Python 语言程序设计

陈 峦 副教授

13880209111，chluan@uestc.edu.cn

研究院大楼316#

第十三章
图形用户界面设计

- 图形用户界面（**Graphical User Interface，GUI**）是指由窗口、菜单、对话框等各种图形元素组成的用户界面，用户通过一定的方法（如鼠标动作或键盘操作）选择、激活这些图形元素，使计算机产生某种动作或变化，如实现计算、绘图等。

13.1 创建图形用户界面的步骤

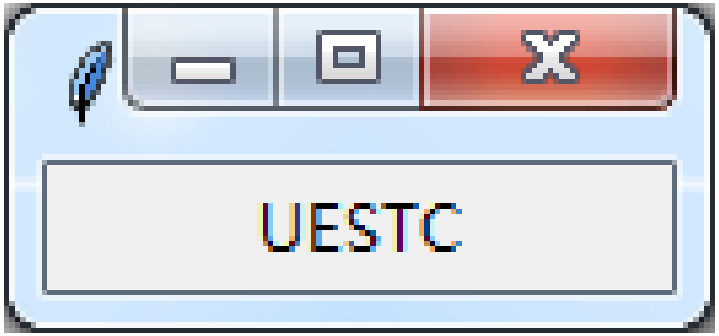
- Python的图形用户界面包括一个主窗口，主窗口中又包含各种控件。
- 主窗口和各种控件都称为对象，对象就具有属性和方法。

- Tkinter图形库是一个基于面向对象思想的图形用户界面设计工具包，图形用户界面的设计也是围绕各种对象的设计而展开的。
- Tkinter图形库的主体内容是tkinter模块。

- 使用tkinter模块创建一个图形用户界面应用程序的步骤:
- （1）创建主窗口。
- （2）在主窗口中添加各种控件并设置其属性。
- （3）调整对象的位置和大小。
- （4）为控件定义事件处理程序。
- （5）进入主事件循环。

- **1. 创建主窗口**
- 主窗口是图形用户界面的顶层窗口，也是控件的容器。
- 每一个图形用户界面程序必须有且只能有一个主窗口，并且要先于其他对象创建，其他对象都是主窗口的子对象。
- 如果程序没有显式创建主窗口而直接去创建其他控件，系统仍然会自动创建主窗口。

- 例：
- from tkinter import *
- s=Label(text="UESTC")
- #创建一个标签控件，同时自动创建主窗口。
- s.pack()



- **2. 在主窗口中添加各种控件并设置其属性**
- **tkinter模块中定义了许多控件类，利用这些控件类的构造函数可以创建控件对象，从而建立图形用户界面。**
- **例：**
- **from tkinter import ***
- **w=Tk()**
- **a=Label(w,text="UESTC")**
- **a.pack()**

- （1）常用控件
- Label:标签，用于显示说明文字。
- Message:消息，类似标签，但可显示多行文本。
- Button:按钮，用于执行命令。
- Radiobutton:单选按钮，用于从多个选项中选择一个选项。
- Checkbutton:复选框，用于表示是否选择某个选项。

- **Entry**:单行文本框，用于输入、编辑一行文本。
- **Text**:多行文本框，用于显示和编辑多行文本，支持嵌入图像。
- **Frame**:框架，是容器控件，用于控件组合与界面布局。
- **Listbox**:列表框，用于显示若干选项。
- **Scrollbar**:滚动条，用于滚动显示更多内容。

- **OptionMenu**:可选项，单击可选项按钮时，打开选项列表，按钮中显示被选中的项目。
- **Scale**:刻度条，允许用户通过移动滑块来选择数值。
- **Menu**:菜单，用于创建下拉式菜单或弹出式菜单。
- **Toplevel**:顶层窗口，这是一个容器控件，用于多窗口应用程序。
- **Canvas**:画布，用于绘图。

- （2）控件对象属性
- 每个控件都是对象，创建对象时设置的各种参数都是对象的属性，如标签控件的text属性。
- 控件有许多属性，在用构造函数创建控件对象时可以为一些属性设置属性值，而没有设置的属性也都有预定义的默认值。
- 创建对象时采用“属性名=属性值”的形式来设置属性值。

- 控件对象的属性值既可以在创建时指定，也可以在将来任何时候设置或修改。
- 每种控件类都提供configure()方法（configure可简写为config）用于修改属性值。
- 例：
- a.config(text="Goodbye!")

- tkinter模块还提供了另一种修改控件对象属性值的方法。
- 将对象视为一个字典，该字典以属性名作为关键字，以属性值作为字典的值。
- 例：
- a["text"]="Goodbye!"

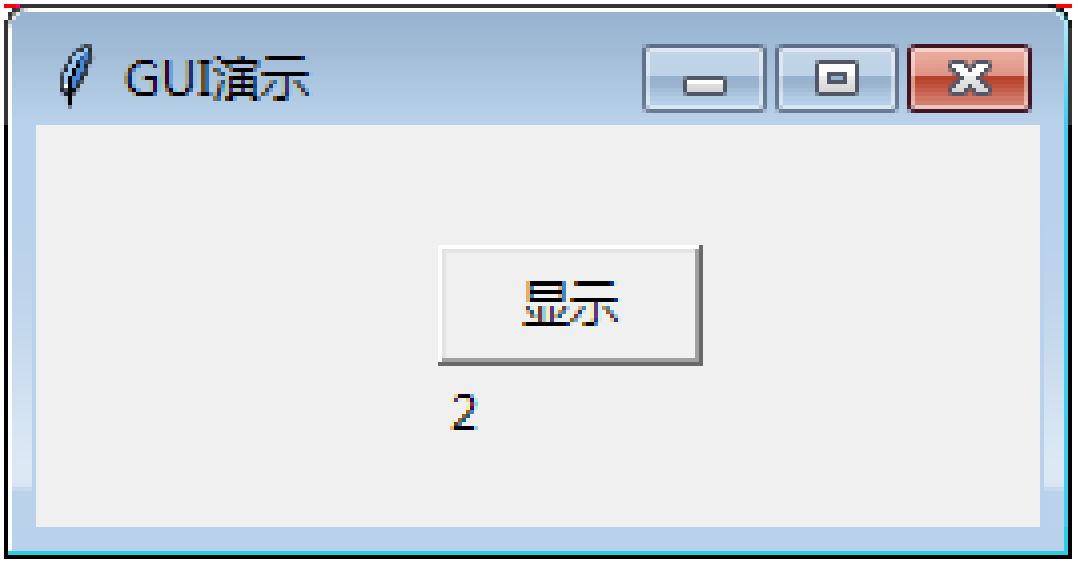
- 用字典方法每次只能修改一个属性的值，而用config()方法一次可以修改多个属性的值。
- 例：
- a.config(text="Goodbye!",fg="red",bg="blue")

- **3. 调整对象的位置和大小**
- 调用对象的pack()、grid()或place()方法，通过布局管理器来调整其位置和大小。
- 大多数控件在创建之后并不会立即显示在主窗口中，必须经由布局管理器进行布置之后才变成可见的，因此多数控件都要经历创建和布局两个步骤。

- **4. 为控件定义事件处理程序**
- 用户操作会引发事件，如果控件绑定了事件处理程序，则在控件上发生该事件时会调用相应的事件处理程序。

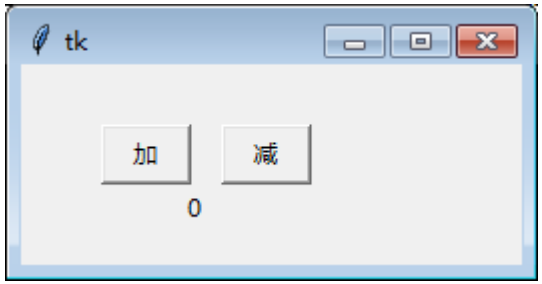
- **5. 进入主事件循环**
- 最后调用主窗口的mainloop()方法，进入主事件循环，准备处理事件。
- 除非用户关闭窗口，否则程序将一直处于主循环中。

- 例：创建图形用户界面，界面中有一个按钮和一个标签，单击按钮对象时，标签的内容会发生改变。

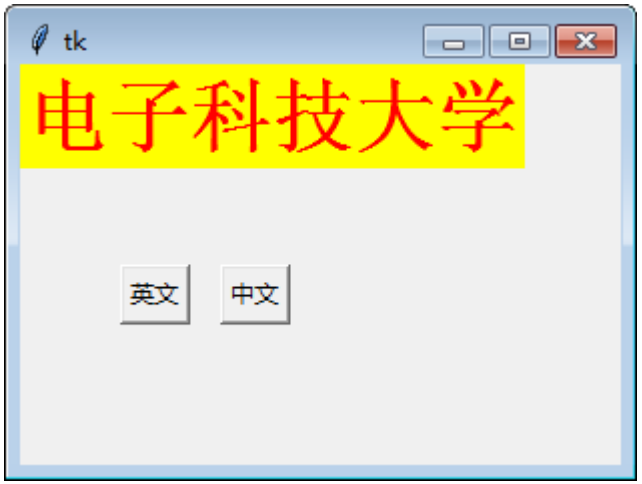


```
def show():  #定义按钮的回调函数
    global c
    c=c+1
    s.config(text=str(c))
from tkinter import *
w=Tk()
w.title("GUI演示")
w.geometry("250x100+100+100")
c=0
#command属性指定按钮的回调函数show
b=Button(w,text="显示",width=8,command=show)
s=Label(w,text=str(c))
b.place(x=100,y=30)   #设置按钮的位置
s.place(x=100,y=60)   #设置标签的位置
w.mainloop()          #进入主事件循环
```

```
def f():
    global c
    c=c+1
    s.config(text=str(c))
def g():
    global c
    c=c-1
    s.config(text=str(c))
from tkinter import *
w=Tk()
w.geometry("250x100+100+100")
c=0
b1=Button(w,text="加",width=5,command=f)
b2=Button(w,text="减",width=5,command=g)
s=Label(w,text=str(c))
b1.place(x=40,y=30)
b2.place(x=100,y=30)
s.place(x=80,y=60)
w.mainloop()
```



- def f():s.config(text="UESTC")
- def g():s.config(text="电子科技大学")
- from tkinter import *
- w=Tk()
- w.geometry("300x200+0+0")
- s=Label(w,fg="red",bg="yellow",font=("Arial",30,"bold"),text="")
- b1=Button(w,text="英文",command=f);b1.place(x=50,y=100)
- b2=Button(w,text="中文",command=g);b2.place(x=100,y=100)
- s.place(x=0,y=0)
- w.mainloop()



13.2 常用控件

- 创建控件之前首先要导入tkinter模块并创建主窗口。
- 例：
- from tkinter import *
- w=Tk()

13.2.1 提示性控件

- 标签（label）和消息（message）都是用来在窗口中显示文本提示信息，但消息对象可以用于显示多行文本。

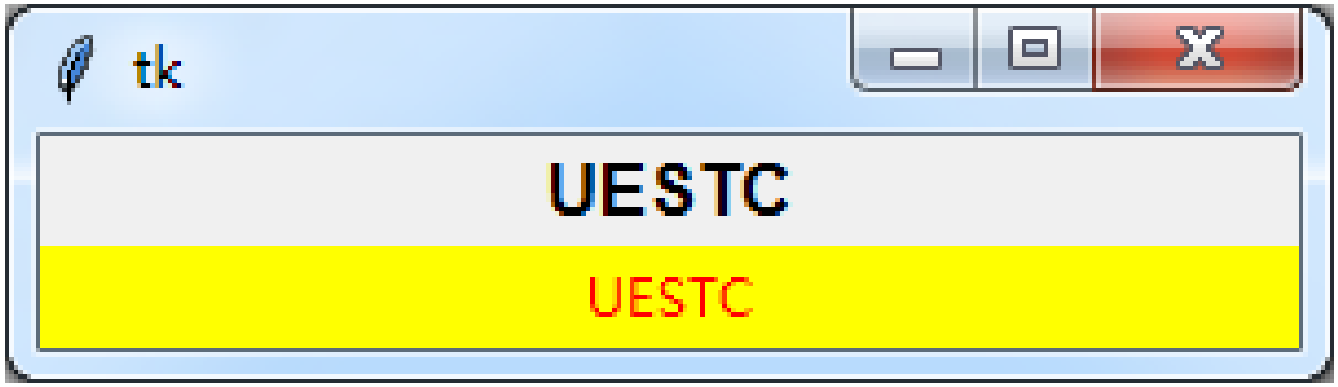
- 1. 标签
- tkinter模块定义了Label类来创建标签控件。
- 创建标签时需要指定其父控件和文本内容，前者由Label构造函数的第一个参数指定，后者用属性text指定。
- 例：
- from tkinter import *
- w=Tk()
- a=Label(w,text="UESTC")
- a.pack()

- 为使控件在窗口中可见，需要调用布局管理器对该控件进行位置安排。
- pack布局管理器简单易用，但不适合进行复杂布局。
- pack有“打包”之意，即所有对象以紧凑的方式布置。
- 主窗口的大小变成刚好可以放置新加入的标签控件，这正是pack的效果。

- pack()方法是在tkinter模块的基类中定义的，而所有控件都是这个基类的子类，从而都继承了这个方法，所以对标签和按钮以及其他各种控件都可以调用pack()方法。

- 标签控件的**font**属性指定文本字体。
- 字体描述使用一个三元组，包含字体名称、尺寸（以磅为单位）和字形名称，常用的英文字体有**Arial**、**Verdana**、**Helvetica**、**Times New Roman**、**Courier New**、**Comic Sans MS**等，也可以用中文字体，如宋体、楷体、仿宋、隶书等。
- 字形名称可以是**normal**、**bold**、**roman**、**italic**、**underline**和**overstrike**等。

- 例：
- from tkinter import *
- w=Tk()
- Label(w,text="UESTC",font=("Arial",12,"bold")).pack()
- Label(w,text="UESTC",bg="yellow",fg="red",width=40)
  .pack()

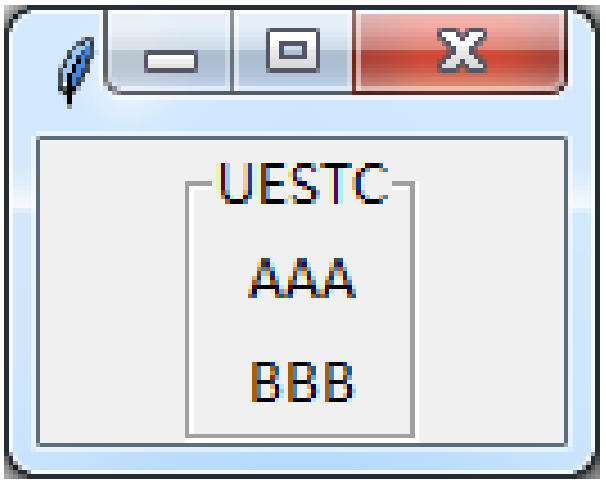


- 说明:
- 语句中的属性**bg**（或**background**）、**fg**（或**foreground**）和**width**分别表示标签文本的背景颜色、文本颜色（前景颜色）和标签的宽度。

- 说明:
- 该语句采用了将对象创建和对象方法调用结合在一起的方式。
- 先调用构造函数Label()创建一个标签对象，然后直接对这个对象调用方法pack()，而不是先定义一个对象，然后通过对象来调用对象的方法。
- 对象创建与对象方法调用合并的写法不适合需要多次引用一个对象的场合。

- **2. 标签框架**
- 标签框架是一个带标签的矩形框，它是一个容器控件，其中能容纳其他的控件。
- **tkinter模块提供LabelFrame类来创建标签框架对象。**

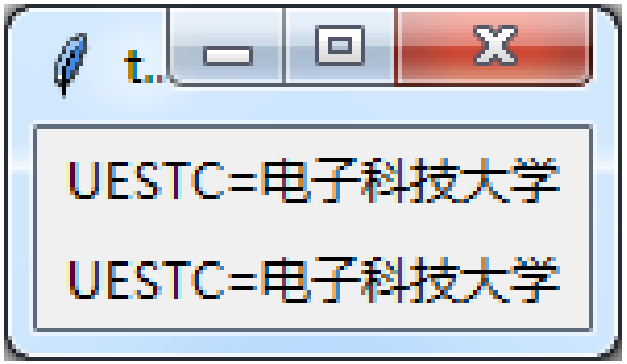
- 例：创建标签框架，并在其中创建标签对象。
- from tkinter import *
- w=Tk()
- f=LabelFrame(w,text="UESTC")
- f.pack()
- s=Label(f,text="AAA")
- s.pack()
- s=Label(f,text="BBB")
- s.pack()



- **3. 消息**
- 消息和标签的用法基本一样。
- 消息与标签的区别：消息可以显示多行文本。
- 如果不想让消息换行的话，可以指定足够大的宽度。
- 消息的aspect属性指定消息的宽高比例。在默认情况下，消息的宽高比是150，即消息的宽度是高度的150%。假如将aspect属性设置为400，即宽为高的4倍。

- 例：
- from tkinter import *
- w=Tk()
- m=Message(w,text="UESTC=电子科技大学")
- m.pack()
- s=Label(w,text="UESTC=电子科技大学")
- s.pack()

- 例：
- from tkinter import *
- w=Tk()
- m=Message(w,text="UESTC=电子科技大学",width=200)
- m.pack()
- s=Label(w,text="UESTC=电子科技大学")
- s.pack()



13.2.2 按钮控件

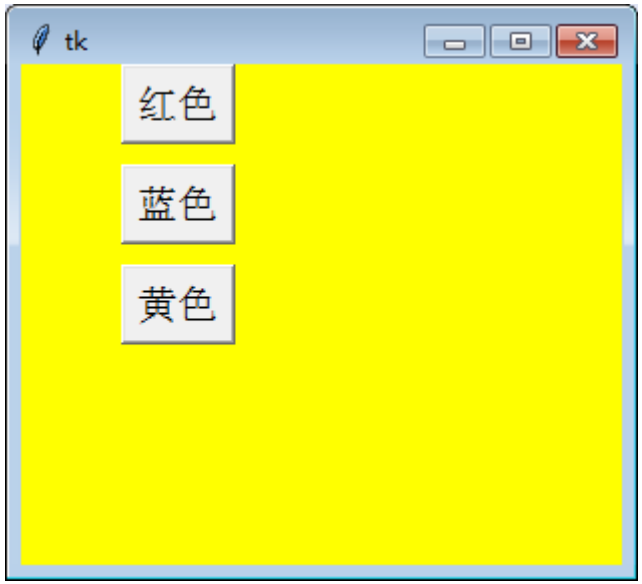
- 按钮（button）也称命令按钮（command button），它是用户命令程序执行某项操作的基本手段。
- 按钮控件的属性command用于指定按钮的事件处理函数，将按钮与某个函数或方法关联起来。
- 用鼠标单击按钮时就执行command属性指定的函数或方法。

- 例：
- from tkinter import *
- w=Tk()
- b=Button(w,text="Exit",command=w.quit)
- b.pack()
- w.mainloop()

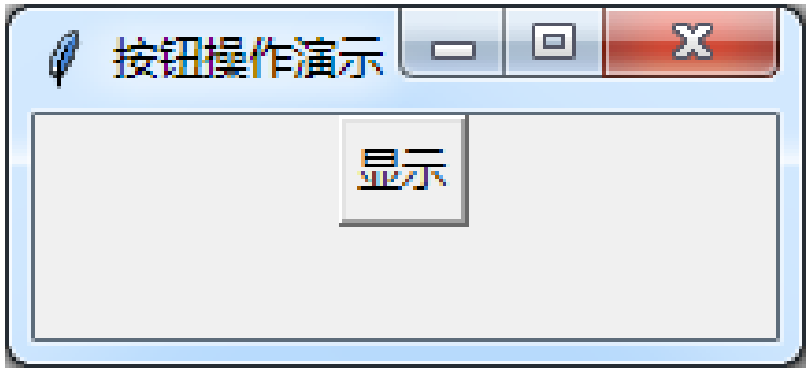
- 说明:
- command将按钮与主窗口w的内置方法quit()相关联，其功能是退出主循环。
- 注意：传递给command属性的是函数对象，函数名后不能加括号。

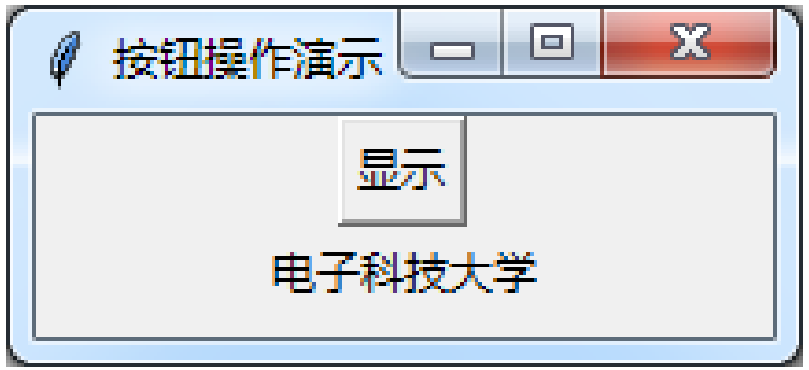
- 说明:
- 程序运行后，可以看到Python解释器的“>>>”提示符没有了，表明现在tkinter程序接管了控制权。
- 单击“Exit”按钮，可以看到又回到了Python解释器的提示符状态，这就是w.quit()方法的作用。

```
from tkinter import *
w=Tk()
w.geometry("300x250+0+0")
t=("Arial",15,"normal")
def f1():w["bg"]="red"
def f2():w["bg"]="blue"
def f3():w["bg"]="yellow"
b1=Button(w,text="红色",font=t,command=f1)
b2=Button(w,text="蓝色",font=t,command=f2)
b3=Button(w,text="黄色",font=t,command=f3)
b1.place(x=50,y=0)
b2.place(x=50,y=50)
b3.place(x=50,y=100)
w.mainloop()
```

