



# Python程序设计

## —从基础到开发

# 目录

---

## 第7章 Tkinter图形界面设计

1	Python图形开发库
2	常用Tkinter组件的使用
3	图形绘制
4	Tkinter字体
5	Python事件处理
6	图形界面程序设计的应用

# (示例) tkinter.pack实现的界面



# 7.1 Python图形开发库

---

## 常用Python GUI库

- Python提供了多个图形开发界面的库，几个常用Python GUI库如下：
- (1) **Tkinter**：该模块是Python的标准Tk GUI工具包接口。可以在应用在 Unix/Windows/Macintosh系统中。
  - (2) **wxPython**：wxPython 是一款开源软件，是 Python 语言的一套优秀的 GUI 图形库，允许 Python 程序员很方便的创建完整的、功能健全的 GUI 用户界面。
  - (3) **Jython**：Jython程序可以和Java无缝集成。除了一些标准模块，Jython 使用Java的模块。Jython几乎拥有标准的Python中不依赖于C语言的全部模块。

# 7.1 Python图形开发库

## 7.1.1 创建Windows窗口

- Tkinter是Python的标准GUI库，内置到Python安装包中，所以安装Python后，只要import Tkinter，就可以使用这个GUI图形开发库。
- Python的IDEL也是用Tkinter编写的。
- 【例】使用Tkinter创建一个Windows窗口的GUI程序。

```
import tkinter          #导入Tkinter模块
win = tkinter.Tk()       #创建Windows窗口对象
win.title('我的第一个GUI程序') #设置窗口标题
win.mainloop()          #进入消息循环，也就是显示窗口
```



# 7.1 Python图形开发库

## 7.1.1 创建Windows窗口

- 在创建Windows窗口对象后，可以使用geometry()方法设置窗口的大小，格式如下：窗口对象.geometry(size)
- 【例】创建并显示一个Windows窗口，初始大小为800\*600

这种方式导入模块，使用模块内的函数时，可以不用带模块名字，但是不能有不同模块有相同函数名字的。

**不建议使用这种通配符的导入方式**

```
from tkinter import *  
win = Tk()  
win.title('我的第一个GUI程序')  
win.geometry("800x600")  
win.minsize(400, 600)  
win.maxsize(1440, 800)  
win.mainloop()
```

```
#导入Tkinter模块  
#创建Windows窗口对象  
#设置窗口标题  
#设置窗口最小尺寸  
#设置窗口最大尺寸  
#进入消息循环，也就是显示窗口
```

# 7.1 Python图形开发库

## 7.1.1 创建Windows窗口

- 在创建Windows窗口对象后，可以使用geometry()方法设置窗口的大小，格式如下：窗口对象.geometry(size)
- 【例】创建并显示一个Windows窗口，初始大小为800\*600

建议这种方式导入

```
import tkinter as tk  
win = tk.Tk()  
win.title('我的第一个GUI程序')  
win.geometry("800x600")  
win.minsize(400, 600)  
win.maxsize(1440, 800)  
win.mainloop()
```

```
#导入Tkinter模块  
#创建Windows窗口对象  
#设置窗口标题  
#设置窗口最小尺寸  
#设置窗口最大尺寸  
#进入消息循环，也就是显示窗口
```

# 7.1 Python图形开发库

## 7.1.2 几何布局管理器

- Tkinter几何布局管理器用于组织和管理父组件（往往是窗口）中，子组件的布局方式。
- Tkinter提供了3种不同风格的几何布局管理类：
  - ❑ pack
  - ❑ grid
  - ❑ place
- 注意这三种布局管理在同一个 master window 里一定不可以混用!
- 布局管理有以下功能：
  - ❑ 将控件放置在屏幕上，包括控件的位置及控件的大小
  - ❑ 将控件注册到本地窗口系统中
  - ❑ 管理控件在屏幕上的显示



## 7.1.2 几何布局管理器



### 1. pack几何布局管理器

- pack几何布局管理器采用块的方式组织组件。根据组件的生成顺序，快速放在界面中。pack 是三种布局管理中最常用的。

```
import tkinter as tk
root = tk.Tk()
tk.Label(root, text="Red Sun", bg="red", fg="white").pack()
tk.Label(root, text="Green Grass", bg="green", fg="black").pack()
tk.Label(root, text="Blue Sky", bg="blue", fg="white").pack()
root.mainloop()
```

- Pack布局函数的语法：**对象.pack (option)**

# 7.1.2 几何布局管理器

## 1. pack几何布局管理器

- Pack布局函数的语法: **子组件.pack (option)**
- Pack方法的参数选项如下表:

选项	描述	取值范围
side	停靠在父组件的哪一边上	top (默认) , bottom, left, right
anchor	锚点位置 (停靠位置) , 对应于东南西北以及四个角	n, s, e, w, nw, sw, se, ne, center (默认)
fill	填充空间	'x', 'y', 'both', none
expand	扩展空间	0或1
ipadx, ipady	组件内部在x/y方向上填充的空间大小	单位为c(厘米), m(毫米), i(英寸), p(打印机的点)
padx, pady	组件外部在x/y方向上填充的空间大小	单位为c(厘米), m(毫米), i(英寸), p(打印机的点)

