, название отображается в заголовке окна. Поэтому, изначально делаем флажок включенным с помощью метода ***select().***

cb.select()

Checkbutton

```
class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Checkbutton")

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

        cb = Checkbutton(self, text="Show title",
            variable=self.var, command=self.onClick)
        cb.select()
        cb.place(x=50, y=50)

    def onClick(self):
        if self.var.get() == True:
            self.master.title("Checkbutton")
        else:
            self.master.title("")
```



Шкала

Scale – это виджет, который позволяет пользователю графически выбирать значение, перемещая определенный ползунок по ограниченной линии.

Создаем виджет Scale, указываем нижнюю и верхнюю рамки значений. From – это регулярное ключевое слово в Python, поэтому необходимо прописывать символ подчеркивание (Underscope) после параметра. Когда мы двигаем ползунок по шкале, вызывается метод onScale().

```
scale = Scale(self, from_=0, to=100, command=self.onScale)
```

Создаются держатель целого значения и виджет ярлык. Значение из держателя будет отображаться в виджете ярлыке.

```
self.var = IntVar()
self.label = Label(self, text=0, textvariable=self.var)
```

Метод onScale() получает текущее выбранное значение из виджета шкалы как параметр. Значение сначала конвертируется в значение с плавающей точкой, после чего конвертируется в целое. В результате, значение передается держателю и отображается на виджете ярлыке.

```
def onScale(self, val):
    v = int(float(val))
    self.var.set(v)
```

Шкала

```
class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)

        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Scale")
        self.style = Style()
        self.style.theme_use("default")

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

        scale = Scale(self, from_=0, to=100,
            command=self.onScale)
        scale.pack(side=LEFT, padx=15)

        self.var = IntVar()
        self.label = Label(self, text=0, textvariable=self.var)
        self.label.pack(side=LEFT)

    def onScale(self, val):
        v = int(float(val))
        self.var.set(v)
```

Списки

Виджет ***Listbox*** позволяет отображать список объектов. Он позволяет пользователю выбирать один или несколько объектов.

Список, который будет показан:

acts = ['Scarlett Johansson', 'Rachel Weiss', 'Natalie Portman', 'Jessica Alba']

создаем экземпляр Listbox и включаем в него все объекты из указанного выше списка.

```
lb = Listbox(self)
for i in acts:
    lb.insert(END, i)
```

При выборе объекта в списке, генерируется событие <>. Присваиваем метод ***onSelect()*** этому событию.

```
lb.bind("<<ListboxSelect>>", self.onSelect)
```

Списки

Создаем ярлык и *держатель его значения*. С помощью этого ярлыка отображаем выбранный на текущий момент объект.

self.var = StringVar()

self.label = Label(self, text=0, textvariable=self.var)

Получаем отправителя события – это виджет списка.

sender = val.widget

Узнаем индекс выбранного элемента при помощи метода ***curselection():***

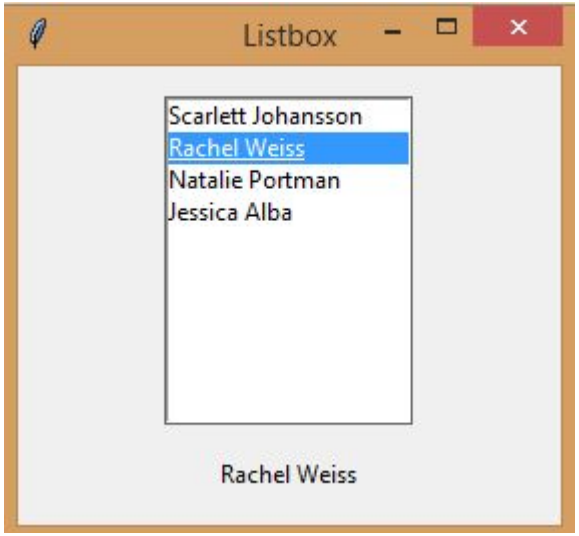
idx = sender.curselection()

Фактическое значение извлекается при помощи метода get(), который получает индекс объекта:

value = sender.get(idx)