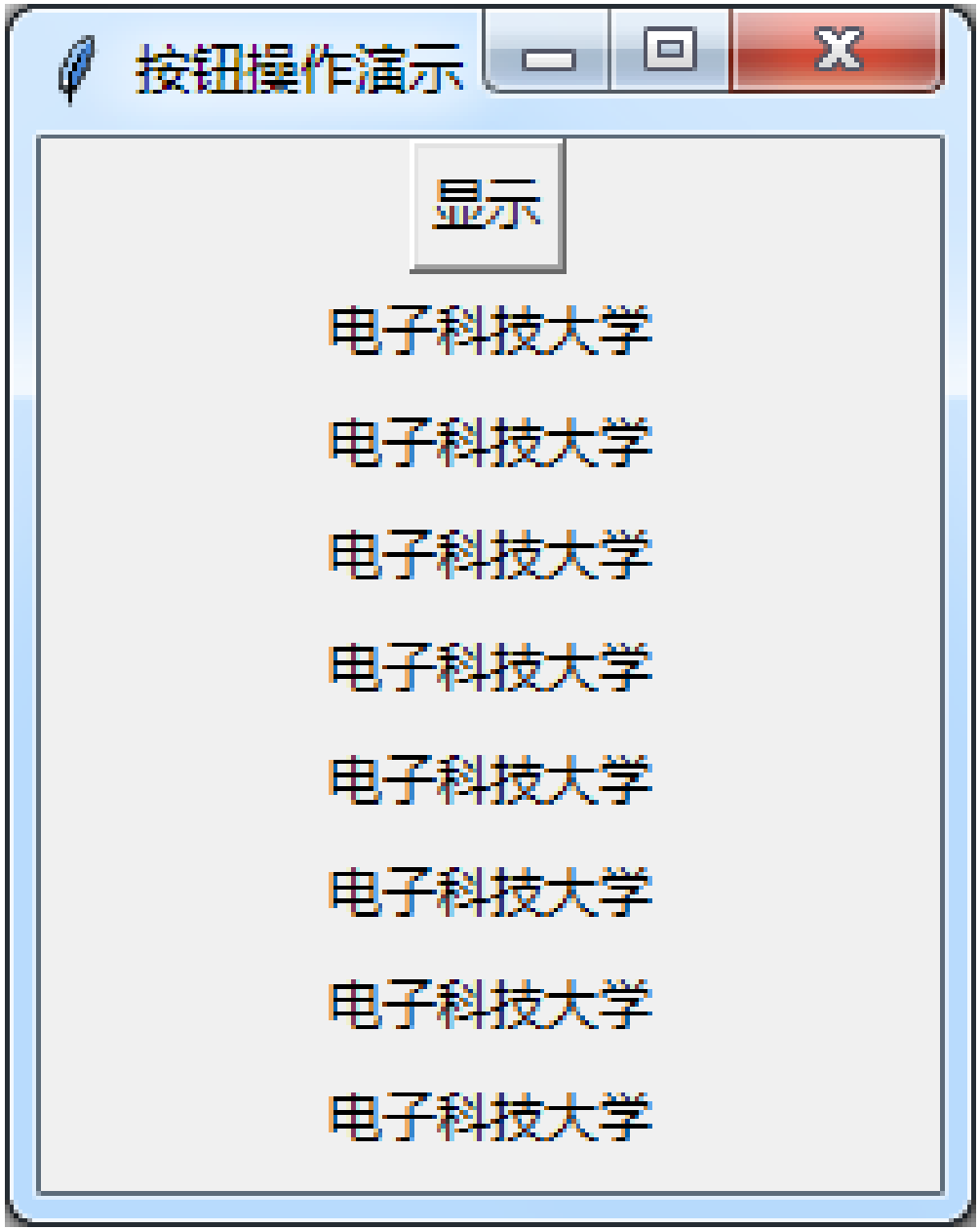


- 例：
- 在主窗口中创建一个“显示”按钮，单击该按钮时在主窗口中显示“电子科技大学”。





```
def btnf():
    Message(w,text='电子科技大学',width=200).pack()
from tkinter import *
w=Tk()
w.geometry("200x60")
w.title("按钮操作演示")
btn=Button(w,text='显示',command=btnf)
btn.pack()
w.mainloop()
```

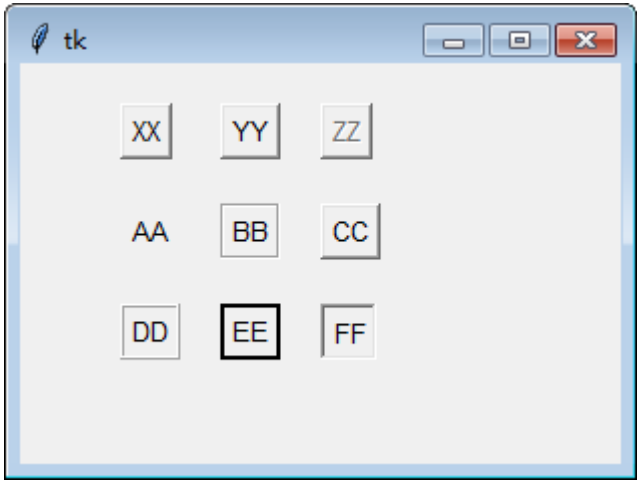

按钮操作演示
显示
电子科技大学
电子科技大学
电子科技大学
电子科技大学
电子科技大学
电子科技大学
电子科技大学
电子科技大学

```
from tkinter import *
def f():
    y["text"]="大家好，我是成电小哥！"
root=Tk()
root.title("打招呼测试")
root.geometry("200x100+100+100")
x=Button(root,text="打招呼",command=f)
x.pack(side=RIGHT)
y=Label(root)
y.pack(side=LEFT,fill=X)
root.mainloop()
```

- 按钮控件的属性:
- width:宽度属性;
- height:高度属性
- fg:文本颜色属性;
- bg:背景颜色属性;
- bd:边框大小属性borderwidth，默认为2个像素;

- **state**:状态属性，有正常NORMAL、激活ACTIVE 、禁用DISABLED三种状态；
- **relief**:边框3D效果属性，默认是FLAT，有FLAT、GROOVE、RAISED、RIDGE、SOLID、SUNKEN等效果设置。

```
from tkinter import *
w=Tk()
w.geometry("300x200+0+0")
t=("Arial",10,"normal");
b1=Button(w,text="XX",font=t,state=NORMAL)
b2=Button(w,text="YY",font=t,state=ACTIVE)
b3=Button(w,text="ZZ",font=t,state=DISABLED)
b4=Button(w,text="AA",font=t,relief=FLAT)
b5=Button(w,text="BB",font=t,relief=GROOVE)
b6=Button(w,text="CC",font=t,relief=RAISED)
b7=Button(w,text="DD",font=t,relief=RIDGE)
b8=Button(w,text="EE",font=t,relief=SOLID)
b9=Button(w,text="FF",font=t,relief=SUNKEN)
b1.place(x=50,y=20);b2.place(x=100,y=20)
b3.place(x=150,y=20);b4.place(x=50,y=70)
b5.place(x=100,y=70);b6.place(x=150,y=70)
b7.place(x=50,y=120);b8.place(x=100,y=120)
b9.place(x=150,y=120)
w.mainloop()
```



13.2.3 选择性控件

- 1.复选框（check button）
- 用来提供一些选项供用户进行选择，可以选择多项。
- 例如，学生的兴趣爱好就可以使用复选框实现。
- 在外观上由一个小方框和一个相邻的描述性标题组成。

- 未选中时，小方框为空白，选中时在小方框中打钩（√），再次选择一个已打钩的复选框将取消选择。
- 对复选框的选择操作一般是用鼠标单击小方框或标题。

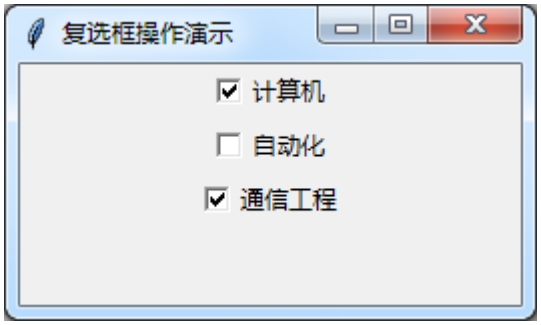
- tkinter模块的Checkbutton类用于创建复选框控件。
- 例：
- >>>Checkbutton(w,text="Python").pack()
- 在实际应用中，通常将多个复选框组合为一组，为用户提供多个相关的选项，用户可以从中选择一个或多个选项，当然也可以不选。

- 如果程序中需要查询和设置选项的状态，可以使用variable属性将复选框与一个IntVar或StringVar类型的控制变量关联。
- 格式:
- **>>>v=IntVar()**
- **>>>Checkbutton(w,text="Python",variable=v).pack()**

- 程序中可以通过v.get()和v.set()来查询或设置复选框的状态。
- 处于选中状态时，对应1或字符1；
- 处于未选中状态时，对应0或字符0。

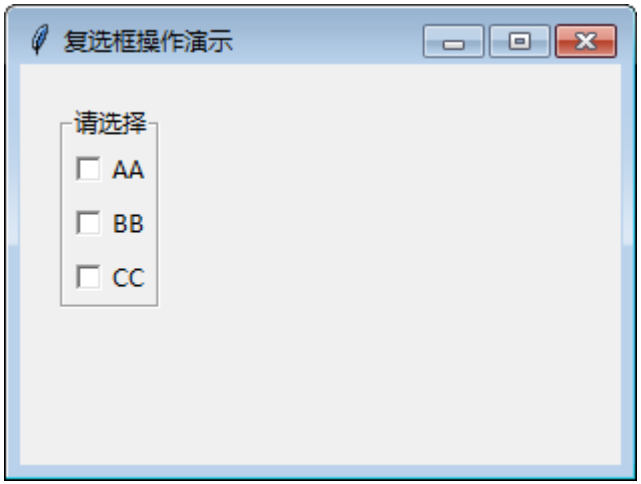
- 例：复选框组包括三个复选框，要求将一个整数与复选框的状态关联，每次单击复选框，显示当前选中的内容。利用复选框的command属性指定复选框的事件处理函数。

```
def callChk():
    str="选择了"
    if v1.get()==1:
        str+="计算机！"
    if v2.get()==1:
        str+="自动化！"
    if v3.get()==1:
        str+="通信工程！"
    Label(w,text=str).pack()
from tkinter import *
w=Tk()
w.geometry("250x120")
w.title("复选框操作演示")
v1=IntVar()
v2=IntVar()
v3=IntVar()
v1.set(1)    #设置复选框的状态
v2.set(0)
v3.set(1)
Checkbutton(w,variable=v1,text='计算机',command=callChk).pack()
Checkbutton(w,variable=v2,text='自动化',command=callChk).pack()
Checkbutton(w,variable=v3,text='通信工程',command=callChk).pack()
w.mainloop()
```



- 图中的各复选框选项排列得不太整齐，这是因为pack()方法在布局时默认采用了居中对齐方式。（布局管理器的其他对齐方式，详见下节）
- 在实际图形用户界面设计中，为了表示一组复选框的相关性，通常用一个框架将它们组合起来（参见下节）。

```
def f():
    str=""
    if v1.get()==1:str+="AA"
    if v2.get()==1:str+="BB"
    if v3.get()==1:str+="CC"
    s.config(text="你选择的是： "+str)
from tkinter import *
w=Tk()
w.geometry("300x200+0+0")
w.title("复选框操作演示")
c=LabelFrame(w,text="请选择");c.place(x=20,y=20)
v1=IntVar();v2=IntVar();v3=IntVar()
c1=Checkbutton(c,variable=v1,text='AA',command=f)
c2=Checkbutton(c,variable=v2,text='BB',command=f)
c3=Checkbutton(c,variable=v3,text='CC',command=f)
c1.pack();c2.pack();c3.pack()
s=Label(w,text="");s.place(x=80,y=20)
w.mainloop()
```



```
from tkinter import *
def f():
    i=v1.get()+v2.get()+v3.get()
    m=""
    if(i==1):m="你选择的是:"+x[0]
    elif(i==2):m="你选择的是:"+x[1]
    elif(i==3):m="你选择的是:"+x[0]+','+x[1]
    elif(i==4):m="你选择的是:"+x[0]
    elif(i==5):m="你选择的是:"+x[0]+','+x[2]
    elif(i==6):m="你选择的是:"+x[1]+','+x[2]
    elif(i==7):m="你选择的是:"+x[0]+','+x[1]+','+x[2]
    s.config(text=m)
w=Tk()
x=["游泳","唱歌","编程"]
v1=IntVar()
v2=IntVar()
v3=IntVar()
Label(w,text="请选择你的兴趣爱好：").pack()
t=Frame(w,bd=4,relief=GROOVE)
t.pack()
r1=Checkbutton(t,variable=v1,onvalue=1,text=x[0],command=f)
r2=Checkbutton(t,variable=v2,onvalue=2,text=x[1],command=f)
r3=Checkbutton(t,variable=v3,onvalue=4,text=x[2],command=f)
r1.pack()
r2.pack()
r3.pack()
s=Label(w)
s.pack()
w.mainloop()
```

- **2.单选按钮（radio button）**
- 复选框和单选按钮都是用来提供一些选项供用户选择，这些选项有选中或未选中两种状态。
- 两者的区别是，复选框主要适合多选多的情况，单选按钮适合多选一的情况，同组的单选按钮在任意时刻只能有一个被选中，每当换选其他单选按钮时，原先选中的单选按钮即被取消。
- 例如，选择学生的“性别”就适合用单选按钮。

- 单选按钮的外观是一个小圆圈加上相邻的描述性标题.
- 未选中时，小圆圈内是空白；
- 选中时，小圆圈中出现一个圆点。
- 对单选按钮的选择操作是用鼠标单击小圆圈或标题。

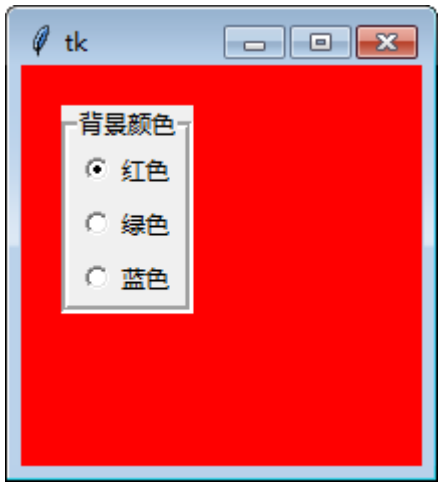
- tkinter模块提供的Radiobutton类可用于创建单选按钮。
- 例：
- >>>Radiobutton(w,text="Python").pack()

- 在实际应用中都是将若干个相关的单选按钮组合成一个组，使得每次只能有一个单选按钮被选中。
- 为了实现单选按钮的组合，可以先创建一个IntVar或StringVar类型的控制变量，然后将同组的每个单选按钮的variable属性都设置成该控制变量。
- 由于多个单选按钮共享一个控制变量，而控制变量每次只能取一个值，所以选中一个单选按钮就会导致取消另一个。

- 为了在程序中获取当前被选中的单选按钮的信息，可以为同组中的每个单选按钮设置value属性的值。
- 当选中一个单选按钮时，控制变量即被设置为它的value值，程序中即可通过控制变量的当前值来判断是哪个单选按钮被选中了。

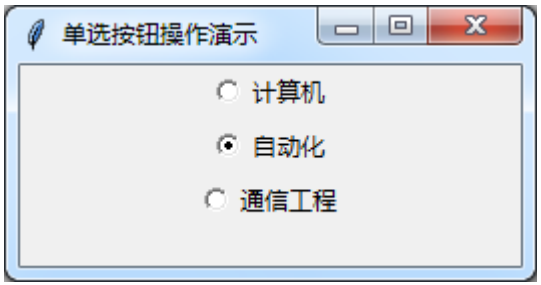
- 注意:
- value属性的值应当与控制变量的类型匹配。如果控制变量是IntVar类型，则应为每个单选按钮赋予不同的整数值；
- 如果控制变量是StringVar类型，则应为每个单选按钮赋予不同的字符串值。

```
def f():
    s=["red","green","blue"]
    i=v.get();w["bg"]=s[i]
from tkinter import *
w=Tk()
w.geometry("200x200+0+0");w["bg"]="red"
v=IntVar()
c=LabelFrame(w,text="背景颜色",bd=4,relief=GROOVE)
c.place(x=20,y=20)
r1=Radiobutton(c,variable=v,value=0,text="红色",command=f)
r2=Radiobutton(c,variable=v,value=1,text="绿色",command=f)
r3=Radiobutton(c,variable=v,value=2,text="蓝色",command=f)
r1.pack();r2.pack();r3.pack()
w.mainloop()
```



- 例：
- 创建一个包含三个单选按钮的单选按钮组，并在主窗口中输出单选按钮的选定状态。

```
def callRad():
    Message(w,text=v.get()).pack()
from tkinter import *
w=Tk()
w.geometry("250x100")
w.title("单选按钮操作演示")
v=StringVar()
v.set('1')
lst=['计算机','自动化','通信工程']
for i in range(3):
    Radiobutton(w,variable=v,text=lst[i],\
        value=str(i),command=callRad).pack()
w.mainloop()
```

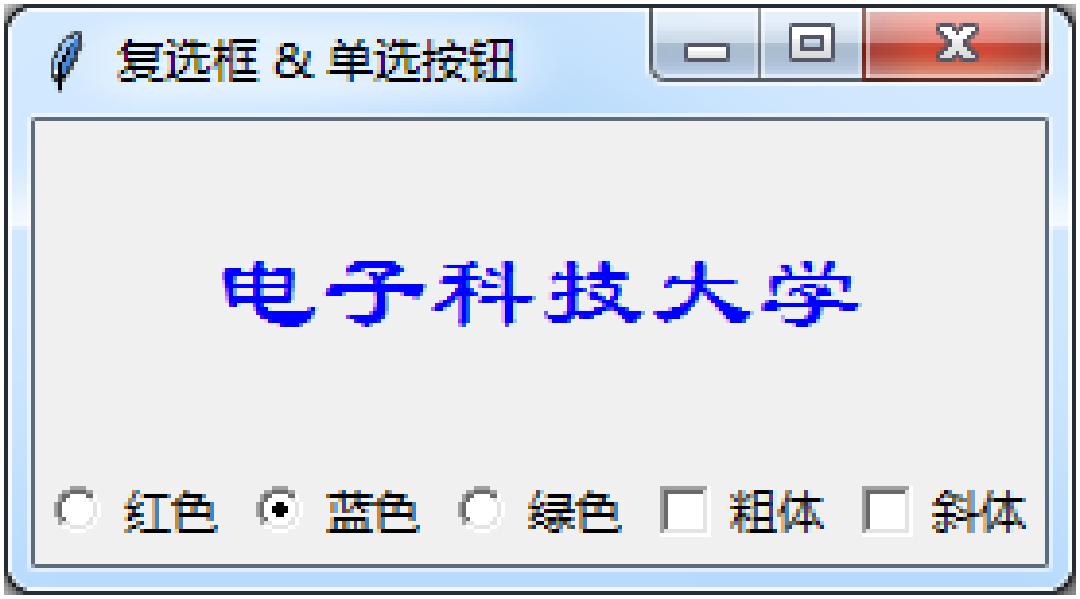


```
import tkinter as tk
def f():
    i=v.get()
    s.config(text="你选择的是"+str(i)+"，即"+x[i])
w=tk.Tk()
x=["男","女"]
v=tk.IntVar()
tk.Label(w,text="请选择性别：").pack()
t=tk.Frame(w,bd=4,relief=tk.GROOVE)
t.pack()
r1=tk.Radiobutton(t,variable=v,value=0,text=x[0],command=f)
r2=tk.Radiobutton(t,variable=v,value=1,text=x[1],command=f)
r1.pack()
r2.pack()
s=tk.Label(w)
s.pack()
w.mainloop()
```

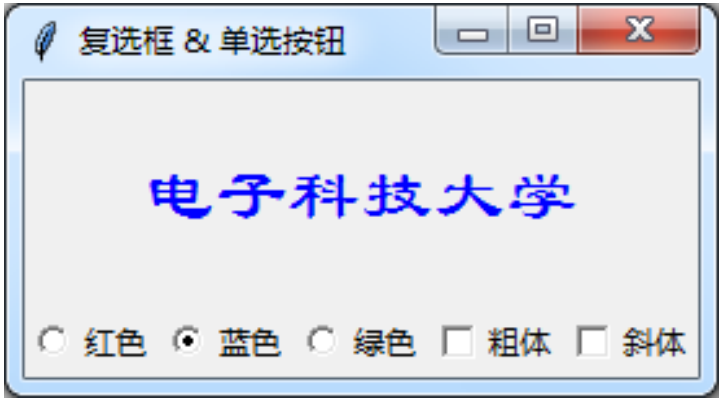
- 例：选择相应单选按钮时，将窗口背景设置成相应颜色。

```
def callb():
    w.config(bg=tf[v.get()])
from tkinter import *
w=Tk()
w.title("改变窗口背景颜色")
w.geometry("250x100")
v=IntVar()
v.set(2) #将第三个单选按钮设为默认按钮
f=Frame(w,bd=4,relief=GROOVE) #建立框架
f.pack()
tf=['red','blue','yellow']
for n in range(len(tf)):
    r=Radiobutton(f,variable=v,text=tf[n],value=n,command=callb)
    r.grid(row=n,column=1,sticky=W)  #靠左放
w.mainloop()
```

- 例：用标签显示一行文字，通过单选按钮选择文字颜色，通过复选框选择字形，程序运行界面如图所示。



```
def colorChecked():   #改变标签文本颜色
    lbl.config(fg=color.get())
def typeChecked():    #改变文本字形
    if typeBlod.get()==1 and typeItalic.get()==0:
        lbl.config(font=("隶书",20,"bold"))
    elif typeItalic.get()==1 and typeBlod.get()==0:
        lbl.config(font=("隶书",20,"italic"))
    elif typeBlod.get()==1 and typeItalic.get()==1:
        lbl.config(font=("隶书",20,"bold","italic"))
    else:
        lbl.config(font=("隶书",20))
from tkinter import *
w=Tk()
w.title("复选框 & 单选按钮")
lbl=Label(w,text="电子科技大学",height=3,font=("隶书",20),fg='blue')
lbl.pack()
color=StringVar()
color.set('blue')     #设置默认选项
Radiobutton(w,text="红色",variable=color,value="red",command=colorChecked).pack(side=LEFT)
Radiobutton(w,text="蓝色",variable=color,value="blue",command=colorChecked).pack(side=LEFT)
Radiobutton(w,text="绿色",variable=color,value="green",command=colorChecked).pack(side=LEFT)
typeBlod=IntVar()
typeItalic=IntVar()
Checkbutton(w,text="粗体",variable=typeBlod,command=typeChecked).pack(side=LEFT)
Checkbutton(w,text="斜体",variable=typeItalic,command=typeChecked).pack(side=LEFT)
w.mainloop()
```