Списки

```
from tkinter import Tk, BOTH, Listbox, StringVar, END
from tkinter import ttk
from tkinter.ttk import Frame, Label, Scale, Style

class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Listbox")
        self.pack(fill=BOTH, expand=1)
        acts = ['Scarlett Johansson', 'Rachel Weiss',
            'Natalie Portman', 'Jessica Alba']

        lb = Listbox(self)
        for i in acts:
            lb.insert(END, i)

        lb.bind("<<ListboxSelect>>", self.onSelect)

        lb.pack(pady=15)

        self.var = StringVar()
        self.label = Label(self, text=0, textvariable=self.var)
        self.label.pack()

    def onSelect(self, val):
        sender = val.widget
        idx = sender.curselection()
        value = sender.get(idx)
        self.var.set(value)

def main():
    root = Tk()
    ex = Example(root)
    root.geometry("300x250+300+300")
    root.mainloop()

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



Простое меню

Меню – одна из наиболее заметных и используемых частей графического интерфейса приложений. Создадим меню с одним объектом. Выбирая объект «Exit» (Выход) в меню, мы закрываем приложение. Создаем панель меню. Используем виджет ***Menu***, который настраиваем для отображения в качестве меню для корневого окна.

menubar = Menu(self.parent)

self.parent.config(menu=menubar)

Создаем объект меню ***file***. Меню – это выпадающее окно, содержащее команды.

fileMenu = Menu(menubar)

Добавляем команду к file меню. Эта команда будет вызывать метод onExit().

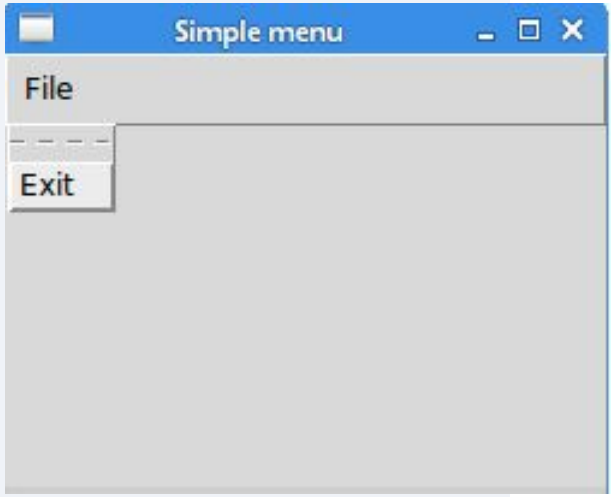
fileMenu.add_command(label="Exit", command=self.onExit)

Меню file добавляется на панель меню при помощи метода add_cascade().

menubar.add_cascade(label="File", menu=fileMenu)

Простое меню

```
from Tkinter import Tk, Frame, Menu

class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Simple menu")

        menubar = Menu(self.parent)
        self.parent.config(menu=menubar)

        fileMenu = Menu(menubar)
        fileMenu.add_command(label="Exit", command=self.onExit)
        menubar.add_cascade(label="File", menu=fileMenu)

    def onExit(self):
        self.quit()
```



Подменю

Это меню, встроенные в другие объекты меню. Реализуем три опции в подменю от основного меню «file». У нас есть подменю с тремя командами.

```
submenu = Menu(fileMenu)
submenu.add_command(label="New feed")
submenu.add_command(label="Bookmarks")
submenu.add_command(label="Mail")
```

Добавляя меню к fileMenu, но не к панели меню. Так создается подменю. При помощи параметра underline можно создать горячие клавиши. Этот параметр подчеркивает символ, обозначающий горячую клавишу команды. В нашем случае, это первая буква. Позиции символов начинаются с нуля. При нажатии на меню ***File***, появляется контекстное окно. В меню Import также только один символ является подчеркнутым. Выбрать этот пункт меню можно при помощи мыши или сочетанием горячих клавиш Alt+I.

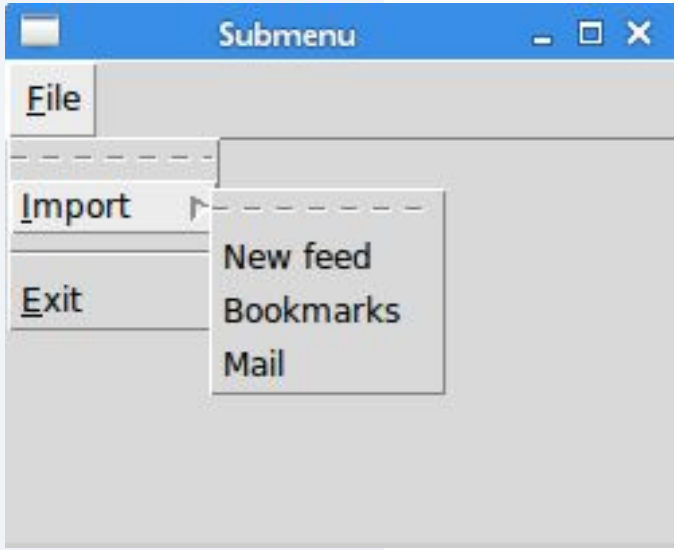
```
fileMenu.add_cascade(label='Import', menu=submenu, underline=0)
```

Разделитель – это горизонтальная линия, которая визуально разделяет команды меню. Благодаря разделителям мы можем группировать команды в меню.

```
fileMenu.add_separator()
```

Подменю

```
from Tkinter import Tk, Frame, Menu

class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Submenu")

        menubar = Menu(self.parent)
        self.parent.config(menu=menubar)

        fileMenu = Menu(menubar)

        submenu = Menu(fileMenu)
        submenu.add_command(label="New feed")
        submenu.add_command(label="Bookmarks")
        submenu.add_command(label="Mail")
        fileMenu.add_cascade(label='Import', menu=submenu, underline=0)

        fileMenu.add_separator()

        fileMenu.add_command(label="Exit", underline=0, command=self.onExit)
        menubar.add_cascade(label="File", underline=0, menu=fileMenu)

    def onExit(self):
        self.quit()
```



Файловый диалог

Диалог ***filedialog*** позволяет пользователю выбирать файл из системы компьютера.

В виджете Text будет отображаться контент выбранного файла.

```
self.txt = Text(self)
```

Это фильтры файлов. Первый показывает файлы Python, остальные показывают другие форматы файлов:

```
ftypes = [('Python files', '*.py'), ('All files', '*')]
```

Диалог был создан и отображается на экране. Мы получаем возвращенное значение, которое является названием выбранного файла.

```
dlg = filedialog.Open(self, filetypes = ftypes)
fl = dlg.show()
```

Читаем контент внутри файла.

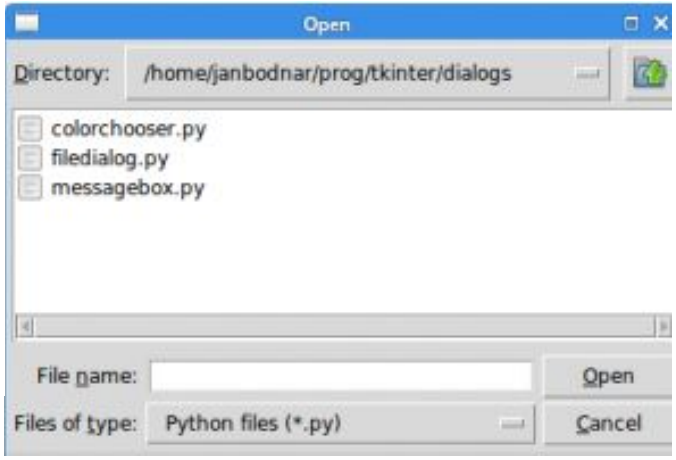
```
text = self.readFile(fl)
```

Текст вставляется в виджет Text.

```
self.txt.insert(END, text)
```

Файловый диалог

```
from tkinter import Tk, BOTH, Listbox, StringVar, END, Text, Menu
from tkinter import ttk
from tkinter.ttk import Frame, Label, Scale, Style
from tkinter import filedialog
class Example(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)

        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("File dialog")
        self.pack(fill=BOTH, expand=1)

        menubar = Menu(self.parent)
        self.parent.config(menu=menubar)

        fileMenu = Menu(menubar)
        fileMenu.add_command(label="Open", command=self.onOpen)
        menubar.add_cascade(label="File", menu=fileMenu)

        self.txt = Text(self)
        self.txt.pack(fill=BOTH, expand=1)

    def onOpen(self):
        ftypes = [('Python files', '*.py'), ('All files', '*')]
        dlg = filedialog.Open(self, filetypes = ftypes)
        fl = dlg.show()
        if fl != '':
            text = self.readFile(fl)
            self.txt.insert(END, text)

    def readFile(self, filename):
        f = open(filename, "r")
        text = f.read()
        return text

def main():
    root = Tk()
    ex = Example(root)
    root.geometry("300x250+300+300")
    root.mainloop()

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



Линии

Линия – это примитивный геометрический элемент.
На виджете Canvas создать линию можно при помощи метода create_line().

Параметрами метода create_line() являются координаты x и y, которые обозначают стартовую и конечную точки линии.

canvas.create_line(15, 25, 200, 25)

Нарисуем вертикальную линию. Опция dash позволяет оформить линию в форме ряда тире. Оформим линию, которая состоит из одинаковых сегментов тире в 4 пикселя и пустом пространстве в 2 пикселя.

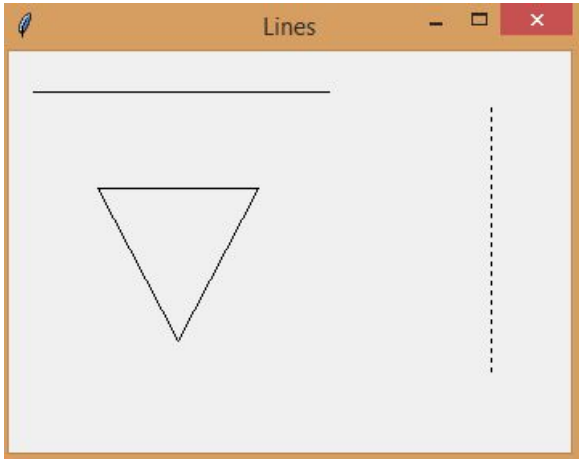
canvas.create_line(300, 35, 300, 200, dash=(4, 2))