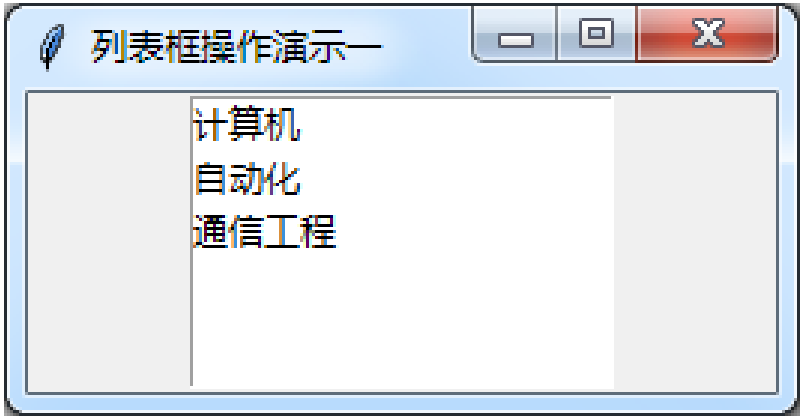


- **3.列表框（list box）**
- 列表框可以包含一个或多个选项供用户选择。
- **tkinter模块的Listbox类用于创建列表框控件。**

- 使用列表框的insert()方法来向列表框中添加一个选项。
- insert()方法有两个参数:一个为添加项的索引值，另一个为添加的项目。
- 索引值有两个特殊的值:ACTIVE和END。
- ACTIVE是向当前选中的项目前插入一个（即使用当前选中的索引作为插入位置）；
- END是向列表框的最后一项插入一项。

- 例：创建一个列表框，向其中添加三个选项。

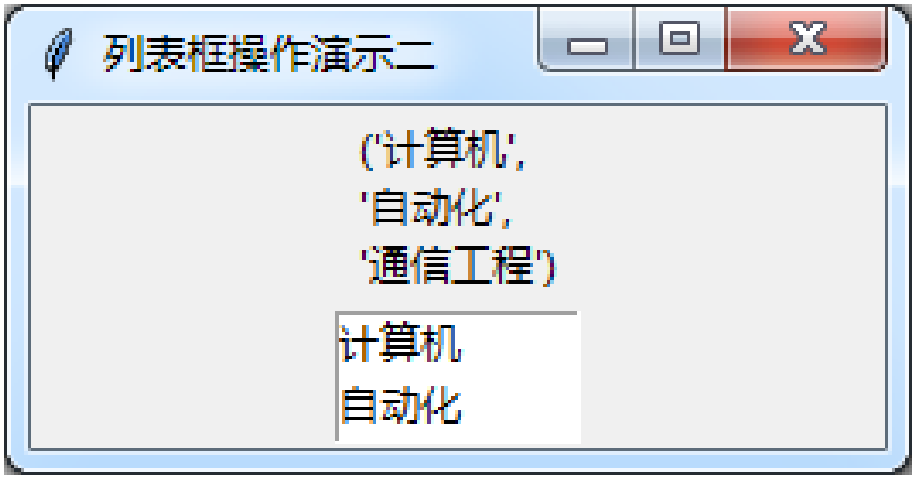
- from tkinter import *
- w=Tk()
- w.geometry("250x100")
- w.title("列表框操作演示一")
- lb=Listbox(w)
- for item in ['计算机','自动化','通信工程']:
- 　　lb.insert(END,item)
- lb.pack()
- w.mainloop()



- 使用列表框的delete()方法可以删除选项。
- delete()方法也有两个参数：第一个为开始的索引值，第二个为结束的索引值。
- 如果不指定第二个参数，则只删除第一个索引项。
- 例：
- >>>lb.delete(1,3)
- #删除第2~4项，对应的索引值为1，2，3。
- >>>lb.delete(0,END)
- #指定第一个索引值0和最后一个项目索引值为END，即删除全部项目。

- 调用size()方法得到当前列表框中的项目个数；
- selection_set()方法选中指定的项目；
- get()方法返回指定索引的项目；
- curselection()方法返回当前选项的索引；
- selection_includes()判断一个选项是否被选中。
- 列表框可以与变量绑定，通过变量来修改项目。

- 例：列表框操作演示二。

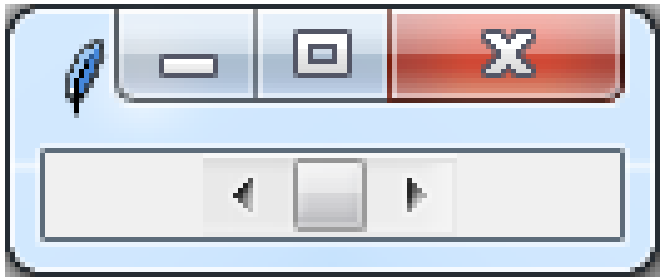


- from tkinter import *
- w=Tk()
- w.geometry("250x100")
- w.title("列表框操作演示二")
- v=StringVar()
- lb=Listbox(w,listvariable=v,width=10,heigh=3)
- v.set(('计算机','自动化','通信工程')) #改变列表框的项目，元组与选项对应
- Message(w,text=v.get()).pack() #显示当前列表中的项目
- lb.pack()
- w.mainloop()

```
from tkinter import *
def f(event):
    w["bg"]=t.get(t.curselection())
w=Tk()
w.geometry("250x100")
t=Listbox(w,width=10,heigh=3)
t.pack()
ys=('red','green','blue','yellow','cyan','pink')
for i in ys:
    t.insert(END,i)
t.bind('<Double-Button-1>',f)
w.mainloop()
```

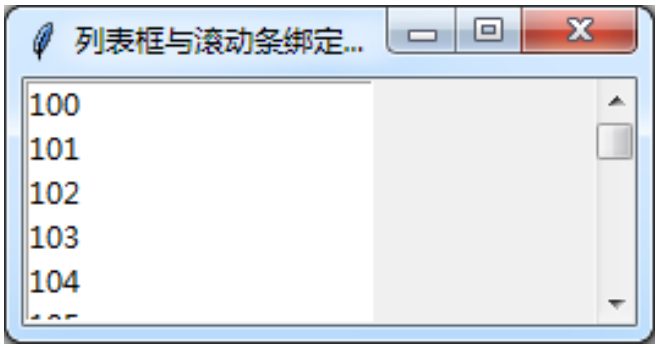
- **4. 滚动条（scrollbar）**
- 滚动条用于移动选项的可见范围，可以单独使用，但最多的还是与列表框、多行文本框、画布等其他控件配合使用。
- **tkinter模块的Scrollbar类用于创建滚动条控件，滚动条分垂直滚动条和水平滚动条两种，默认创建垂直滚动条。**

- 例：创建水平滚动条，并通过set()方法来设置滑块的位置。
- from tkinter import *
- w=Tk()
- sl=Scrollbar(w,orient=HORIZONTAL) #创建水平滚动条
- sl.set(0.5,1)
- sl.pack()
- w.mainloop()



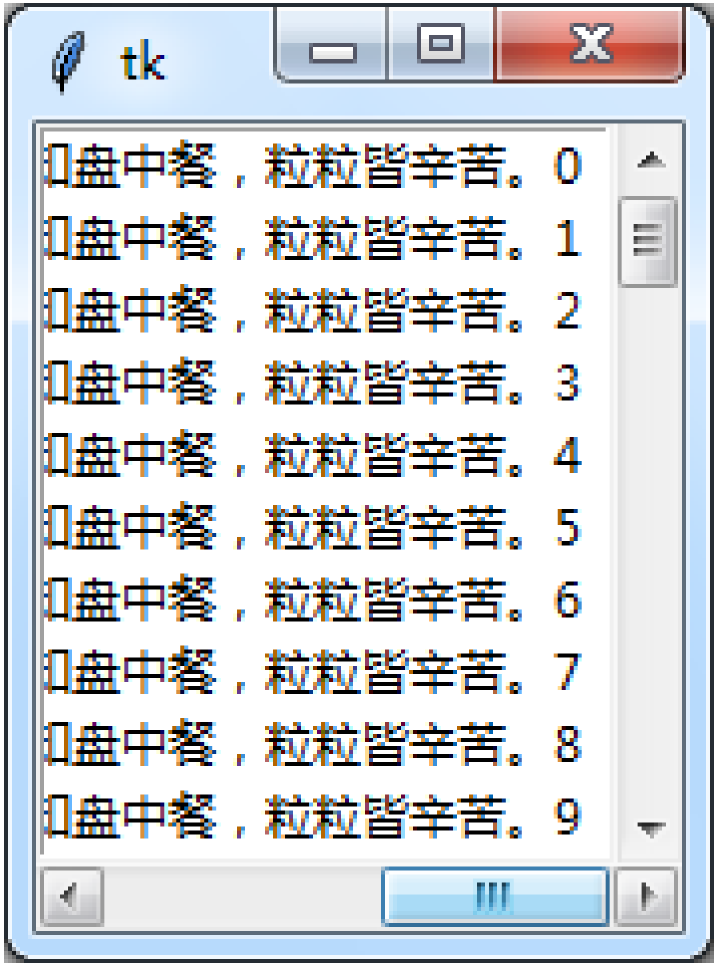
- 通过set()将值设置为(0.5,1)，即滑块占整个滚动条的一半。
- 单独使用滚动条比较少见，大部分应用还是与其他控件绑定使用。

- 例：列表框与滚动条绑定使用。
- 分析：为了在列表框中添加垂直滚动条，需要做两项操作:
- 一是指定列表框的yscrollbar属性的回调函数为滚动条的set()方法；
- 二是指定滚动条的command属性的回调函数是列表框的yview()方法。



```
from tkinter import *
w=Tk()
w.geometry("250x100")
w.title("列表框与滚动条绑定操作")
lb=Listbox(w)
sl=Scrollbar(w)
sl.pack(side=RIGHT,fill=Y)#side指定滚动条居右，fill指定充满整个剩余区域
#指定列表框的yscrollbar的回调函数为滚动条的set()方法
lb['yscrollcommand']=sl.set
for i in range(100,1000):
    lb.insert(END,str(i))
lb.pack(side=LEFT)  #side指定列表框居左
#指定滚动条的command的回调函数是列表框的yview()方法。
sl['command']=lb.yview
w.mainloop()
```

- 当列表框改变时，滚动条调用set()方法改变滑块的位置；
- 当滚动条改变了滑块的位置时，列表框调用yview()方法以显示新的列表选项。

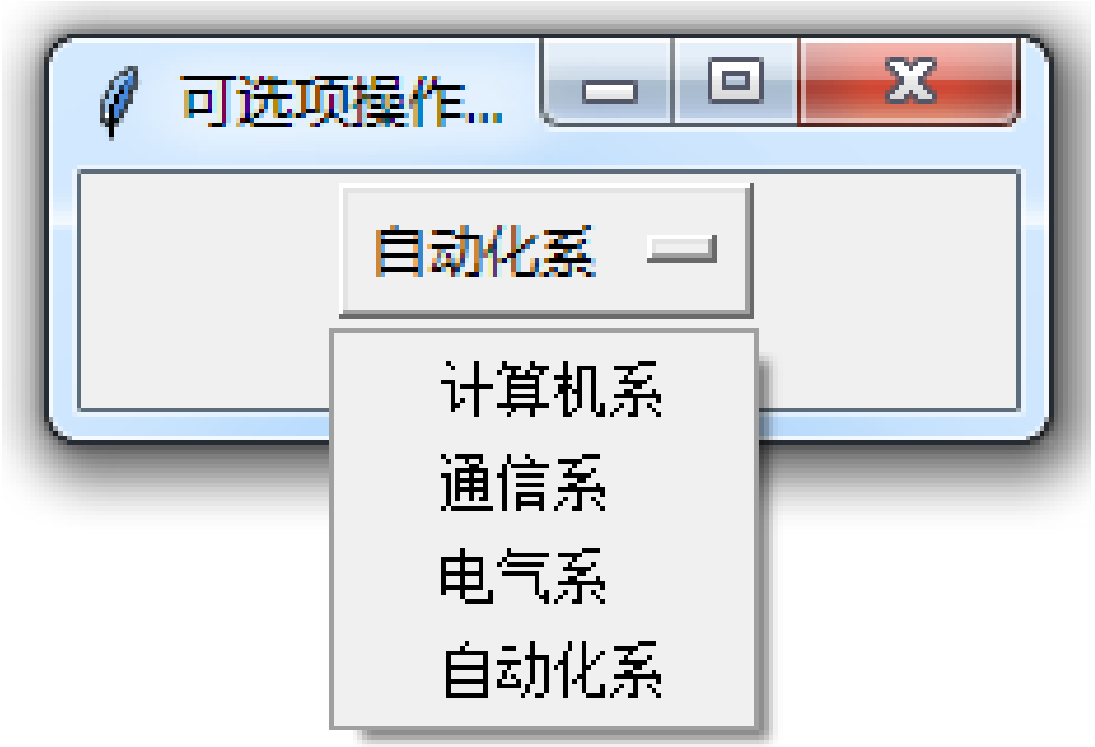

tk
盘中餐，粒粒皆辛苦。0
盘中餐，粒粒皆辛苦。1
盘中餐，粒粒皆辛苦。2
盘中餐，粒粒皆辛苦。3
盘中餐，粒粒皆辛苦。4
盘中餐，粒粒皆辛苦。5
盘中餐，粒粒皆辛苦。6
盘中餐，粒粒皆辛苦。7
盘中餐，粒粒皆辛苦。8
盘中餐，粒粒皆辛苦。9

```
from tkinter import *
w=Tk()
sh=Scrollbar(w,orient=HORIZONTAL)
sh.pack(side=BOTTOM,fill=X)
sv=Scrollbar(w)
sv.pack(side=RIGHT,fill=Y)
t=Listbox(w,xscrollcommand=sh.set,yscrollcommand=sv.set)
for i in range(60):
    t.insert(END,"锄禾日当午，汗滴禾下土；谁知盘中餐，粒粒皆辛苦。
"+str(i))
t.pack(side=LEFT,fill=BOTH)
sh.config(command=t.xview)
sv.config(command=t.yview)
w.mainloop()
```

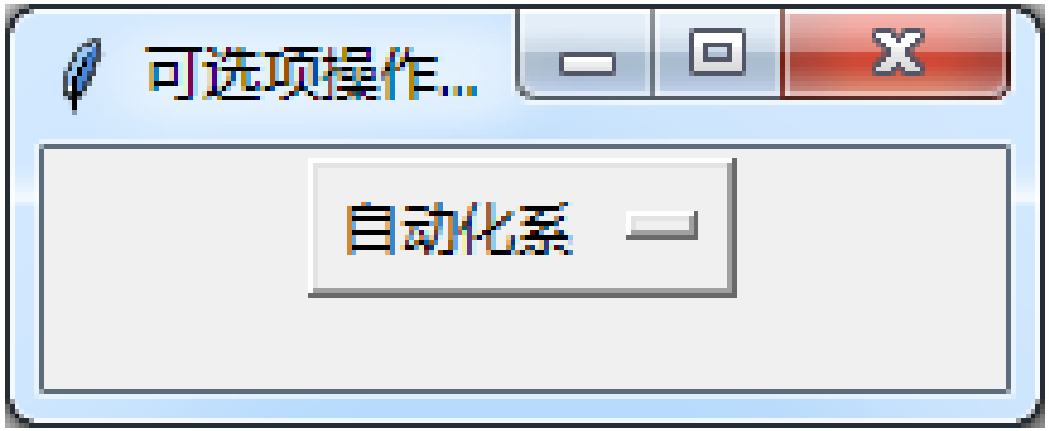
- 5.可选项（option menu）
- 可选项提供一个选项列表，平时是收拢状态，单击可选项按钮时，打开列表供用户选择，被选中的选项显示在按钮中。
- tkinter模块提供OptionMemu类来创建可选项。

- 创建时需要两个必要的参数:
- 一是与当前值绑定的变量，通常为一个StringVar类型的变量;
- 另一个是提供可选的项目列表。
- 当可选项与变量绑定后，直接使用给变量赋值的方法即可改变当前的值。

- 例：可选项操作演示。

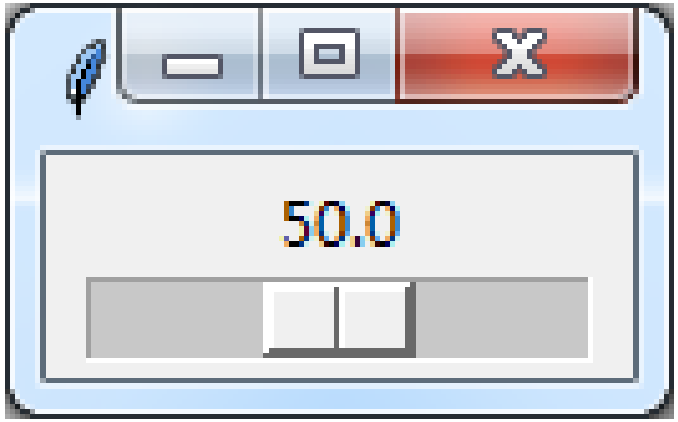


- from tkinter import *
- w=Tk()
- w.geometry("200x50")
- w.title("可选项操作演示")
- v=StringVar(w)
- v.set('自动化系')
- om=OptionMenu(w,v,'计算机系','通信系','电气系','自动化系')
- om.pack()
- print(v.get())
- w.mainloop()

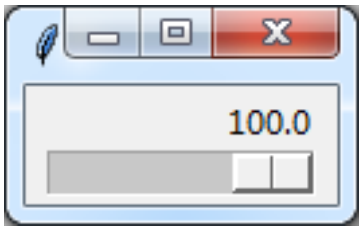


- 6. 刻度条（scale）
- 刻度条用于在指定的范围内，通过移动滑块来选择的数值。
- tkinter模块提供scale类来创建刻度条。
- 创建时需要为刻度条指定最小值、最大值和移动的步距，也可以和变量绑定，还可以使用回调函数打印当前的值。

- 例：创建一个水平刻度条，最小值为0，最大值为100，步距为0.1。

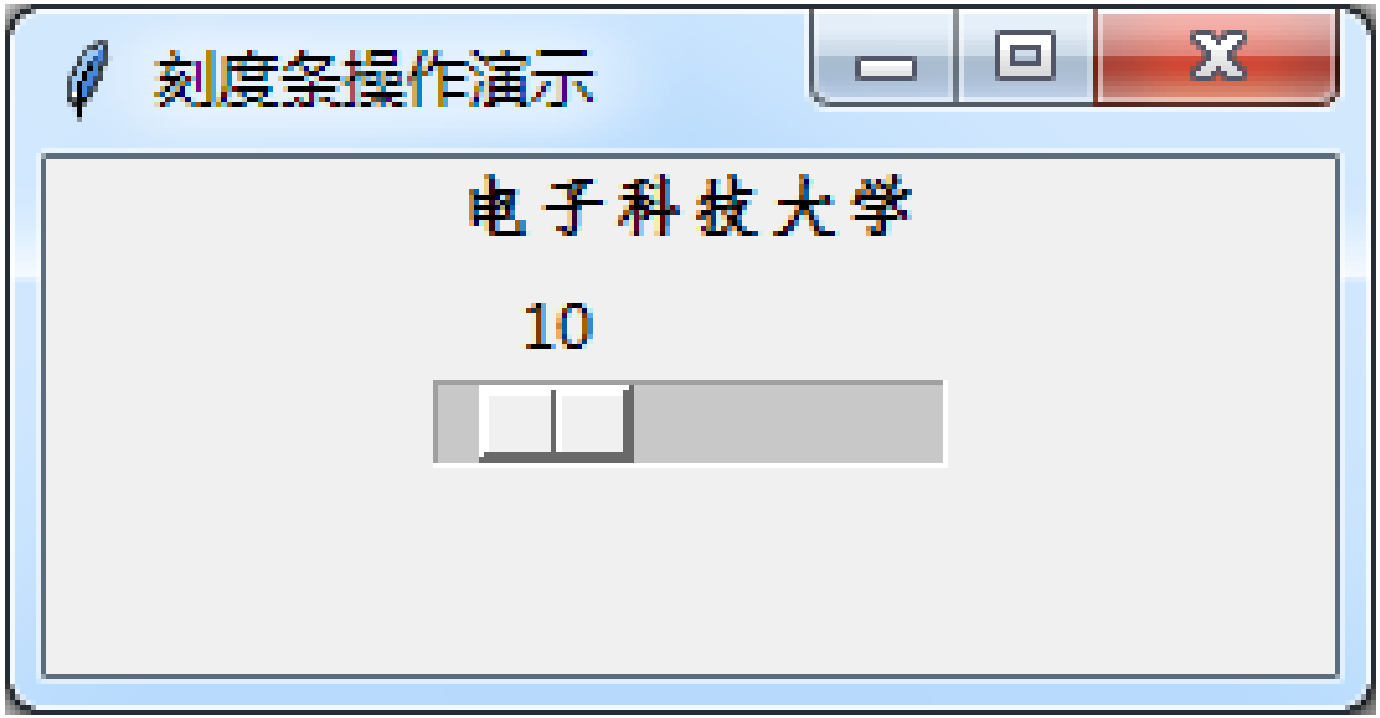


- from tkinter import *
- w=Tk()
- def show(text):
- print("v=",v.get())
- v=StringVar()
- scl=Scale(w,from_=0,to=100,resolution=0.1,orient=HORIZONTAL,variable=v,command=show)
- scl.set(50)
- scl.pack()
- w.mainloop()



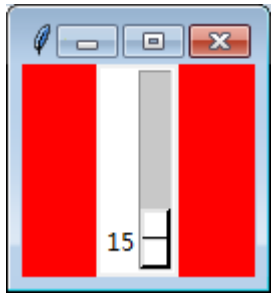
- 程序中的回调函数有一个参数，这个值是滑块的当前值，每移动一个步距就会调用一次函数。
- 注意：from_选项的使用方式，在其后添加了下画线(_)，避免与关键字from冲突。