Метод create_line() может содержать несколько точек. Этот код мы нарисовали треугольник.

canvas.create_line(55, 85, 155, 85, 105, 180, 55, 85)

Линии

```
from tkinter import Tk, Canvas, Frame, BOTH
from tkinter import ttk
from tkinter.ttk import Frame, Label, Scale, Style

class Example(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Lines")
        self.pack(fill=BOTH, expand=1)

        canvas = Canvas(self)
        canvas.create_line(15, 25, 200, 25)
        canvas.create_line(300, 35, 300, 200, dash=(4, 2))
        canvas.create_line(55, 85, 155, 85, 105, 180, 55, 85)

        canvas.pack(fill=BOTH, expand=1)

def main():
    root = Tk()
    ex = Example(root)
    root.geometry("400x250+300+300")
    root.mainloop()

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



Цвета

Цвет является объектом, который отображает комбинацию Красного, Зеленого и Синего цветов (RGB).

Создаем виджет canvas.

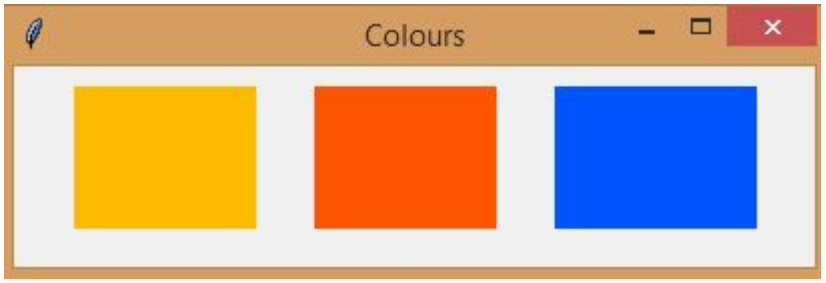
canvas = Canvas(self)

С помощью create_rectangle() создаем прямоугольники на холсте. Первыми четырьмя параметрами являются x и y координаты двух ограничительных точек: верхней левой и нижней правой. При помощи параметра outline можем задать цвет контура прямоугольников. А параметр fill используется для окрашивания всей внутренней области прямоугольника.

```
canvas.create_rectangle(30, 10, 120, 80,
    outline="#fb0", fill="#fb0")
```

Цвета

```
from tkinter import Tk, Canvas, Frame, BOTH
from tkinter import ttk
from tkinter.ttk import Frame, Label, Scale, Style
class Example(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)

        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Colours")
        self.pack(fill=BOTH, expand=1)

        canvas = Canvas(self)
        canvas.create_rectangle(30, 10, 120, 80,
            outline="#fb0", fill="#fb0")
        canvas.create_rectangle(150, 10, 240, 80,
            outline="#f50", fill="#f50")
        canvas.create_rectangle(270, 10, 370, 80,
            outline="#05f", fill="#05f")
        canvas.pack(fill=BOTH, expand=1)

def main():
    root = Tk()
    ex = Example(root)
    root.geometry("400x100+300+300")
    root.mainloop()

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



Рисуем текст

Первые два параметра – это x и y координаты центральной точки текста. Если закрепить текстовый объект по направлению запада, текст будет начинаться в этой части окна. Параметр font позволяет изменять шрифт текста, а параметр text отображает написанный текст в окне.

```
canvas.create_text(20, 30, anchor=W, font="Purisa",
    text="Most relationships seem so transitory")
```

Рисуем текст

```
from tkinter import Tk, Canvas, Frame, BOTH, W
from tkinter import ttk
class Example(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)

        self.parent = parent
        self.initUI()
    def initUI(self):
        self.parent.title("Lyrics")
        self.pack(fill=BOTH, expand=1)

        canvas = Canvas(self)
        canvas.create_text(20, 30, anchor=W, font="Purisa",
            text="Most relationships seem so transitory")
        canvas.create_text(20, 60, anchor=W, font="Purisa",
            text="They're good but not the permanent one")
        canvas.create_text(20, 130, anchor=W, font="Purisa",
            text="Who doesn't long for someone to hold")
        canvas.create_text(20, 160, anchor=W, font="Purisa",
            text="Who knows how to love without being told")
        canvas.create_text(20, 190, anchor=W, font="Purisa",
            text="Somebody tell me why I'm on my own")
        canvas.create_text(20, 220, anchor=W, font="Purisa",
            text="If there's a soulmate for everyone")
        canvas.pack(fill=BOTH, expand=1)
def main():
    root = Tk()
    ex = Example(root)
    root.geometry("420x250+300+300")
    root.mainloop()
if __name__ == '__main__':
    main()
```