- 例：通过拖动刻度条来改变窗口文字的大小。

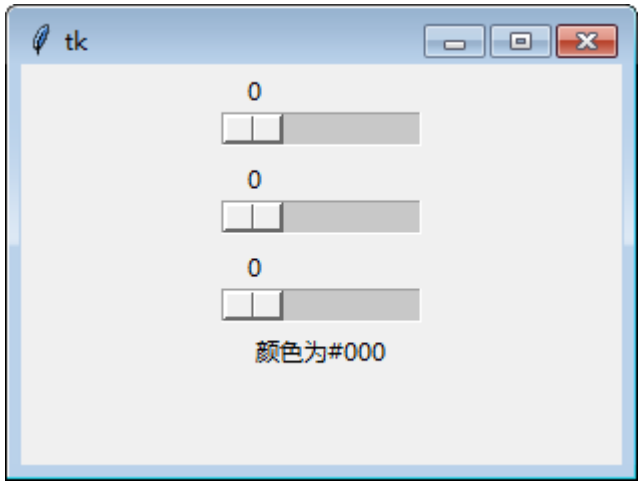


```
def change(value):     #改变文字大小的函数
    lbl.config(font=('仿宋',sl.get(),'bold')) #改变字体大小
from tkinter import *
w=Tk()
w.geometry('250x100')
w.title("刻度条操作演示")
lbl=Label(w,text='Python程序设计!',font=('仿宋',10,'bold'))
lbl.pack()
sl=Scale(w,from_=5,to=50,orient=HORIZONTAL,command=change)
sl.set(10)  #设置滑块起始位置
sl.pack()
w.mainloop()
```

```
def f(text):
    t=hex(v.get())
    w["bg"]="#"+t[2]+"00"
    s["bg"]="#f"+t[2]+"f"
from tkinter import *
w=Tk()
v=IntVar()
s=Scale(w,from_=0,to=15,resolution=1,orient=VERTICAL)
s.config(variable=v,command=f)
s.set(50)
s.pack()
w.mainloop()
```

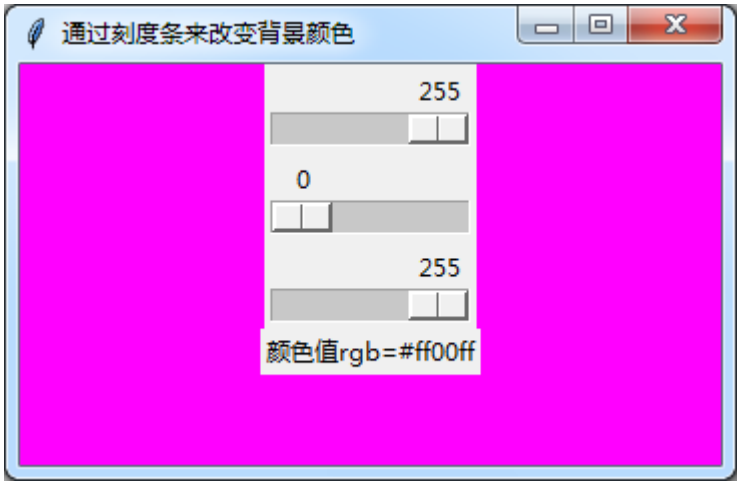


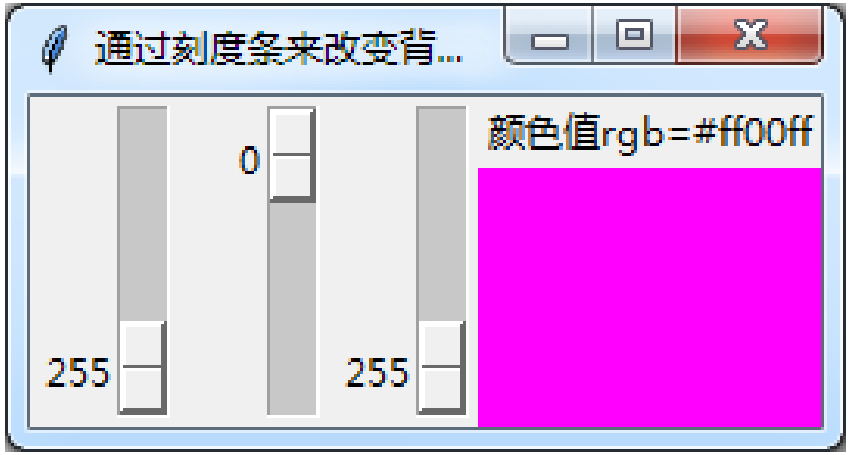
```
def f(text):
    r=hex(v1.get());g=hex(v2.get());b=hex(v3.get());
    rgb="#"+r[2]+g[2]+b[2]
    w["bg"]=rgb
    s["text"]="颜色为"+rgb
from tkinter import *
w=Tk()
w.geometry("300x200+0+0")
v1=IntVar();v2=IntVar();v3=IntVar();
sr=Scale(w,from_=0,to=15,resolution=1,orient=HORIZONTAL,variable=v1,command=f)
sg=Scale(w,from_=0,to=15,resolution=1,orient=HORIZONTAL,variable=v2,command=f)
sb=Scale(w,from_=0,to=15,resolution=1,orient=HORIZONTAL,variable=v3,command=f)
sr.set(0);sg.set(0);sb.set(0);
sr.pack();sg.pack();sb.pack()
s=Label(w,text="颜色为#000");s.pack()
w.mainloop()
```



例：通过拖动刻度条来改变窗口背景颜色

```
def change(value):
    r=sr.get()
    g=sg.get()
    b=sb.get()
    s=hex(0xFF<<24|r<<16|g<<8|b)
    rgb="#"+s[4:]
    t["text"]="颜色值rgb="+rgb
    w["bg"]=rgb
from tkinter import *
w=Tk()
w.geometry('350x200')
w.title("通过刻度条来改变背景颜色")
sr=Scale(w,from_=0,to=255,resolution=1,orient=HORIZONTAL,command=change)
sr.set(0)  #设置滑块起始位置
sr.pack()
sg=Scale(w,from_=0,to=255,resolution=1,orient=HORIZONTAL,command=change)
sg.set(0)  #设置滑块起始位置
sg.pack()
sb=Scale(w,from_=0,to=255,resolution=1,orient=HORIZONTAL,command=change)
sb.set(0)  #设置滑块起始位置
sb.pack()
t=Label(w,text="颜色值rgb=#000000")
t.pack()
w.mainloop()
```






```
def change(value):
    r=sr.get()
    g=sg.get()
    b=sb.get()
    s=hex(0xFF<<24|r<<16|g<<8|b)
    rgb="#"+s[4:]
    t["text"]="颜色值rgb="+rgb
    w["bg"]=rgb
from tkinter import *
w=Tk()
#w.geometry('350x200')
w.title("通过刻度条来改变背景颜色")
sr=Scale(w,from_=0,to=255,resolution=1,orient=VERTICAL,command=change)
sr.set(0)  #设置滑块起始位置
sr.pack(side=LEFT)
sg=Scale(w,from_=0,to=255,resolution=1,orient=VERTICAL,command=change)
sg.set(0)  #设置滑块起始位置
sg.pack(side=LEFT)
sb=Scale(w,from_=0,to=255,resolution=1,orient=VERTICAL,command=change)
sb.set(0)  #设置滑块起始位置
sb.pack(side=LEFT)
t=Label(w,text="颜色值rgb=#000000")
t.pack(side=TOP)
w.mainloop()
```

13.2.4 文本框与框架控件

- 文本框用于输入和编辑文本。
- 输入过程中随时可以进行编辑，如光标定位、修改、插入等。
- Python的文本框包括单行文本框（entry）和多行文本框（text）。
- 文本编辑区一般可以设置行和列的大小，并且可以通过滚动条来显示、编辑更多的文本。

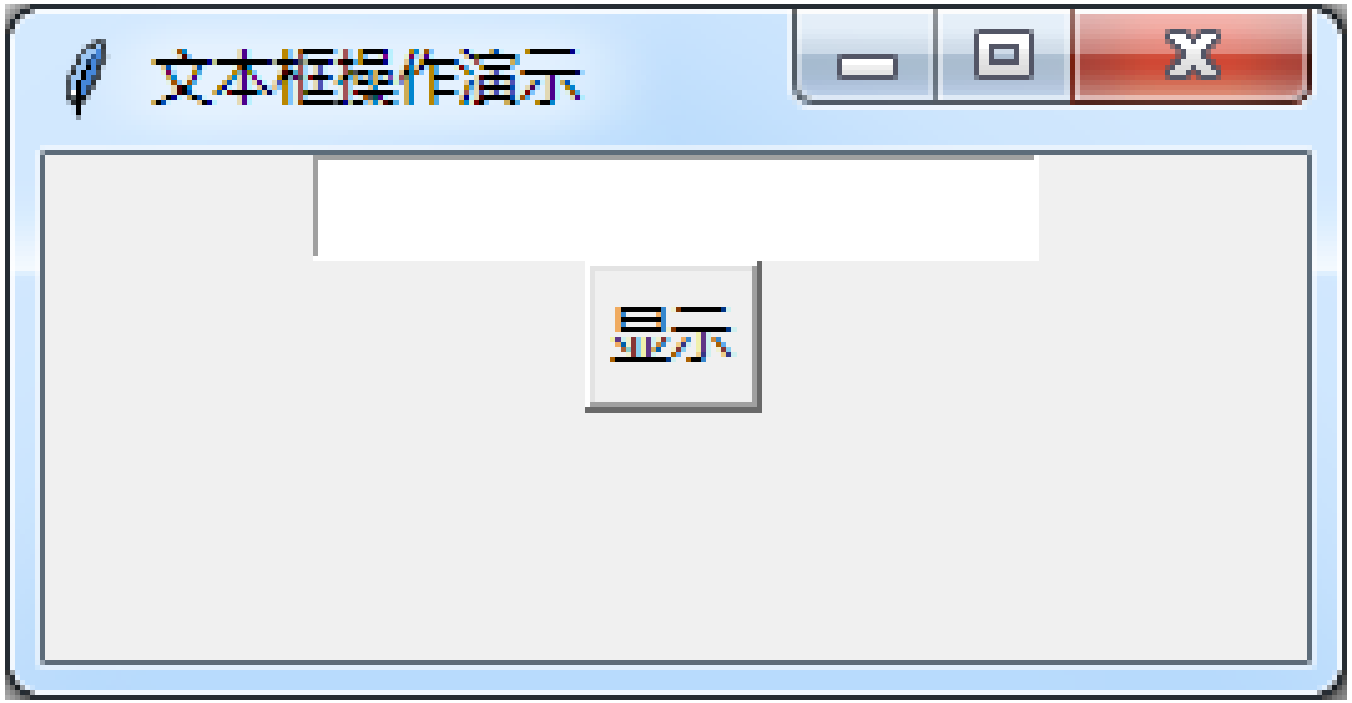
- 框架（frame）就是一种容器，用于将一组相关的基本控件组合成为一个复合控件。
- 利用框架对窗口进行模块化分隔，即可建立复杂的图形用户界面。
- 每个框架都是一个独立的区域，可以独立地对其包含的控件进行布局。

- 1. 单行文本框
- tkinter模块提供的Entry类可以实现单行文本的输入和编辑。
- 例：创建并布置一个单行文本框控件:
- >>>Entry(w).pack()
- 语句运行后在窗口中出现一行空白区域，单击此区域后会出现一个闪烁的光标，这时就可以在其中输入文本了。

- 当用户输入了数据之后，应用程序显然需要某种手段来获取用户的输入，以便对数据进行处理。
- 为此，可以通过**Entry**对象的**textvariable**属性将文本框与一个**StringVar**类型的控制变量相关联。

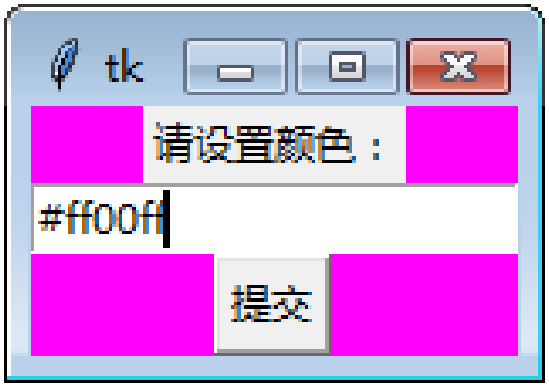
- 例：
- **>>>v=StringVar()**
- **>>>ety=Entry(w,textvariable=v)**
- **>>>ety.pack()**
- **>>>print(v.get())** #利用**v.get()**来获取文本框中的文本内容
- **>>>v.set('uestc')** #通过**v.set()**来设置文本框的内容

- 例：在文本框中输入文本后，单击“显示”按钮，把在文本框中输入的文本显示在标签中。



```
def entf():
    Label(w,text=v.get()).pack()
from tkinter import *
w=Tk()
w.geometry("250x100")
w.title("文本框操作演示")
v=StringVar()
Entry(w,textvariable=v).pack()
Button(w,text='显示',command=entf).pack()
w.mainloop()
```

```
def f():
    w["bg"]=v.get()
from tkinter import *
w=Tk()
v=StringVar()
s=Label(w,text="请设置颜色：")
t=Entry(w,width=20,textvariable=v)
b=Button(w,text="提交",command=f)
s.pack()
t.pack()
b.pack()
w.mainloop()
```



- 很多应用程序利用文本框作为用户登录系统时输入用户名和密码的界面元素，其中密码文本框一般不回显用户的输入，而是用“*”代替。
- 这在tkinter模块中只需将文本框对象show属性设置为“*”即可。
- 例：
- ety.config(show='*')

```
    def f1():
        global n
        n=n+1
        if n>3:
            showinfo(title="提示",message='你已经连续错了3次！')
            v1.set("")
            v2.set("")
            b1.config(state=DISABLED)
            b2.config(state=DISABLED)
            t1.config(state=DISABLED)
            t2.config(state=DISABLED)
            w.quit()
        else:
            x=v1.get()
            y=v2.get()
            if x=="aaa" and y=="123":
                showinfo(title="欢迎你！",message='欢迎进入本系统！')
                n=0
                w.quit()
            else:
                showinfo(title="提示",message='账号或密码错误！')
                f2()
    def f2():
        v1.set("")
        v2.set("")
    from tkinter import *
    from tkinter.messagebox import *
    w=Tk()
    w.title("系统登录")
    n=0
    v1=StringVar()
    v2=StringVar()
    m1=Frame(w)
    m2=Frame(w)
    m3=Frame(w)
    m1.grid(row=0,column=0)
    m2.grid(row=0,column=1)
    m3.grid(row=1,column=0,columnspan=2)
    Label(m1,text="账号").grid(row=0,column=0)
    Label(m1,text="密码").grid(row=1,column=0)
    t1=Entry(m2,width=20,textvariable=v1)
    t2=Entry(m2,width=20,textvariable=v2,show='*')
    t1.grid(row=0,column=0)
    t2.grid(row=1,column=0)
    b1=Button(m3,text="提交",command=f1)
    b2=Button(m3,text="重填",command=f2)
    b1.grid(row=0,column=0,padx=5,pady=5)
    b2.grid(row=0,column=1,padx=5,pady=5)
    w.mainloop()
```

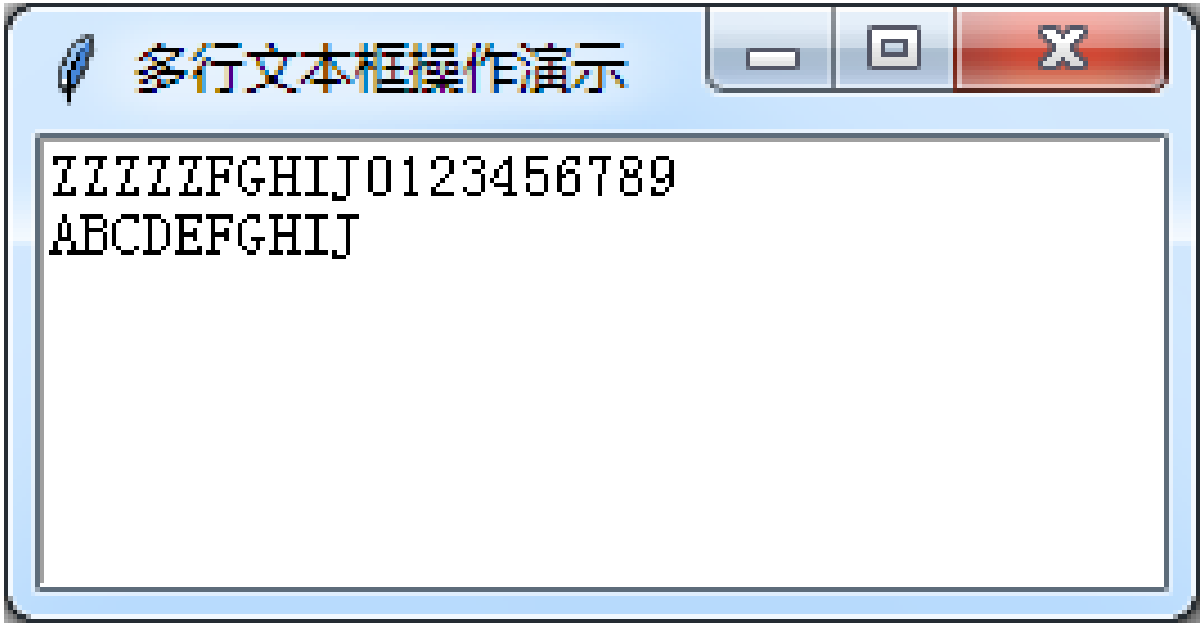
- **2.多行文本框**
- 除Entry类外，tkinter模块还提供一个支持多行文本输入与编辑的文本框控件类Text。
- Text控件的用途非常多，用法也比Entry复杂，但两者的基本用法是类似的。

- 例：
- **>>>txt=Text(w)**
- **>>>txt.pack()**
- 其运行结果是在窗口中出现了一个多行的空白区域，在此区域可输入、编辑多行文本。

- 多行文本框提供文本的获取、删除、插入和替换等功能:
- **txt.get(index1,index2):**获取指定范围的文本。
- **txt.delete(index1,index2):**删除指定范围的文本。
- **txt.insert(index,text):**在**index**位置插入文本。
- **txt.replace(index1,index2,text):**替换指定范围的文本。
- 在方法调用语句中，**index**参数代表文本的位置，其格式为“行标号.列标号”，行标号从**1**开始，列标号从**0**开始，**1.0**表示第一行第一列。

- index也可以用位置标签:
- INSERT代表光标的插入点；
- CURRENT代表鼠标的当前位置；
- END代表文本框的最后一个字符之后的位置；
- SEL_FIRST代表选中文本域的第一个字符，如果没有选中区域则会引发异常；
- SEL_LAST代表选中文本域的最后一个字符，如果没有选中区域则会引发异常。

- 例：多行文本框操作示例。



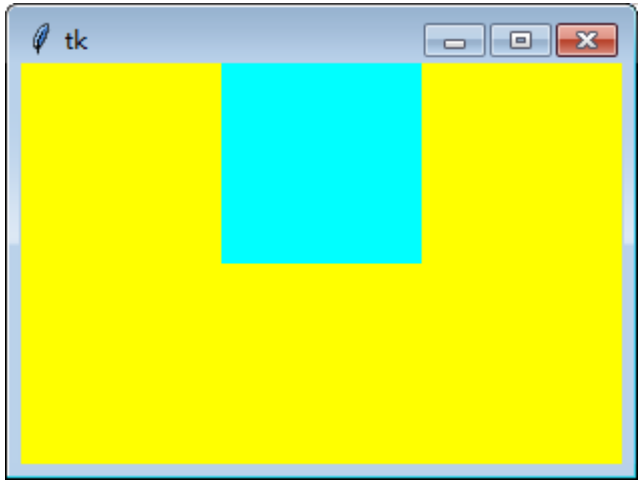
- from tkinter import *
- w=Tk()
- w.geometry("250x100")
- w.title("多行文本框操作演示")
- t=Text(w)
- t.insert(1.0,'0123456789')
- t.insert(END,'\n')
- t.insert(1.0,'ABCDEFGHIJ')
- t.insert(2.0,'ABCDEFGHIJ')
- t.replace(1.0,1.5,'ZZZZZ')
- t.pack()
- w.mainloop()

- 多行文本框还支持其他高级编辑功能，包括剪贴板的操作、操作撤销与重做等。

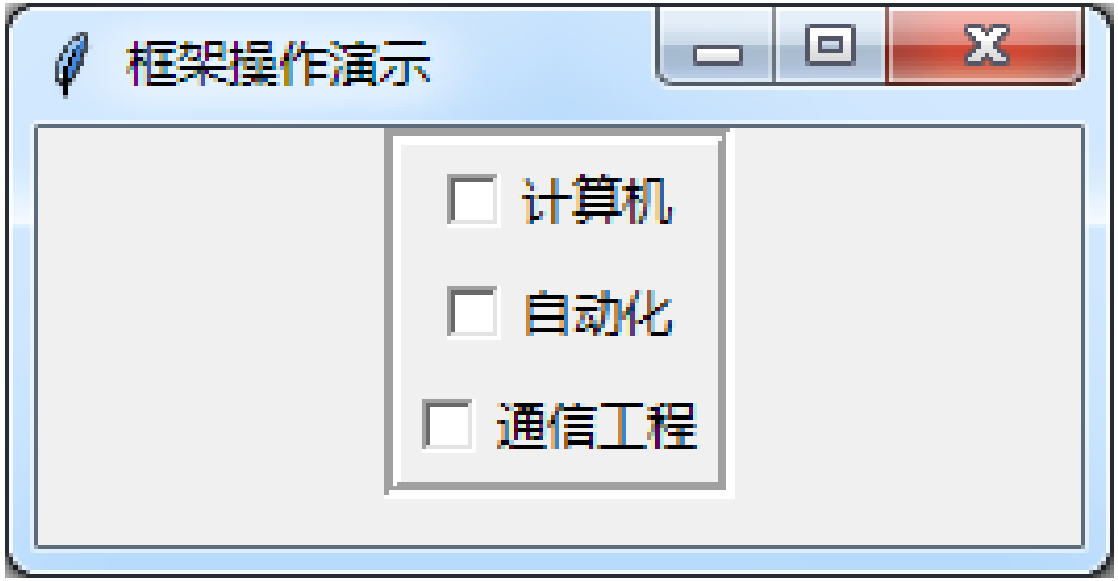
- 3. 框架
- tkinter模块提供了Frame类来创建框架控件，框架的宽度和高度分别用width和height属性来设置，框架的边框粗细用bd（或border、borderwidth）属性来设置（默认值为0，即没有边框）。
- 边框的3D风格用relief属性来设置（与按钮相同）。

- 例：
- >>>f=Frame(w,width=300,height=400,bd=4,relief=GROOVE)
- >>>f.pack()

```
from tkinter import *
w=Tk()
w.geometry("300x200+0+0")
f=Frame(w,width=100,height=100,bd=0,relief=GROOVE)
f.pack()
w["bg"]="yellow"
f["bg"]="cyan"
w.mainloop()
```

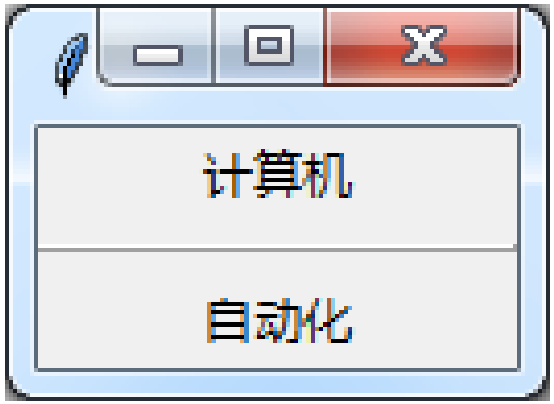


- 例：框架操作示例。



- from tkinter import *
- w=Tk()
- w.geometry("250x100")
- w.title("框架操作演示")
- f=Frame(w,bd=4,relief=GROOVE)
- f.pack()
- Checkbutton(f,text="计算机").pack()
- Checkbutton(f,text="自动化").pack()
- Checkbutton(f,text="通信工程").pack()
- w.mainloop()

- 框架除了作为容器来组合多个控件外，还用于图形界面的空间分隔或填充。
- 例：框架只起着分隔两个标签控件的作用，该框架在主窗口中以x方向填满（fill=X）、y方向上下各填充5个像素空间（pady=5）的方式进行pack布局。



- from tkinter import *
- w=Tk()
- Label(w,text="计算机").pack()
- Frame(height=2,bd=1,relief=RIDGE).pack(fill=X,pady=5)
- Label(text="自动化").pack()
- w.mainloop()

13.2.5 菜单与顶层窗口控件

- 菜单（menu）控件是一个由许多菜单项组成的列表，每个菜单项表示一条命令或一个选项。
- 用户通过鼠标或键盘选择菜单项，以执行命令或选中选项。
- 菜单项通常以相邻的方式放置在一起，形成窗口的菜单栏，并且一般置于窗口顶端。

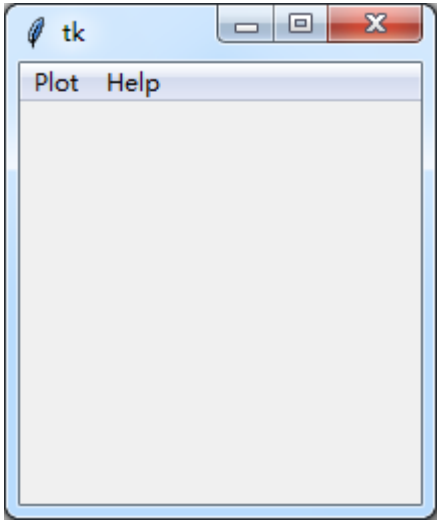
- 除菜单栏里的菜单外，还常用一种上下文菜单，这种菜单平时在界面中是不可见的，当用户在界面中单击鼠标右键时才会弹出一个与单击对象相关的菜单。
- 有时，菜单中一个菜单项的作用是展开另一个菜单，形成级联式菜单。

- tkinter模块支持多窗口应用，除创建主窗口外，还可以创建顶层窗口。
- 顶层窗口的外观与主窗口一样，可以独立地移动和改变大小，并且不需要像其他控件那样必须在主窗口中进行布局后才显示。
- 一个应用程序只能有一个主窗口，但可以创建任意多个顶层窗口。

- 1. 菜单
- tkinter模块提供Menu类用于创建菜单控件，具体用法是先创建一个菜单控件对象，并与某个窗口（主窗口或者顶层窗口）进行关联，然后再为该菜单添加菜单项。
- 与主窗口关联的菜单实际上构成了主窗口的菜单栏。

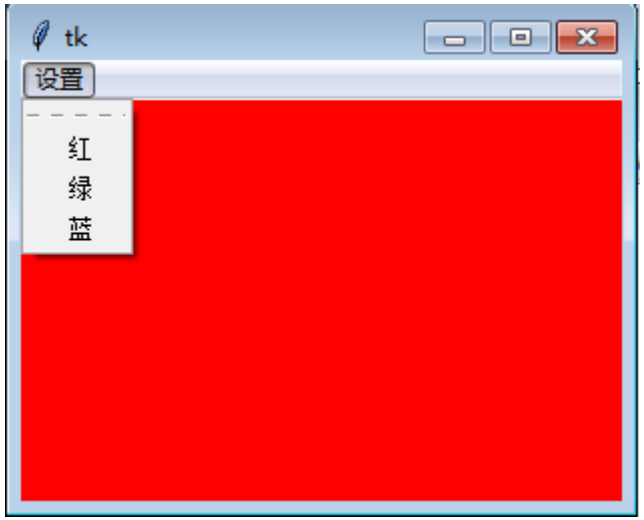
- 菜单项可以是简单命令、级联式菜单、复选框或一组单选按钮，分别用**add_command()**、**add_cascade()**、**add_checkbutton()**和**add_radiobutton()**方法来添加。
- 为了使菜单结构清晰，还可以用**add_separator()**方法在菜单中添加分隔线。

- 例：创建一个菜单。
- from tkinter import *
- w=Tk()
- m=Menu(w)
- w.config(menu=m) #与w['menu']=m等价
- m.add_command(label="Plot")
- m.add_command(label="Help")

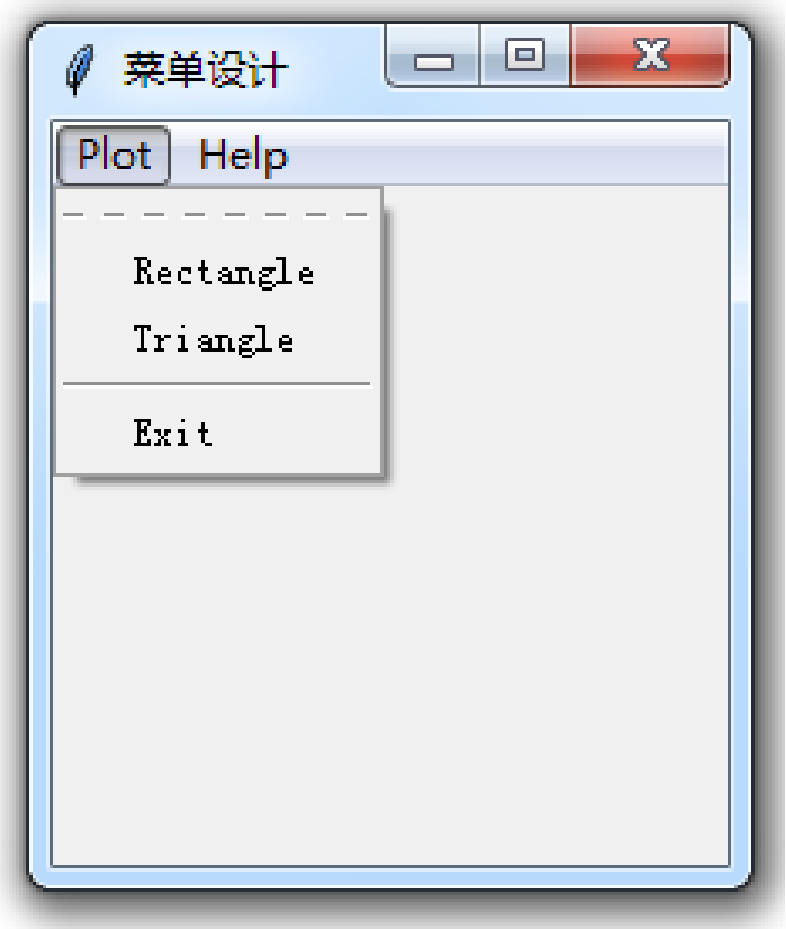


- 菜单控件与前面的控件都不同，不需要调用布局管理器来使之可见，tkinter模块会自动布局并显示菜单。

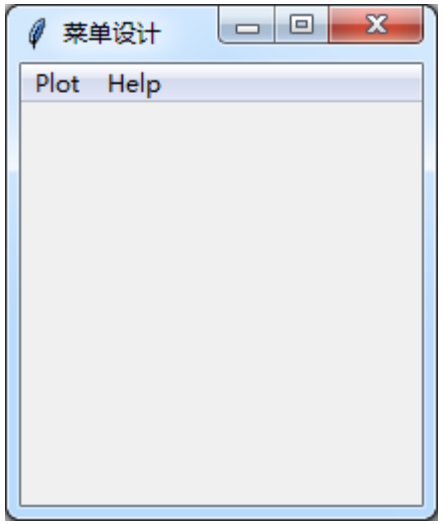
```
def rf():w["bg"]="red"
def gf():w["bg"]="green"
def bf():w["bg"]="blue"
from tkinter import *
w=Tk()
w.geometry("300x200+0+0")
m=Menu(w)
w.config(menu=m)
sm=Menu(m)
m.add_cascade(label="设置",menu=sm)
sm.add_command(label="红",command=rf)
sm.add_command(label="绿",command=gf)
sm.add_command(label="蓝",command=bf)
w.mainloop()
```



- 例：设计绘图菜单，选择菜单项能绘制相应图形。

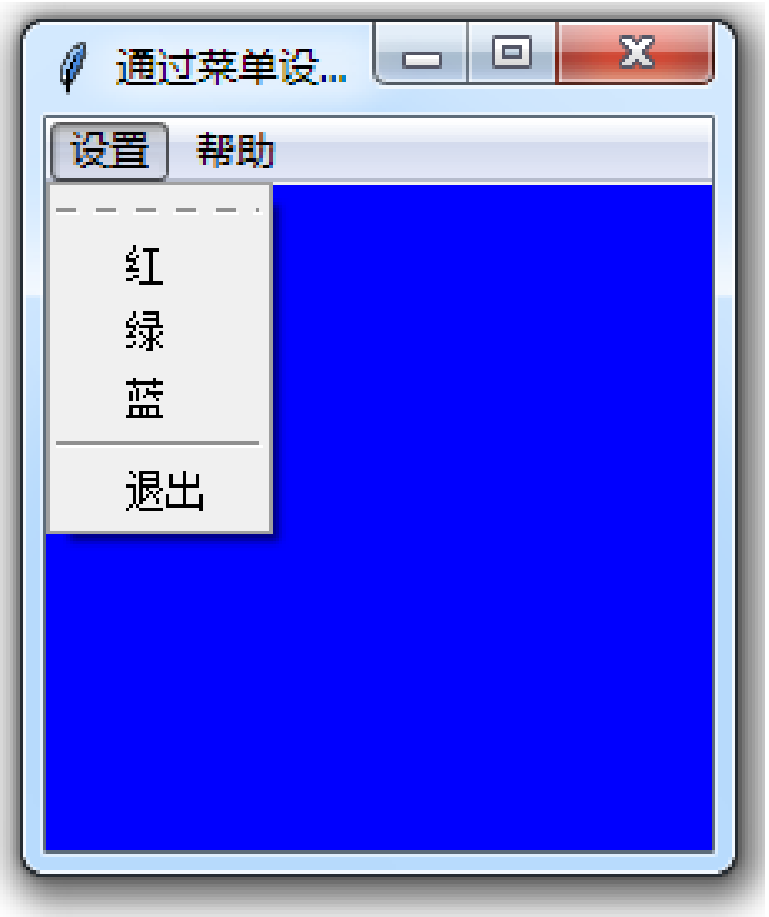


```
from tkinter import *
def callback1():
    c=Canvas(w,width=300,height=200,bg='white')
    c.pack()
    c.create_rectangle(50,50,200,100)
def callback2():
    c=Canvas(w,width=300,height=200,bg='white')
    c.pack()
    c.create_polygon(35,10,10,60,60,60)
def callback3():
    w.quit()
def callback4():
    print("Python is very useful.")
w=Tk()
w.title("菜单设计")
m=Menu(w)
w.config(menu=m)
plotmenu=Menu(m)
m.add_cascade(label="Plot",menu=plotmenu)
plotmenu.add_command(label="Rectangle",command=callback1)
plotmenu.add_command(label="Triangle",command=callback2)
plotmenu.add_separator()
plotmenu.add_command(label="Exit",command=callback3)
helpmenu=Menu(m)
m.add_cascade(label="Help",menu=helpmenu)
helpmenu.add_command(label="About...",command=callback4)
w.mainloop()
```

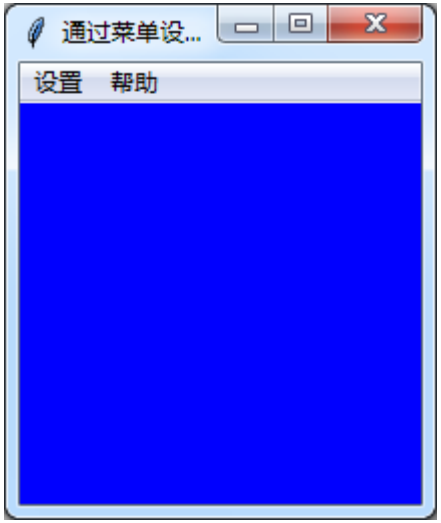


- 各个菜单在界面中的位置由tkinter自动布局，不需要调用布局管理器。
- 各个命令项关联到不同的函数实现不同的功能。
- 菜单项下面有一根虚线，单击虚线可以将菜单项和主窗口分离。
- 在创建级联菜单时，设置tearoff属性为0，则禁止分离，语句为:
- **plotmenu=Menu(m,tearoff=0)**

- 例：通过菜单来设置背景颜色。



```
from tkinter import *
def rf(): w["bg"]="red"
def gf(): w["bg"]="green"
def bf(): w["bg"]="blue"
def xf(): w.quit()
def hf(): print("好好学习，天天向上！")
w=Tk()
w.title("通过菜单设置背景颜色")
m=Menu(w)
w.config(menu=m)
sm=Menu(m)
m.add_cascade(label="设置",menu=sm)
sm.add_command(label="红",command=rf)
sm.add_command(label="绿",command=gf)
sm.add_command(label="蓝",command=bf)
sm.add_separator()
sm.add_command(label="退出",command=xf)
hm=Menu(m)
m.add_cascade(label="帮助",menu=hm)
hm.add_command(label="好好学习，天天向上！",command=hf)
w.mainloop()
```



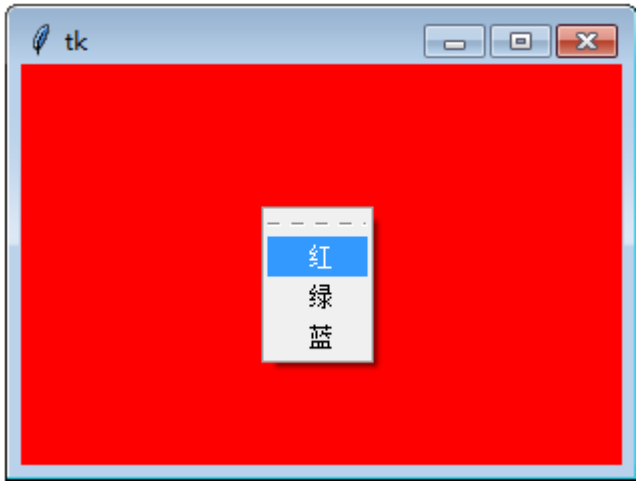
```
from tkinter import *
def f1():global x;x=c.create_rectangle(10,10,50,100)
def f2():global y;y=c.create_polygon(60,100,80,10,100,100)
def f3():global z;z=c.create_oval(110,10,150,100)
def f4():c.delete(x);
def f5():c.delete(y);
def f6():c.delete(z);
def f7():c.delete(ALL);
def f8():w.quit()
def f9():print("Python is very useful.")
w=Tk()
w.title("绘图")
c=Canvas(w,width=300,height=200)
c.pack()
m=Menu(w)
w.config(menu=m)
pm=Menu(m)
m.add_cascade(label="Plot",menu=pm)
pm.add_command(label="Rectangle",command=f1)
pm.add_command(label="Triangle",command=f2)
pm.add_command(label="Oval",command=f3)
pm.add_separator()
dm=Menu(pm)
pm.add_cascade(label="Clear",menu=dm)
dm.add_command(label="Clear Rectangle",command=f4)
dm.add_command(label="Clear Triangle",command=f5)
dm.add_command(label="Clear Oval",command=f6)
dm.add_command(label="Clear All",command=f7)
pm.add_separator()
pm.add_command(label="Exit",command=f8)
hm=Menu(m)
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="About...",command=f9)
w.mainloop()
```

- **2. 上下文菜单**
- 上下文菜单也叫快捷菜单，它是右击某个对象而弹出的菜单。
- 创建上下文菜单先要创建菜单，然后将绑定对象的鼠标右击事件，并在事件处理函数中弹出菜单。

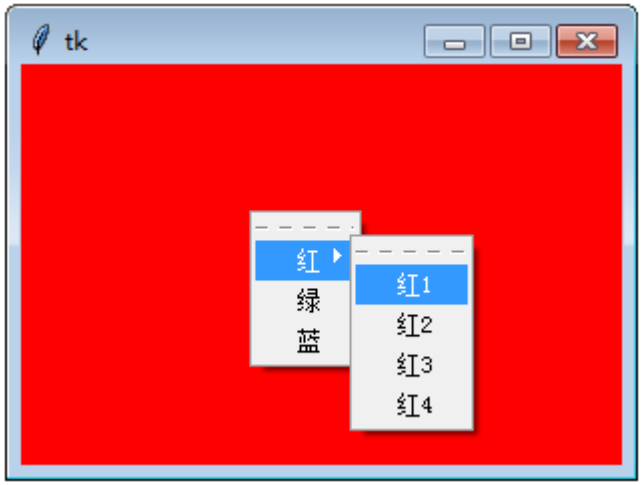
- 例：将上例的菜单改为上下文菜单，方法为:
- 不要将主窗口w的menu属性设置为菜单对象m，即去掉“w.config(menu=m)”或“w['menu']=m”语句，而用以下语句代替。

- def popup(event): #事件处理函数
- m.post(event.x_root,event.y_root) #在鼠标右键位置显示菜单
- w.bind('<Button-3>',popup)
- 绑定鼠标右键事件，右击时调用popup()函数。
- 这里与菜单绑定的是主窗口w，也可以设置为其他的控件，在绑定的控件上右击就可以弹出菜单。

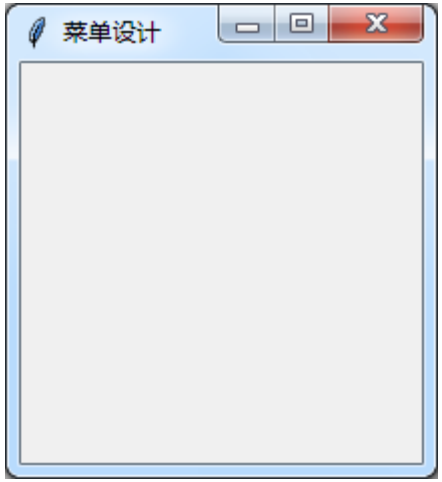
```
def popup(event):m.post(event.x_root,event.y_root)
def rf():w["bg"]="red"
def gf():w["bg"]="green"
def bf():w["bg"]="blue"
from tkinter import *
w=Tk()
w.geometry("300x200+0+0")
m=Menu(w)
w.bind("<Button-3>",popup)
m.add_command(label="红",command=rf)
m.add_command(label="绿",command=gf)
m.add_command(label="蓝",command=bf)
w.mainloop()
```

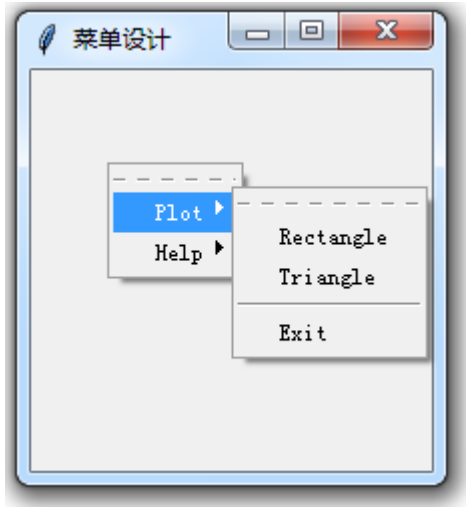


```
def popup(event):m1.post(event.x_root,event.y_root)
def r1():w["bg"]="red1"
def r2():w["bg"]="red2"
def r3():w["bg"]="red3"
def r4():w["bg"]="red4"
def gf():w["bg"]="green"
def bf():w["bg"]="blue"
from tkinter import *
w=Tk()
w.geometry("300x200+0+0")
m1=Menu(w)
w.bind("<Button-3>",popup)
m2=Menu(m1)
m1.add_cascade(label="红",menu=m2)
m1.add_command(label="绿",command=gf)
m1.add_command(label="蓝",command=bf)
m2.add_command(label="红1",command=r1)
m2.add_command(label="红2",command=r2)
m2.add_command(label="红3",command=r3)
m2.add_command(label="红4",command=r4)
w.mainloop()
```

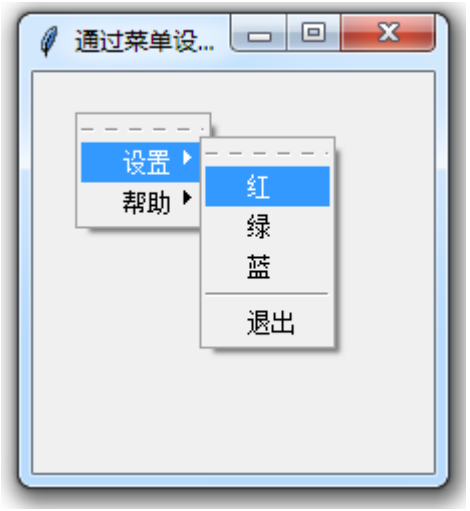


```
from tkinter import *
def callback1():
    c=Canvas(w,width=300,height=200,bg='white')
    c.pack()
    c.create_rectangle(50,50,200,100)
def callback2():
    c=Canvas(w,width=300,height=200,bg='white')
    c.pack()
    c.create_polygon(35,10,10,60,60,60)
def callback3():
    w.quit()
def callback4():
    print("Python is very useful.")
def popup(event): #事件处理函数
    m.post(event.x_root,event.y_root) #在鼠标右键位置显示菜单
w=Tk()
w.title("菜单设计")
m=Menu(w)
w.bind('<Button-3>',popup)
plotmenu=Menu(m)
m.add_cascade(label="Plot",menu=plotmenu)
plotmenu.add_command(label="Rectangle",command=callback1)
plotmenu.add_command(label="Triangle",command=callback2)
plotmenu.add_separator()
plotmenu.add_command(label="Exit",command=callback3)
helpmenu=Menu(m)
m.add_cascade(label="Help",menu=helpmenu)
helpmenu.add_command(label="About...",command=callback4)
w.mainloop()
```





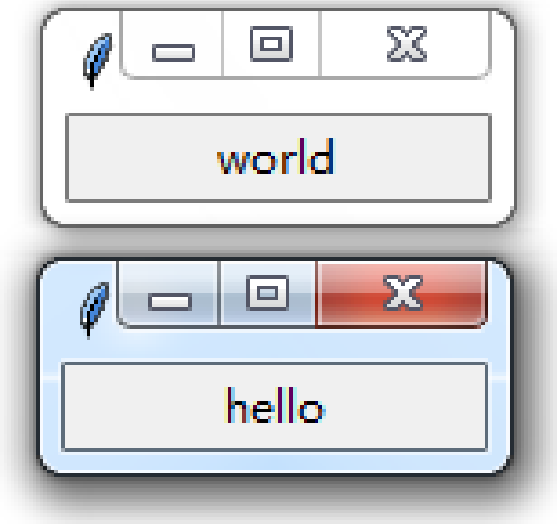
```
from tkinter import *
def rf(): w["bg"]="red"
def gf(): w["bg"]="green"
def bf(): w["bg"]="blue"
def xf(): w.quit()
def pf(): print("好好学习，天天向上！")
def popup(event): #在鼠标右键位置显示菜单
    m.post(event.x_root,event.y_root)
w=Tk()
w.title("通过菜单设置背景颜色")
m=Menu(w)
w.bind('<Button-3>',popup)
pm=Menu(m)
m.add_cascade(label="设置",menu=pm)
pm.add_command(label="红",command=rf)
pm.add_command(label="绿",command=gf)
pm.add_command(label="蓝",command=bf)
pm.add_separator()
pm.add_command(label="退出",command=xf)
hm=Menu(m)
m.add_cascade(label="帮助",menu=hm)
hm.add_command(label="好好学习，天天向上！",command=pf)
w.mainloop()
```



```
from tkinter import *
def f1():global x;x=c.create_rectangle(10,10,50,100)
def f2():global y;y=c.create_polygon(60,100,80,10,100,100)
def f3():global z;z=c.create_oval(110,10,150,100)
def f4():c.delete(x);
def f5():c.delete(y);
def f6():c.delete(z);
def f7():c.delete(ALL);
def f8():w.quit()
def f9():print("Python is very useful.")
def popup(event):
    m.post(event.x_root,event.y_root)
w=Tk()
w.title("绘图")
c=Canvas(w,width=300,height=200)
c.pack()
m=Menu(w)
w.bind('<Button-3>',popup)
pm=Menu(m)
m.add_cascade(label="Plot",menu=pm)
pm.add_command(label="Rectangle",command=f1)
pm.add_command(label="Triangle",command=f2)
pm.add_command(label="Oval",command=f3)
pm.add_separator()
dm=Menu(pm)
pm.add_cascade(label="Clear",menu=dm)
dm.add_command(label="Clear Rectangle",command=f4)
dm.add_command(label="Clear Triangle",command=f5)
dm.add_command(label="Clear Oval",command=f6)
dm.add_command(label="Clear All",command=f7)
pm.add_separator()
pm.add_command(label="Exit",command=f8)
hm=Menu(m)
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="About...",command=f9)
w.mainloop()
```

- **3. 顶层窗口**
- **tkinter模块提供Toplevel类用于创建顶层窗口控件。**

- 例：
- **from tkinter import ***
- **w=Tk()**
- **Label(w,text="hello").pack()**
- **top=Toplevel()**
- **Label(top,text="world").pack()**



- 该程序先创建主窗口，在主窗口中创建一个标签，然后创建了一个顶层窗口，又在顶层窗口中创建了一个标签。

- 虽然创建顶层窗口对象top时没有指定以主窗口w作为父控件，但top确实是w的子控件，因此关闭top并不会结束程序，因为主窗口仍在工作。
- 但若关闭主窗口，则包含top在内的整个界面都会关闭。
- 所以顶层窗口虽然具有相对的独立性，但它不能脱离主窗口而存在。
- 即使在没有主窗口的情况下直接创建顶层窗口，系统也会自动先创建主窗口。

- 与主窗口类似，可以调用**Toplevel**类的**title()**和**geometry()**方法来设置它的标题和大小。
- 例：
- **top.title('顶层窗口')**
- **top.geometry('400x300')**
- 也可以为顶层窗口建立菜单，操作方法和主窗口类似。

13.2.6 ttk子模块控件

- tkinter模块包括ttk子模块。
- ttk包括了tkinter中没有的Combobox、Notebook、Progressbar、Separator、Sizegrip and Treeview等控件，使得tkinter更实用。
- ttk子模块的Style对象可以统一设置控件的样式。

- 例：设置每一个按钮的内边距（**padding**）和背景颜色（**background**）。

- from tkinter import ttk
- import tkinter
- w=tkinter.Tk()
- style=ttk.Style()
- style.configure("TButton",padding=5,background="
  yellow")
- ttk.Button(text="Sample").pack()
- ttk.Button(text="显示").pack()
- w.mainloop()

13.3 对象的布局方式

- 布局指的是子控件在父控件中的位置安排。
- tkinter模块提供了三种布局管理器，即pack、grid和place，其任务是根据设计要求来安排控件的位置。

13.3.1 pack布局管理器

- pack布局管理器将所有控件组织为一行或一列，默认的布局方式是根据控件创建的顺序将控件自上而下地添加到父控件中。
- 可以使用side、fill、expand、ipadx/ipady、padx/pady等属性对控件的布局进行控制。

- side属性改变控件的排列位置，LEFT居左放置，RIGHT居右放置，TOP居上放置，BOTTOM居下放置。
- fill属性设置填充空间，取值为X则在x方向填充，取值为Y则在y方向填充，取值为BOTH则在x、y两个方向上填充，取值为NONE则不填充。

- expand属性指定如何使用额外的“空白”空间，取值为1则随着父控件的大小变化而变化，取值为0则子控件大小不能扩展。
- ipadx/ipady设置控件内部在x/y方向的间隙。
- padx/pady设置控件外部在x/y方向的间隙。

- 例： pack布局管理器应用示例。

- from tkinter import *
- w=Tk()
- w.geometry('250x100') #改变w的大小为250x100
- Lbl1=Label(w,text='北京',bg='yellow3')
- Lbl1.pack(expand=1,side=LEFT,ipadx=20)
- Lbl2=Label(w,text='天津',bg='red')
- Lbl2.pack(fill=BOTH,expand=1,side=LEFT,padx=10)
- Lbl3=Label(w,text='上海',bg='green')
- Lbl3.pack(fill=X,expand=0,side=RIGHT,padx=10)
- w.mainloop()

13.3.2 grid布局管理器

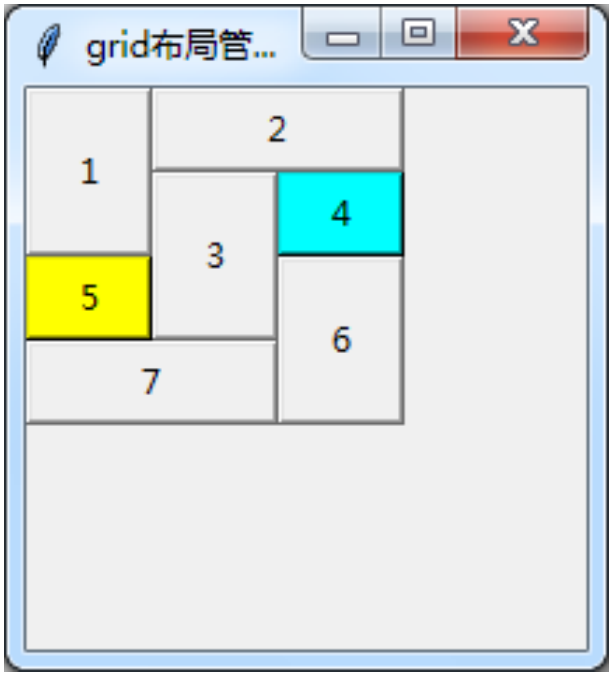
- grid布局管理器将窗口或框架视为一个由行和列构成的二维表格，并将控件放入行列交叉处的单元格中。
- grid布局管理用grid()方法的选项row和column指定行、列编号。
- 行、列都是从0开始编号，row的默认值为当前的空行，column的默认值总为0。

- 可以在布置控件时指定不连续的行号或列号，这相对于预留了一些行列，但这些预留的行列是不可见的，因为行列上没有控件存在，也就没有宽度和高度。

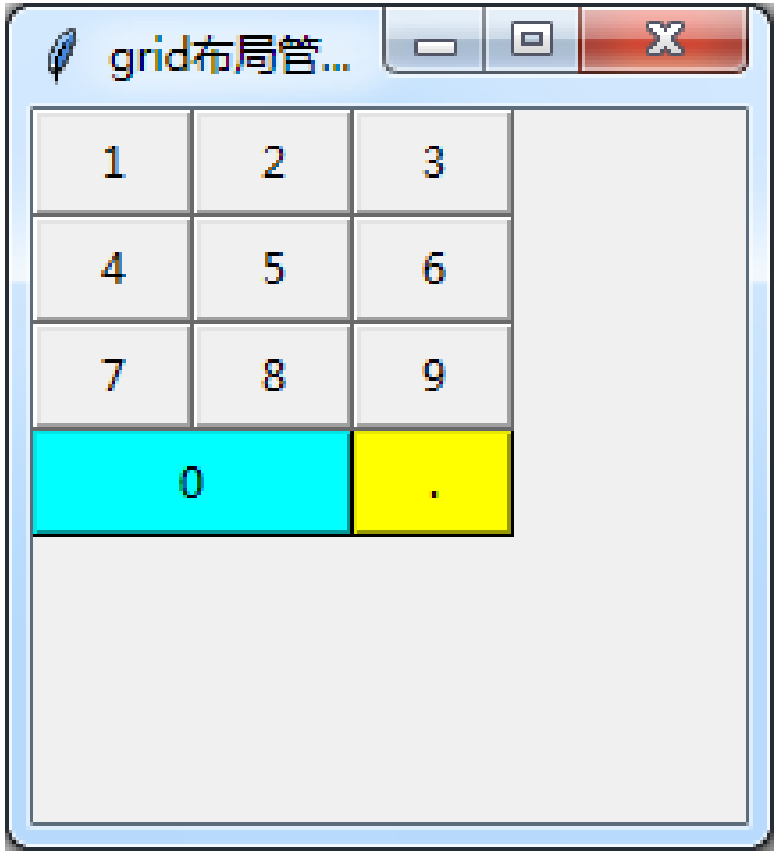
- grid()方法的sticky选项用来改变对齐方式。
- tkinter模块中常利用方位概念来指定对齐方式，具体方位值包括N、S、E、W、CENTER，分别代表上、下、左、右、中心点，还可以取NE、SE、NW、SW，分别代表右上角、右下角、左上角、左下角。
- 将sticky选项设置为某个方位，就表示将控件沿单元格的某条边或某个角对齐。

- 如果控件比单元格小，未能填满单元格，则可以指定如何处理多余空间，比如在水平方向或垂直
- 方向上拉伸控件以填满单元格。
- 可以利用方位值的组合来延伸控件，例如若将sticky设置为E+W，则控件将在水平方向上延伸，占满单元格的宽度；若设置为E+W+N+S（或NW+SE），则控件将在水平和垂直两个方向上延伸，占满整个单元格。

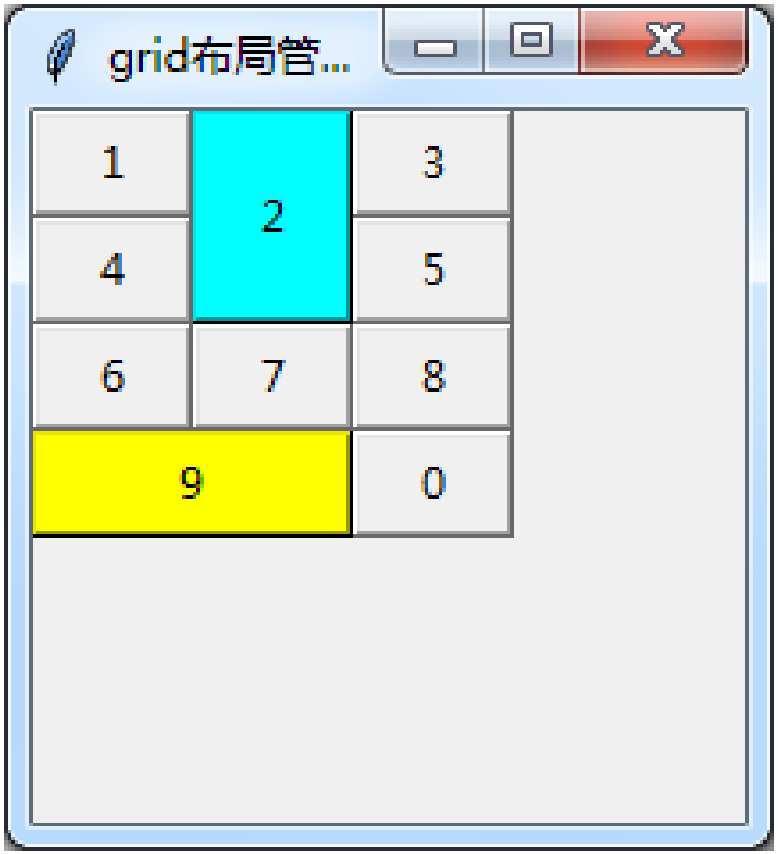
- 如果想让一个控件占据多个单元格，可以使用grid()方法的rowspan和columnspan选项来指定在行和列方向上的跨度。
- 例：grid布局管理器应用示例。



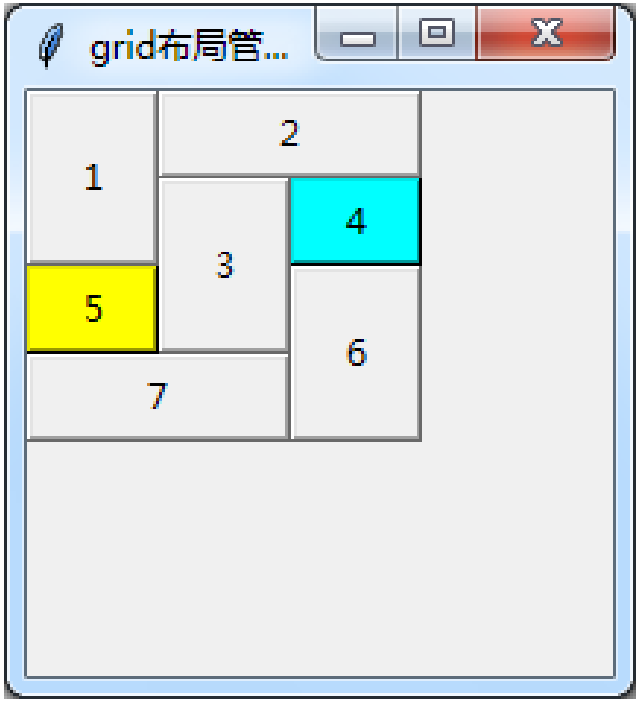
```
from tkinter import *
w=Tk()
w.geometry('200x200+150+150')
w.title("grid布局管理器")
b1=Button(w,text='1',width=5)
b2=Button(w,text='2',width=5)
b3=Button(w,text='3',width=5)
b4=Button(w,text='4',width=5)
b5=Button(w,text='5',width=5)
b6=Button(w,text='6',width=5)
b7=Button(w,text='7',width=5)
b8=Button(w,text='8',width=5)
b9=Button(w,text='9',width=5)
b0=Button(w,text='0',width=5,bg='cyan')
bp=Button(w,text='.',width=5,bg='yellow')
b1.grid(row=0,column=0)
b2.grid(row=0,column=1)
b3.grid(row=0,column=2)
b4.grid(row=1,column=0)
b5.grid(row=1,column=1)
b6.grid(row=1,column=2)
b7.grid(row=2,column=0)
b8.grid(row=2,column=1)
b9.grid(row=2,column=2)
b0.grid(row=3,column=0,columnspan=2,sticky=E+W)
bp.grid(row=3,column=2,sticky=E+W)
w.mainloop()
```



```
from tkinter import *
w=Tk()
w.geometry('200x200+150+150')
w.title("grid布局管理器")
b1=Button(w,text='1',width=5)
b2=Button(w,text='2',width=5,bg='cyan')
b3=Button(w,text='3',width=5)
b4=Button(w,text='4',width=5)
b5=Button(w,text='5',width=5)
b6=Button(w,text='6',width=5)
b7=Button(w,text='7',width=5)
b8=Button(w,text='8',width=5)
b9=Button(w,text='9',width=5,bg='yellow')
b0=Button(w,text='0',width=5)
b1.grid(row=0,column=0)
b2.grid(row=0,column=1,rowspan=2,sticky=N+S)
b3.grid(row=0,column=2)
b4.grid(row=1,column=0)
b5.grid(row=1,column=2)
b6.grid(row=2,column=0)
b7.grid(row=2,column=1)
b8.grid(row=2,column=2)
b9.grid(row=3,column=0,columnspan=2,sticky=E+W)
b0.grid(row=3,column=2)
w.mainloop()
```



- from tkinter import *
- w=Tk()
- w.geometry('200x200+150+150')
- w.title("grid布局管理器")
- b1=Button(w,text='1',width=5)
- b2=Button(w,text='2',width=5)
- b3=Button(w,text='3',width=5)
- b4=Button(w,text='4',width=5,bg='cyan')
- b5=Button(w,text='5',width=5,bg='yellow')
- b6=Button(w,text='6',width=5)
- b7=Button(w,text='7',width=5)
- b1.grid(row=0,column=0,rowspan=2,sticky=N+S)
- b2.grid(row=0,column=1,columnspan=2,sticky=E+W)
- b3.grid(row=1,column=1,rowspan=2,sticky=N+S)
- b4.grid(row=1,column=2)
- b5.grid(row=2,column=0)
- b6.grid(row=2,column=2,rowspan=2,sticky=N+S)
- b7.grid(row=3,column=0,columnspan=2,sticky=E+W)
- w.mainloop()



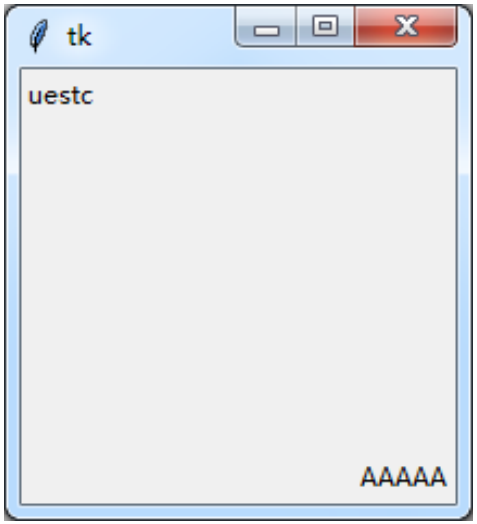
- from tkinter import *
- w=Tk()
- v1=IntVar()
- v2=IntVar()
- Label(w,text="姓名").grid(row=0,column=0,sticky=W)
- Label(w,text="住址").grid(row=1,column=0,sticky=W)
- Entry(w).grid(row=0,column=1)
- Entry(w).grid(row=1,column=1)
- f=LabelFrame(w,text='性别')
- r1=Radiobutton(f,text='男',variable=v1)
- r2=Radiobutton(f,text='女',variable=v2)
- f.grid(sticky=W)
- r1.grid(sticky=W)
- r2.grid(sticky=W)
- p=PhotoImage(file="d:\\0.gif")
- s=Label(image=p)
- s.image=p
- s.grid(row=2,column=1,sticky=W+E+N+S,padx=5,pady=5)
- w.mainloop()

13.3.3 place布局管理器

- place布局管理器直接指定控件在父控件（窗口或框架）中的位置坐标。
- 为使用这种布局，只需先创建控件，再调用控件的place()方法，该方法的选项x和y用于设定坐标。
- 父控件的坐标系以左上角为(0，0)，x轴方向向右，y轴方向向下。

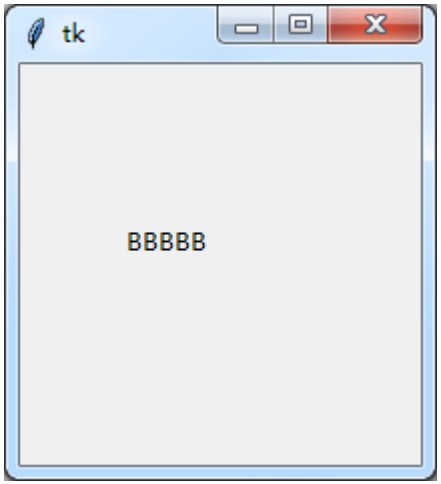
- 由于(x，y)坐标确定的是一个点，而子控件可看成一个矩形，这个矩形怎么放置在一个点上呢？
- place布局管理器通过“锚点”来处理这个问题:利用方位值指定子控件的锚点，再利用place()方法的anchor选项来将子控件的锚点定位于父控件的指定坐标处。
- 利用这种精确的定位，可以实现一个或多个控件在父控件中的各种对齐方式。

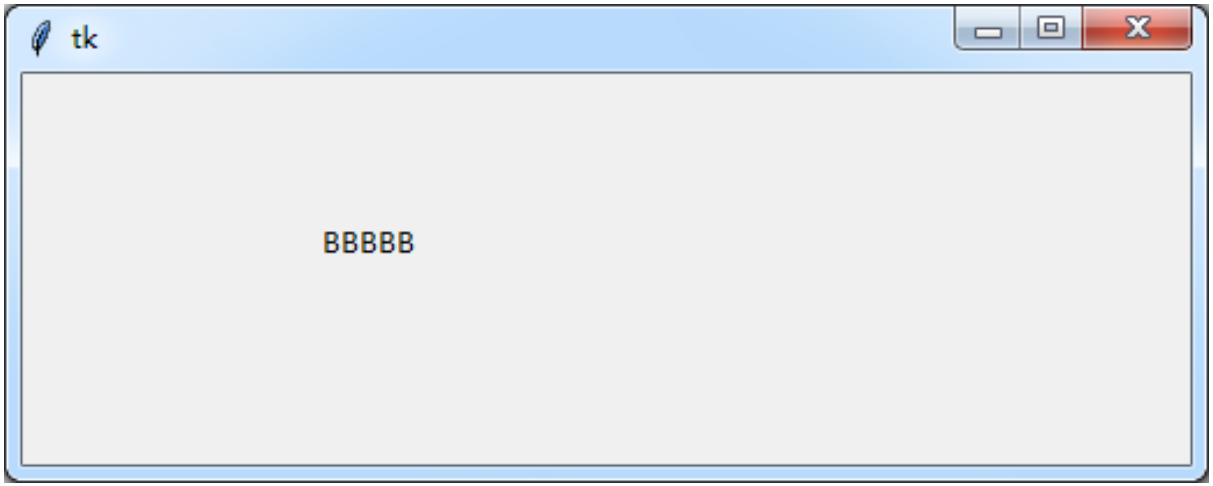
- anchor的默认值为NW，即控件的左上角。
- 例：分别将两个标签置于主窗口的(0，0)和(199，199)处，定位锚点分别是NW（默认值）和SE。
- from tkinter import *
- w=Tk()
- Label(w,text="uestc").place(x=0,y=0)
- Label(w,text="AAAAA").place(x=199,y=199,anchor=SE)



- place布局管理器既可以用绝对坐标指定位置，也可以用相对坐标指定位置。
- 相对坐标通过选项relx和rely来设置，取值范围为0~1，表示控件在父控件中的相对比例位置。
- 例：relx=0.5即表示父控件在x方向上的1/2处。
- 相对坐标的好处:当窗口改变大小时，控件位置将随之调整，不像绝对坐标固定不变。

- 例：将标签布置于水平方向1/4、垂直方向1/2，定位锚点是SW。
- from tkinter import *
- w=Tk()
- Label(w,text="BBBBB").place(relx=0.25,rely=0.5,anchor=SW)





- 除了指定控件位置外，place布局管理器还可以指定控件大小。
- 既可以通过选项width和height来定义控件的绝对尺寸，也可以通过选项relwidth和relheight来定义控件的相对尺寸，即相对于父控件两个方向上的比例值。

- place是最灵活的布局管理器，但用起来比较麻烦，通常不适合对普通窗口和对话框进行布局，其主要用途是实现复合控件的定制布局。

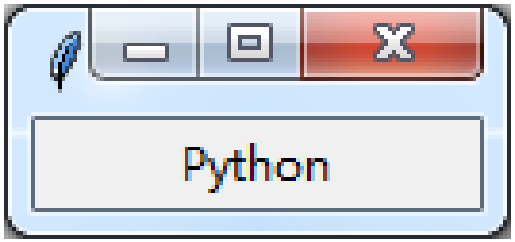
13.4 对话框

- 对话框是一个独立的顶层窗口，通常是在程序执行过程中根据需要而弹出的窗口，用于从用户获取输入或者向用户显示消息。
- 对话框包括自定义对话框和标准对话框。

13.4.1 自定义对话框

- 设计自定义的对话框窗口与创建其他窗口相似，主要步骤都是先创建顶层窗口对象，然后添加所需的按钮和其他控件。

- 例：用Msg()函数创建一个简易对话框（顶层窗口）。
- from tkinter import *
- def Msg():
- top=Toplevel(width=400,height=200)
- Label(top,text='Python').pack()
- w=Tk()
- Button(w,text='OK',command=Msg).pack()
- w.mainloop()



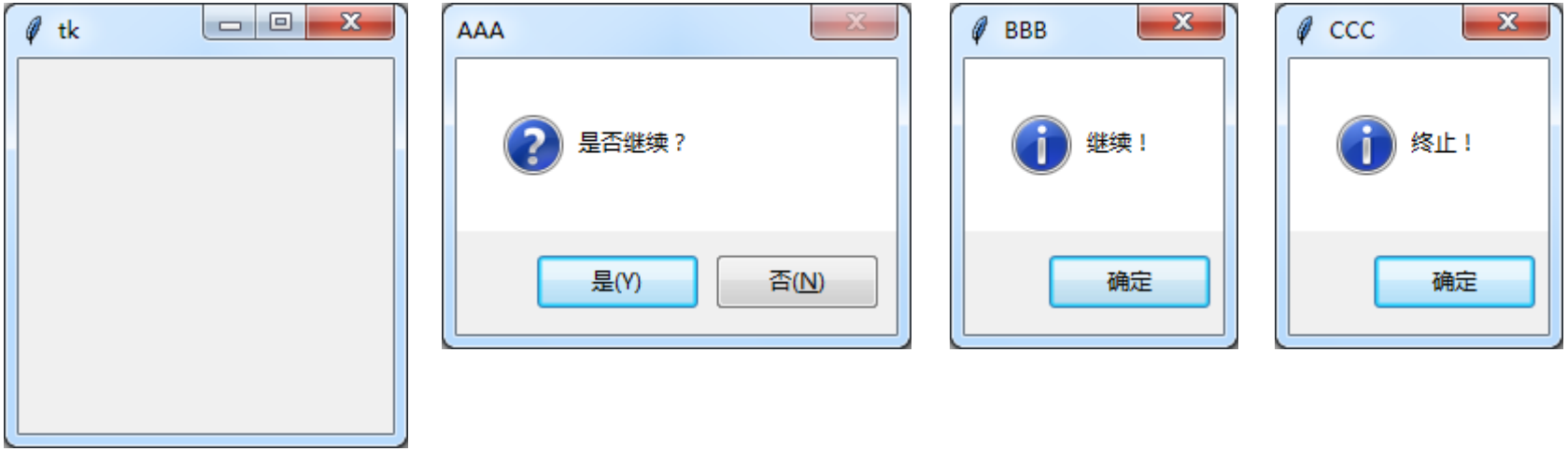
13.4.2 标准对话框

- tkinter模块还提供一些子模块用于创建通用的标准对话框。

1.messagebox子模块

- messagebox子模块提供一系列用于显示信息或进行简单对话的消息框，可通过调用函数askyesno()、askquestion()、askyesnocancel()、askokcancel()、askretrycancel()、showerror()、showinfo()和showwarning()来创建。
- 例：

- from tkinter.messagebox import *
- ask=askyesno(title='AAA',message='是否继续？')
- if ask:
- 　showinfo(title='BBB',message='继续！')
- else:
- 　showinfo(title='CCC',message='终止！')

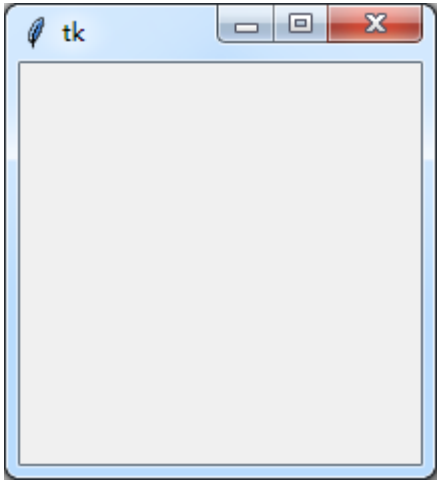


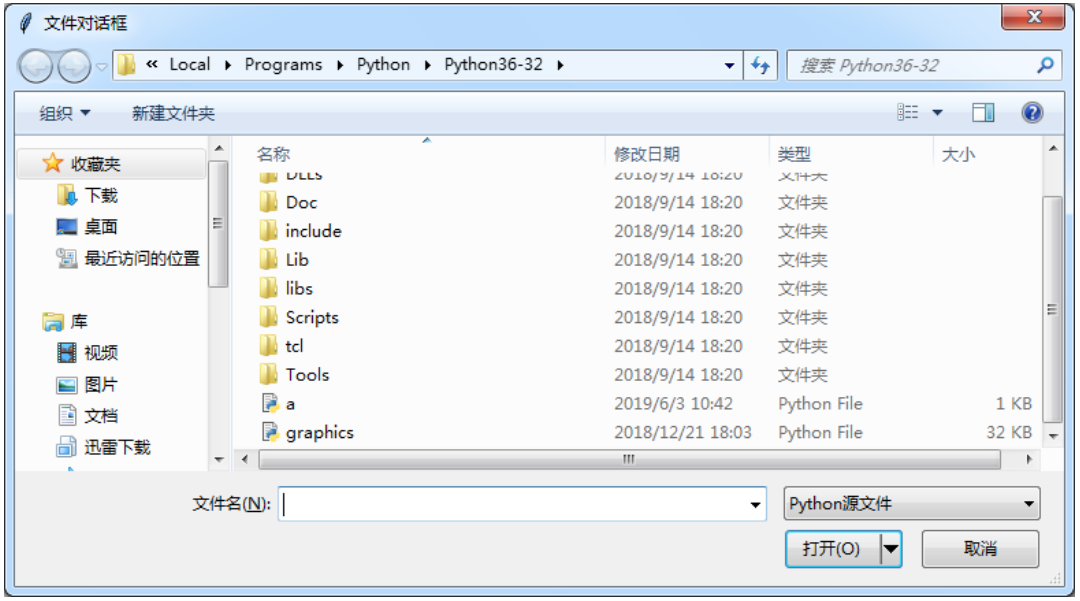
```
import tkinter as tk
from tkinter import messagebox as m
def b1f():m.showinfo('Notice', 'constructure information!')
def b2f():m.showwarning('Warning', 'This is a warning tips!')
def b3f():m.showerror('Error', 'Error message')
def b4f():m.askquestion('Question', 'Is or Is not?')
def b5f():m.askokcancel('Ask', 'Ask for confirming!')
def b6f():m.askyesno('Choose', 'Yes or No?')
def b7f():m.askretrycancel('Retry', 'Retry or cancel')
top=tk.Tk()
top.title("MsgBox Test")
b1=tk.Button(top,text="showinfo",command=b1f)
b1.pack(fill=tk.X)
b2=tk.Button(top,text="showwarning",command=b2f)
b2.pack(fill=tk.X)
b3=tk.Button(top,text="showerror",command=b3f)
b3.pack(fill=tk.X)
b4=tk.Button(top,text="askquestion",command=b4f)
b4.pack(fill=tk.X)
b5=tk.Button(top,text="askokcancel",command=b5f)
b5.pack(fill=tk.X)
b6=tk.Button(top,text="askyesno",command=b6f)
b6.pack(fill=tk.X)
b7=tk.Button(top,text="askretrycancel",command=b7f)
b7.pack(fill=tk.X)
top.mainloop()
```

2.filedialog子模块

- filedialog子模块提供用于文件浏览、打开和保存的对话框，可通过调用函数askopenfilename()、asksaveasfilename()等函数来创建。
- 例：

- from tkinter.filedialog import *
- askopenfilename(title='文件对话框',\
- filetypes=[('Python源文件','.py')])

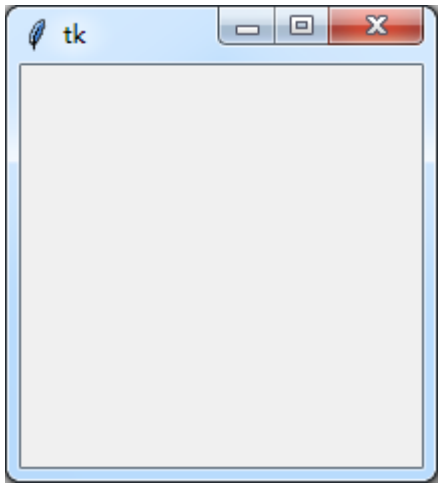


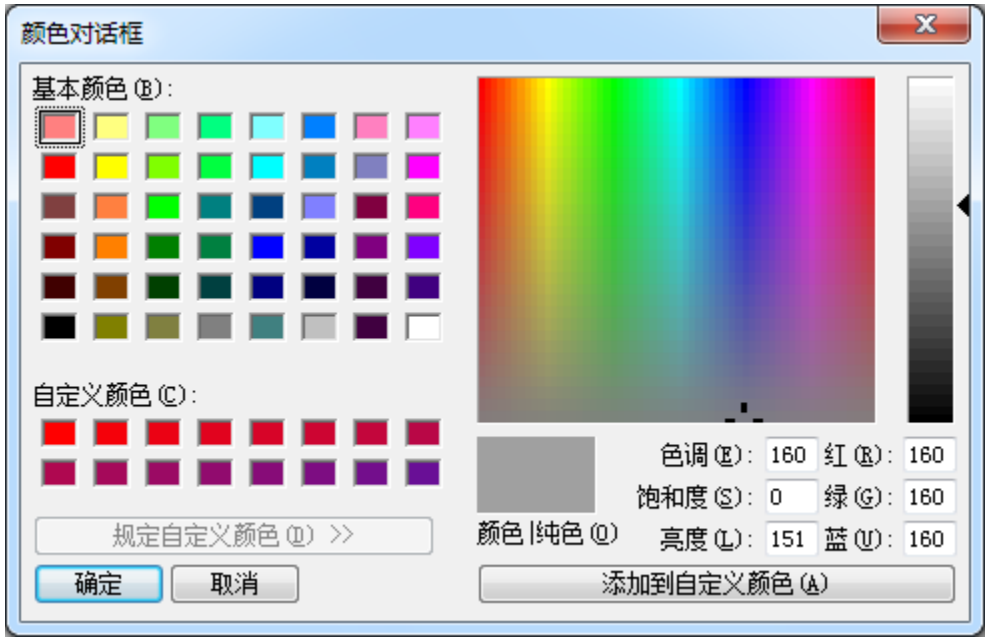


3.colorchooser子模块

- colorchooser子模块提供用于选择颜色的对话框，可通过函数askcolor()来创建。
- 例：

- from tkinter.colorchooser import *
- askcolor(title='颜色对话框')





```
from tkinter import *
from tkinter import filedialog as fd
from tkinter import colorchooser as cc
def b1f():
    fd.askopenfilename(title='Open file dialog',filetypes=[('Python源文件','.py')])
def b2f():
    fd.asksaveasfilename(title='Save file dialog',filetypes=[('Python源文件','.py')])
def b3f():
    cc.askcolor(title='Color dialog')
w=Tk()
w.title("Test")
b1=Button(w,text="askopenfilename",command=b1f)
b1.pack(fill=X)
b2=Button(w,text="asksaveasfilename",command=b2f)
b2.pack(fill=X)
b3=Button(w,text="askcolor",command=b3f)
b3.pack(fill=X)
w.mainloop()
```

13.5事件处理

- 利用各种控件以及对象的布局方法，可以设计应用程序用户界面的外观部分。
- 图形用户界面应用程序与一般字符界面应用程序的重要区别是，程序的执行与界面对象的事件相关联，即一种新的程序执行模式——事件驱动。

13.5.1 事件处理程序

- 用户通过键盘或鼠标与图形用户界面交互操作时，会触发各种事件（event）。
- 事件发生时需要应用程序对它做出响应或进行处理。
- tkinter模块中定义了很多种事件，用以支持图形用户界面应用程序开发。

- 1. 事件的描述
- tkinter事件可以用特定形式的字符串来描述，一般形式是:
- <修饰符>-<类型符>-<细节符>
- 其中，修饰符用于描述鼠标的单击、双击，以及键盘组合按键等情况；
- 类型符指定事件类型，最常用的类型有分别表示鼠标事件和键盘事件的Button和Key；

- <修饰符>-<类型符>-<细节符>
- 细节符指定具体的鼠标键或键盘按键，如鼠标的左、中、右三个键分别用1、2、3表示，键盘按键用相应字符或按键名称表示。
- 修饰符和细节符是可选的，而且事件经常可以使用简化形式。
- 例：<Double-Button-1>描述符中，修饰符是Double，类型符是Button，细节符是1，综合起来描述的事件就是双击鼠标左键。

（1）常用鼠标事件

<ButtonPress-1>:按下鼠标左键，可简写为**<Button-1>**或**<1>**。

类似的有**<Button-2>**（按下鼠标中键）和**<Button-3>**（按下鼠标右键）。

<B1-Motion>:按下鼠标左键并移动鼠标。类似的有**<B2-Motion>**和**<B3-Motion>**。

<Double-Button-1>:双击鼠标左键。

<Enter>:鼠标指针进入控件。

<Leave>:鼠标指针离开控件。

```
from tkinter import *
app = Tk()
def callback(event):
    print("当前位置：",event.x,event.y)
frame = Frame(app, width = 200, height = 200)
frame.bind("<Button-1>",callback)
frame.bind("<Button-2>",callback)
frame.bind("<Button-3>",callback)
frame.pack()
mainloop()
```

```
from tkinter import *
def f(event):
    s.config(text="鼠标单击位置： x="+str(event.x)+",y="+str(event.y))
    c.place(x=event.x-5,y=event.y-5)
    c.itemconfig(ALL,state=NORMAL)
def rf(event):
    s.config(text="")
    c.itemconfig(ALL,state=HIDDEN)
w=Tk()
w.title("测试")
w.geometry("300x300+100+100")
s=Label(w,text="单击鼠标左键显示，单击鼠标右键隐藏！ ")
s.pack(side=TOP)
c=Canvas(w,width=10,height=10)
m=c.create_line(0,5,10,5)
n=c.create_line(5,0,5,10)
c.itemconfig(ALL,state=HIDDEN)
w.bind("<Button-1>",f)
w.bind("<Button-3>",rf)
w.mainloop()
```

（2）常用键盘事件

- **<KeyPress-a>**:按下a键。可简写为<Key-a>或a（不用尖括号）。可显示字符（字母、数字和标点符号）都可像字母a这样使用，但有两个例外:空格键对应的事件是<space>，小于号键对应的事件是<less>。
- 注意:不带尖括号的数字（例如1）是键盘事件，而带尖括号的数字（例如<1>）是鼠标事件。

- **<Return>**:按下回车键。不可显示字符都可像回车键这样用<键名>表示对应事件，例如<Tab>、<shift_L>、<Control_R>、<Up>、<Down>、<F1>等。
- **<Key>**:按下任意键。
- **<shift-Up>**:同时按下shift键和↑键。类似的还有Alt键组合、Ctrl键组合。

```
from tkinter import *
def callback(event):
    print(event.char)
app=Tk()
frame=Frame(app,width=200,height=200)
frame.bind("<Key>",callback)
frame.focus_set()
frame.pack()
mainloop()
```

```
from tkinter import *
def callback(event):
    print(event.keysym)
root=Tk()
frame=Frame(root,width=200,height=200)
frame.bind("<Key>",callback)
frame.focus_set()
frame.pack()
mainloop()
```

2. 事件对象

- 每个事件都导致系统创建一个事件对象，并将该对象传递给事件处理函数。
- 事件对象具有描述事件的属性，常用的属性如下:
- **x和y**:鼠标单击位置相对于控件左上角的坐标，单位是像素。
- **x_root和y_root**:鼠标单击位置相对于屏幕左上角的坐标，单位是像素。

- num:单击的鼠标键号，1、2、3分别表示左、中、右键。
- char:如果按下可显示字符键，此属性是该字符。如果按下不可显示键，此属性为空串。
- 例：按下任意键都可触发<Key>事件，在事件处理函数中可以根据传递来的事件对象的char属性来确定具体按下的是哪一个键。

- keysym:如果按下可显示字符键，此属性是该字符。如果按下不可显示键，此属性设置为该键的名称。例：回车键是Return、插入键是Insert、光标上移键是Up。
- keycode:所按键的ASCII码。注意:此编码无法得到上挡字符的ASCII码。
- keysym_num:这是keysym的数值表示。对普通单字符键来说，就是ASCII码。

```
from tkinter import *
def main():
    tk=Tk()
    canvas=Canvas(tk,width=400,height=400)
    canvas.pack()
    m=PhotoImage(file="d:\\0.gif")
    canvas.create_image(10,10,anchor=NW,image=m)
    def f(event):
        if event.keysym == "Up": canvas.move(x,0,-5)
        elif event.keysym=="Down": canvas.move(x,0,5)
        elif event.keysym=="Left": canvas.move(x,-5,0)
        elif event.keysym=="Right": canvas.move(x,5,0)
        else: canvas.move(x,5,5)
    x=canvas.create_rectangle(200,200,220,220,fill="red")
    canvas.bind_all("<KeyPress-Up>",f)
    canvas.bind_all("<KeyPress-Down>", f)
    canvas.bind_all("<KeyPress-Left>",f)
    canvas.bind_all("<KeyPress-Right>", f)
    canvas.bind_all("<KeyPress-Return>",f)
    tk.mainloop()
main()
```

3. 事件处理函数的一般形式

- 事件处理函数是在触发了某个对象的事件时而调用执行的程序段，它一般都带一个event类型的形参，触发事件调用事件处理函数时，将传递一个事件对象。

- 事件处理函数的一般形式如下:
- def 函数名(event):
- 　函数体
- 在函数体中可以调用事件对象的属性。
- 事件处理函数在应用程序中定义，但不由应用程序调用，而是由系统调用，所以一般称为回调（call back）函数。

13.5.2 事件绑定

- 图形用户界面应用程序的核心是对各种事件的处理程序。
- 应用程序一般在完成建立图形界面工作后都会进入一个事件循环，等待事件发生并触发相应的事件处理程序。
- 事件与相应事件处理程序之间是通过绑定建立关联的。

1. 事件绑定的方式

- 在tkinter模块中有四种不同的事件绑定方式。

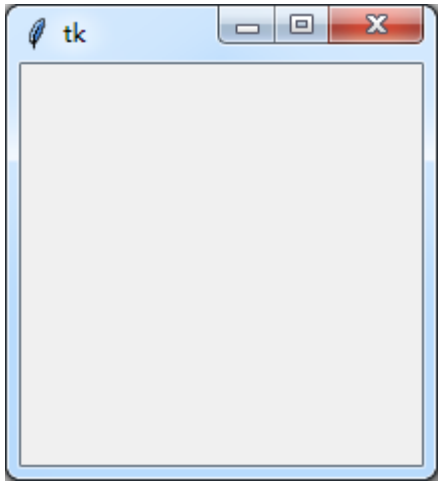
（1）对象绑定和窗口绑定

- 对象绑定最常见的是事件绑定方式。
- 针对某个控件对象进行事件绑定称为对象绑定，也称为实例绑定。
- 对象绑定只对该控件对象有效，对其他对象（即使是同类型的对象）无效。

- 对象绑定调用控件对象的bind()方法实现，一般形式如下:
- 控件对象.bind(事件描达符，事件处理程序)
- 其该语句的含义是，若控件对象发生了与事件描述符相匹配的事件，则调用事件处理程序。
- 调用事件处理程序时，系统会传递一个Event类的对象作为实际参数，该对象描述了所发生事件的详细信息。

- 对象绑定的一种特例是窗口绑定（窗口也是一种对象），此时绑定对窗口（主窗口或项层窗口）中的所有控件对象有效，用窗口的bind()方法实现。
- 例：窗口绑定应用示例。
- 把主窗口与<Button-1>事件进行绑定，对应的事件回调函数是callback()，每当单击主窗口时，都将触发执行callback()。

- from tkinter import *
- def callback(event):
- print("clicked at",event.x,event.y)
- w=Tk()
- w.bind("<Button-1>",callback)
- w.mainloop()




```
clicked at 5 7
clicked at 93 86
clicked at 192 190
```

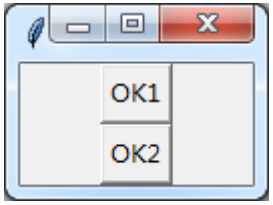
- 调用callback()函数时，将一个描述事件的Event类对象作为参数传递给该函数，该函数从事件对象参数中提取单击位置信息并在Python解释器窗口中输出单击位置的坐标信息。
- 如果在主窗口中有控件对象，则窗口鼠标单击事件对控件对象也有效。

（2）类绑定和应用程序绑定

- 除了对象绑定和窗口绑定外，tkinter模块还提供了其他两种事件绑定方式:类绑定和应用程序绑定。
- 类绑定针对控件类，故对该类的所有对象有效，可用任何控件对象的bind_class()方法实现，一般形式如下:
- 控件对象.bind_class (控件类描述符,事件描述符,事件处理程序)

- 例：类绑定应用示例。
- 为Button类绑定鼠标单击事件，使得所有按钮对象都以同样方式响应鼠标单击事件，执行同一事件处理函数。

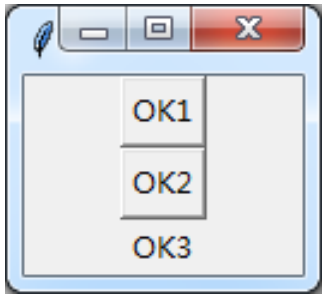
```
from tkinter import *
def callback(event):
    print("Python")
w=Tk()
b1=Button(w,text="OK1")
b2=Button(w,text="OK2")
b1.bind_class("Button","<Button-1>",callback)
b1.pack()
b2.pack()
w.mainloop()
```



Python
Python

- 应用程序绑定对应程序中的所有控件都有效。
- 用任意控件对象的bind_all()方法实现，一般形式如下:
- 控件对象.bind_all(事件描述符,事件处理程序)
- 例：应用程序绑定应用示例。
- 通过应用程序绑定，使得窗口的全部对象执行同样的事件处理函数。

```
from tkinter import *
def callback(event):
    print("Python")
w=Tk()
b1=Button(w,text="OK1")
b2=Button(w,text="OK2")
l=Label(w,text="OK3")
b1.bind_all("<Button-1>",callback)
b1.pack()
b2.pack()
l.pack()
w.mainloop()
```



Python
Python
Python

2.键盘事件与焦点

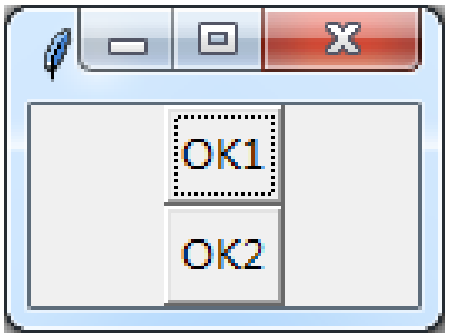
- 所谓焦点（focus）就是当前正在操作的对象，例如，用鼠标单击某个对象，该对象就成为焦点。
- 当用户按下键盘中的一个键时，要求焦点在所期望的位置。

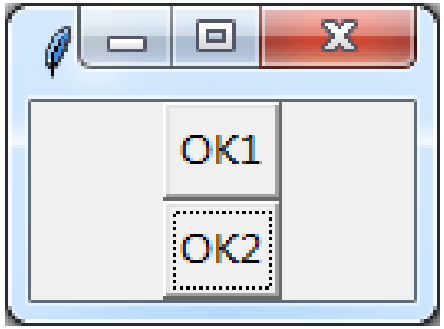
- 图形用户界面中有唯一焦点，任何时刻可以通过对象的focus_set()方法来设置，也可以用键盘上的Tab键来移动焦点。
- 键盘事件处理比鼠标事件处理多了一个设置焦点的步骤。

- 例：焦点应用示例。
- 创建两个按钮控件，按钮都与按任意键事件<Key>进行绑定，事件处理程序是回调函数show1()和show2()。
- 用b.focus_set()语句将“OK1”按钮设为焦点，以后可以用Tab键来移动焦点。
- 当按下任何键时，由焦点所在按钮响应键盘事件并调用相应的回调函数。
- 回调函数中分别显示按键所对应的字符，对于字母分别输出其小写和大写。

```
from tkinter import *
def show1(event):
    print("pressed",(event.char).lower())
def show2(event):
    print("pressed",(event.char).upper())
w=Tk()
b1=Button(w,text='OK1')
b1.bind('<Key>',show1)
b2=Button(w,text='OK2')
b2.bind('<Key>',show2)
b1.focus_set()
b1.pack()
b2.pack()
w.mainloop()
```

```
pressed a
pressed b
pressed c
pressed
pressed A
pressed B
pressed C
```

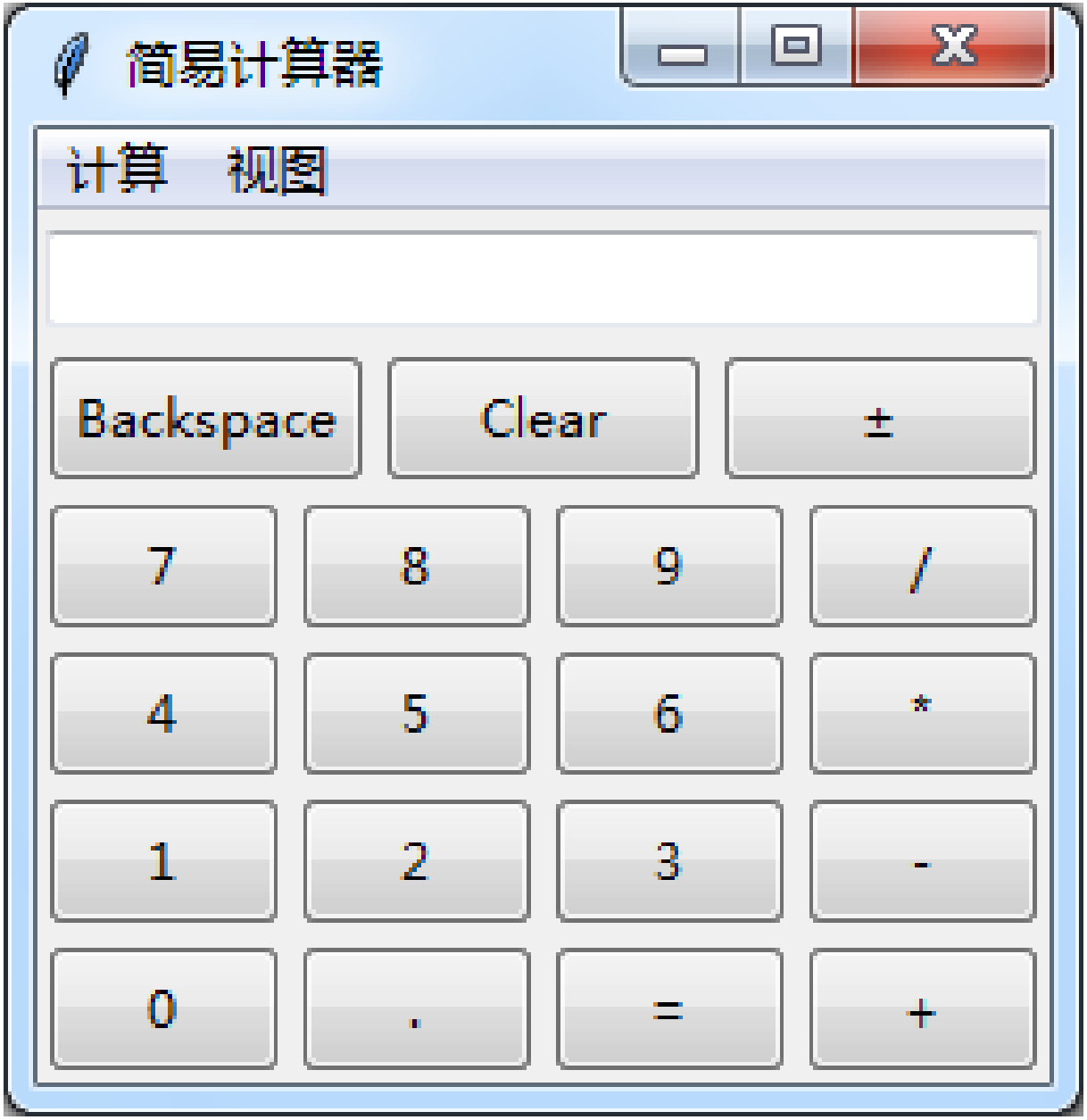




```
    def f1():
      global n
      n=n+1
      if n>3:
        showinfo(title="提示",message='你已经连续错了3次！')
        v1.set("")
        v2.set("")
        b1.config(state=DISABLED)
        b2.config(state=DISABLED)
        t1.config(state=DISABLED)
        t2.config(state=DISABLED)
        w.quit()
      else:
        x=v1.get()
        y=v2.get()
        if x=="aaa" and y=="123":
          showinfo(title="欢迎你！",message='欢迎进入本系统！')
          n=0
          w.quit()
        else:
          showinfo(title="提示",message='账号或密码错误！')
          f2()
    def f2():
      v1.set("")
      v2.set("")
      t1.focus_set()
    from tkinter import *
    from tkinter.messagebox import *
    w=Tk()
    w.title("系统登录")
    n=0
    v1=StringVar()
    v2=StringVar()
    m1=Frame(w)
    m2=Frame(w)
    m3=Frame(w)
    m1.grid(row=0,column=0)
    m2.grid(row=0,column=1)
    m3.grid(row=1,column=0,columnspan=2)
    Label(m1,text="账号").grid(row=0,column=0)
    Label(m1,text="密码").grid(row=1,column=0)
    t1=Entry(m2,width=20,textvariable=v1)
    t2=Entry(m2,width=20,textvariable=v2,show='*')
    t1.grid(row=0,column=0)
    t2.grid(row=1,column=0)
    b1=Button(m3,text="提交",command=f1)
    b2=Button(m3,text="重填",command=f2)
    b1.grid(row=0,column=0,padx=5,pady=5)
    b2.grid(row=0,column=1,padx=5,pady=5)
    w.mainloop()
```

13.6 图形用户界面应用举例

- 例：设计并实现一个简易计算器。
- 要求:计算器能进行加、减、乘、除运算，具有退格键（Backspace）、清除键（Clear）和负号键（±）。计算器有菜单栏，其中有“计算”和“视图”两个菜单，分别有“退出”命令和“显示千位分隔符”复选框（Checkbutton）菜单项。


简易计算器
计算
视图
Backspace
Clear
±
7
8
9
/
4
5
6
*
1
2
3
-
0
.
=
+

```
from tkinter import *
from tkinter.ttk import *
#将框架（Frame）的共同属性作为默认值，以简化创建过程
def my_frame(master):
    w=Frame(master)
    w.pack(side=TOP,expand=YES,fill=BOTH)
    return w
#将按钮（Button）的共同属性作为默认值，以简化创建过程
def my_button(master,text,command):
    w=Button(master,text=text,command=command,width=6)
    w.pack(side=LEFT,expand=YES,fill=BOTH,padx=2,pady=2)
    return w
```

```
#将数字串最末的字符删除并返回
def back(text):
    if len(text)>0:
        return text[:-1]
    else:
        return text
#利用eval函数计算表达式字符串的值
def calc(text):
    try:
        if sep_flag.get()==0:
            return eval(del_sep(text))
        else:
            return add_sep(str(eval(del_sep(text))))
    except (SyntaxError,ZeroDivisionError,NameError):
        return 'Error'
```

- #向参数传入的数字串中添加千位分隔符，分三种情况:纯小数部份、
- #纯整数部份、同时有整数和小数部份。由于字符串是不可改变的，所以
- #先由字符串生成列表，以便执行insert操作和extend操作，操作完成后
- #再由列表生成字符串返回

```
def add_sep(text):
    dot_index=text.find('.')
    if dot_index>0:
        text_head=text[:dot_index]
        text_tail=text[dot_index:]
    elif dot_index<0:
        text_head=text
        text_tail=''
    else:
        text_head=''
        text_tail=text
    list_=[char for char in text_head]
    length=len(list_)
    tmp_index=3
    while length-tmp_index>0:
        list_.insert(length-tmp_index,',')
        tmp_index += 3
    list_.extend(text_tail)
    new_text=''
    for char in list_:
        new_text+=char
    return new_text
```

- #删除数字串中所有的千位分隔符
- def del_sep(text):
- 　return text.replace(',','')
- #开始计算器界面的实现
- wind=Tk()
- wind.title("简易计算器") #设置主窗口标题
- main_menu=Menu(wind) #创建最上层主菜单
- #创建“计算”菜单项，并加入到主菜单
- calc_menu=Menu(main_menu,tearoff=0)
- calc_menu.add_command(label='退出',command=lambda:exit())
- main_menu.add_cascade(label='计算',menu=calc_menu)
- #创建“视图”菜单，并加入到主菜单，其中“显示千位分隔符”菜单项
- #是一个Checkbutton

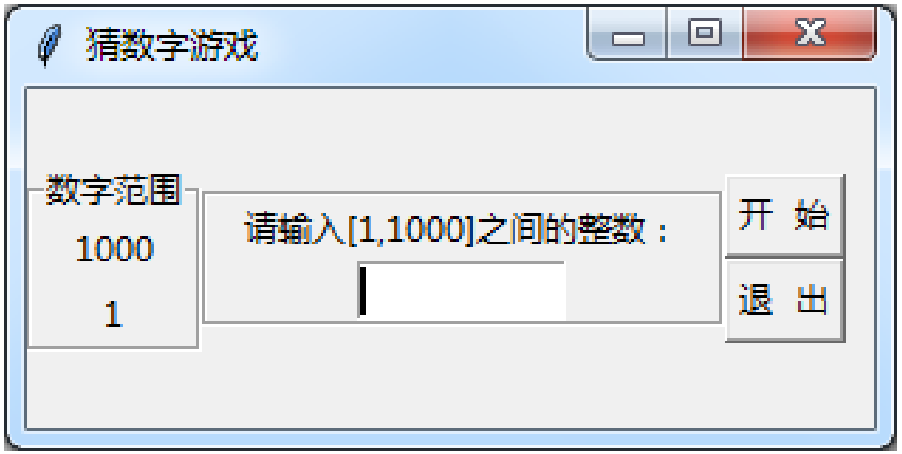
```
text=StringVar()
sep_flag=IntVar()
sep_flag.set(0)
view_menu=Menu(main_menu,tearoff=0)
view_menu.add_checkbutton(label='显示千位分隔符
',variable=sep_flag,\
    command=lambda t=text:t.set(add_sep(t.get())))
main_menu.add_cascade(label='视图',menu=view_menu)
wind['menu']=main_menu    #将主菜单与主窗口wind绑定
#创建文本框
Entry(wind,textvariable=text).pack(expand=YES,fill=BOTH,\
    padx=2,pady=4)
#创建ttk子模块的Style对象，设置按钮内边距
style=Style()
style.configure('TButton',padding=3)
#创建第一行三个按钮
```

```
fedit=my_frame(wind)
my_button(fedit,'Backspace',lambda t=text:t.set(back(t.get())))
my_button(fedit,'Clear',lambda t=text:t.set(''))
my_button(fedit,'±',lambda t=text:t.set('-('+t.get()+')'))
#创建其余四行按钮，每行四个
for key in ('789/','456*','123-','0.=+'):
    fsymb=my_frame(wind)
    for char in key:
        if char=='=':
            my_button(fsymb,char,\
                lambda t=text:t.set(calc(t.get())))
        else:
            my_button(fsymb,char,\
                lambda t=text,c=char:t.set(t.get()+c))
wind.mainloop()
```


简易计算器
计算
视图
退出
Backspace
Clear
±
7
8
9
/
4
5
6
*
1
2
3
-
0
.
=
+


简易计算器
计算
视图
显示千位分隔符
Backspace
Clear
±
7
8
9
/
4
5
6
*
1
2
3
-
0
.
=
+

- 例：猜数字游戏。
- from tkinter import *
- import random
- number=random.randint(1,1000)
- running,num,nmax,nmin=True,0,1000,1
- def btnClose(event):
- 　　w.destroy()　　#退出主窗口



```
def btnGuess(event):
    global nmax,nmin,num,running
    if running:
        val1=int(ent.get())
        if val1==number:
            lbl_qv("恭喜，答对了！")
            num+=1
            running=False
            numGuess()
        elif val1<number:
            if val1>nmin:
                nmin=val1
                num+=1
                lbl_min.config(lbl_min,text=nmin)
            lbl_qv("猜小了，请重新输入！")
        else:
            if val1<nmax:
                nmax=val1
                num+=1
                lbl_max.config(lbl_max,text=nmax)
            lbl_qv("猜大了，请重新输入！")
    else:
        lbl_qv("已经答对了！")
```

```
def numGuess():
    if num==1:
        lbl_qv("恭喜，一次答对！")
    elif num<=15:
        lbl_qv("答对了，尝试次数:"+str(num))
    else:
        lbl_qv("超过15次了")
def lbl_qv(vtxt):
    lbl_q.config(lbl_q,text=vtxt)
```

- w=Tk()
- w.title("猜数字游戏")
- w.geometry("300x120+200+200")
- frm1=LabelFrame(w,text="数字范围")
- lbl_max=Label(frm1,text=nmax)
- lbl_max.pack(side=TOP,fill=X)
- lbl_min=Label(frm1,text=nmin)
- lbl_min.pack(side=BOTTOM,fill=X)
- frm1.pack(side=LEFT,fill=X)
- frm2=LabelFrame(w)
- lbl_q=Label(frm2,width=25)
- lbl_qv("请输入[1,1000]之间的整数:")
- lbl_q.pack(side=TOP)
- ent=Entry(frm2,width=10)
- ent.pack(side=BOTTOM)
- ent.bind("<Return>",btnGuess)

- frm2.pack(side=LEFT,fill=X)
- frm3=Frame(w,relief="groove")
- btnG=Button(frm3,text="开 始")
- btnG.bind("<Button-1>",btnGuess)
- btnG.pack(side=TOP)
- btnC=Button(frm3,text="退 出")
- btnC.bind("<Button-1>",btnClose)
- btnC.pack(side=BOTTOM)
- frm3.pack(side=LEFT,fill=X)
- ent.focus_set()
- w.mainloop()

自测题

一、选择题

- 1．下列控件类中，可用于创建单行文本框的是（　　）。C
- A．Button　　B．Label　　C．Entry　　D．Text
- 2．如果要输入学生的兴趣爱好，比较好的方法是采用（　　）。B
- A．单选按钮　　B．复选框
- C．文本框　　D．列表框

- 3．如果要输入学生的性别，比较好的方法是采用（　　）。A
- A．单选按钮　　B．复选框　　C．文本框　　D．列表框
- 4．为使tkinter模块创建的按钮起作用，应在创建按钮时，为按钮控件类的（　　）方法指明回调函数或语句。D
- A．pack　　B．command　　C．text　　D．bind

- 5．关于tkinter主窗口和顶层窗口（Toplevel对象）关系的描述，错误的是（　　）。C
- A．关闭主窗口，则自动关闭顶层窗口
- B．创建顶层窗口，则自动创建主窗口
- C．顶层窗口和主窗口是相互独立的
- D．顶层窗口不能脱离主窗口而存在
- 6．下列选项中，可用于将tkinter模块创建的控件放置于窗体的是（　　）。A
- A．pack　　B．show　　C．set　　D．bind

- 7. 事件<Button-1>表示（　　）。B
- A. 单击鼠标右键　　B. 单击鼠标左键
- C. 双击鼠标右键　　D. 双击鼠标左键
- 8. 以下表示按下回车键事件的是（　　）。D
- A. <↙>　　B. <回车>　　C. <Enter>
  D. <Return>

- 9．在下列程序运行后按回车键，此时出现的结果是（　　）。B
- def cb1():
- 　print('button1')
- def cb2(Event):
- 　print('button2')
- from tkinter import *
- w=Tk()
- b1=Button(w,text='Button1',command=cb1)
- b2=Button(w,text='Button2')
- b2.bind("<Return>",cb2)
- b1.pack()
- b2.pack()
- b2.focus_set()
- w.mainloop()
- A．button1　　B．button2　　C．Button1　　D．Button2

- 10．关于下列程序运行结果的描述，正确的是（　　）。C
- def hf():
- 　tkinter.messagebox.showinfo("Hello","Python Programming!")
- import tkinter
- import tkinter.messagebox
- win=tkinter.Tk()
- tkinter.Button(win,text="开始",command=hf).pack()
- win.mainloop()
- A．在主窗口显示“Hello”和“Python Programming!”两行文字
- B．单击“开始”按钮后在主窗口中显示“Python Programming!”文字
- C．单击“开始”按钮弹出“Hello”信息提示框
- D．单击“Hello”按钮弹出“开始”信息提示框

二、填空题

- 1．通过控件的（ ）和（ ）属性，可以设置控件的宽度和高度。**width，height**
- 2．通过控件的（ ）属性，可以设置内容停靠位置；通过控件的（ ）属性，可以设置其显示的内容；通过（ ）属性，可指定多行的对齐方式。**anchor，text，justify**

- 3．通过控件的（ ）属性，可以设置其3D显示样式；通过控件的（ ）或（ ）属性，可以设置其边框宽度。relief，borderwidth，bd
- 4．通过控件的（ ）和（ ）属性，可以设置其显示内容与边框之间的填充宽度和高度；通过控件的（ ）属性，可以绑定StringVar对象到控件。padx，pady，textvariable

- 5．tkinter模块提供了三种几何布局管理器，它们分别是（ ）、（ ）和（ ）。pack，grid，place
- 6．利用tkinter模块中的子模块（ ）、（ ）和（ ），可以创建通用的标准对话框。messagebox，filedialog，colorchooser

- 7．用户实施的某个操作就引发一个（ ）。就操作的设备来说，常见的事件有（ ）事件和（ ）事件。事件，鼠标，键盘
- 8．需要用户输入一个特定范围内的值时，可以使用（ ）控件或（ ）控件。单行文本框，刻度条

三、问答题

- 1．创建图形用户界面的步骤是什么？
- 2．控件类和控件对象有何区别？
- 3．Python有哪些常用控件？它们的作用是什么？设置控件的属性有哪些方法？
- 4．菜单有哪些类型？各自的设计方法是什么？
- 5．什么叫事件绑定？事件绑定的方式有哪些？