## 7.1.2 几何布局管理器

### 1. pack几何布局管理器

➤ fill 控件填充方式



选项	描述	取值范围
fill	填充空间	'x', 'y', 'both', none

```
import tkinter as tk
root = tk.Tk()
w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill='x')
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill='x')
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill='x')
root.mainloop()
```

## 7.1.2 几何布局管理器

### 1. pack几何布局管理器

#### ➤ fill 控件填充方式



选项	描述	取值范围
fill	填充空间; 当属性side='top'或'bottom'时, 填充x方向; 当属性side='left'或'right'时, 填充y方向; 当expand选项为'yes'或1时, 填充父组件的剩余空间。	'x', 'y', 'both', none

```
import tkinter as tk
root = tk.Tk()
w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack(side = 'left', fill='y')
root.mainloop()
```

## 7.1.2 几何布局管理器



### 1. pack几何布局管理器

- Pack 可以在四个方面控制控件边距: 内边距, 外边距, 水平边距, 垂直边距
- **padx** - 设置水平方向的外边距

```
import tkinter as tk
root = tk.Tk()
w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill='x', padx=10)
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill='x', padx=10)
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill='x', padx=10)
root.mainloop()
```

# 7.1.2 几何布局管理器

## 1. pack几何布局管理器

- Pack 可以在四个方面控制控件边距: 内边距, 外边距, 水平边距, 垂直边距
- **pady** - 设置竖直方向的外边距

```
import tkinter as tk
root = tk.Tk()
w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill=X, pady=10)
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill=X, pady=10)
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill=X, pady=10)
root.mainloop()
```



## 7.1.2 几何布局管理器

### 1. pack几何布局管理器

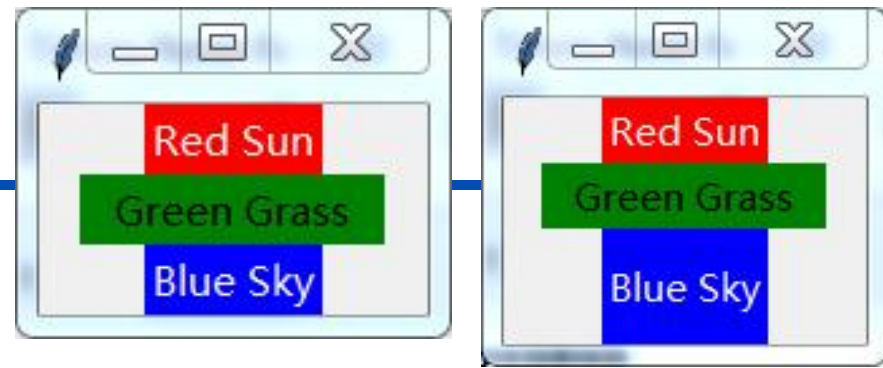


- Pack 可以在四个方面控制控件边距: 内边距, 外边距, 水平边距, 垂直边距
- **ipadx** - 设置水平方向的内边距

```
import tkinter as tk
root = tk.Tk()
w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack()
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack(ipadx=10)
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack()
root.mainloop()
```

## 7.1.2 几何布局管理器

### 1. pack几何布局管理器



- Pack 可以在四个方面控制控件边距: 内边距, 外边距, 水平边距, 垂直边距
- **ipady** - 设置竖直方向的内边距

```
import tkinter as tk
root = tk.Tk()
w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack()
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack(ipadx=10)
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(ipady=10)
root.mainloop()
```



# 7.1.2 几何布局管理器

## 1. pack几何布局管理器

### ➤ side 顺次放置控件

```
import tkinter as tk
root = tk.Tk()
w = tk.Label(root, text="red", bg="red", fg="white")
w.pack(padx=5, pady=10, side='left')
w = tk.Label(root, text="green", bg="green", fg="black")
w.pack(padx=5, pady=20, side='left')
w = tk.Label(root, text="blue", bg="blue", fg="white")
w.pack(padx=5, pady=20, side='left')
root.mainloop()
```



- 如果把上述 side 属性的值都改为 RIGHT, 那么上面 Label 控件的排列顺序就反过来了

# 7.1.2 几何布局管理器

---

## 2. grid几何布局管理器

- Grid 在很多场景下是最好用的布局方式。相比而言，Pack 布局在控制细节方面有些力不从心。
- Grid 把控件位置作为一个二维表结构来维护，即按照行列的方式排列控件。控件位置由其所在的行号和列号决定。
  - ▣ 行号相同而列号不同的几个控件会被彼此左右排列；
  - ▣ 列号相同而行号不同的几个控件会被彼此上下排列。
- 使用 Grid 布局的过程就是为各个控件指定行号和列号的过程。不需要为每个格子指定大小，Grid 布局会自动设置一个合适的大小。
- 每一列中，列宽由这列最宽的单元格确定；子组件可以跨越多行/列

## 7.1.2 几何布局管理器

### 2. grid几何布局管理器——计算器实例

```
import tkinter as tk
```

```
root = tk.Tk()
```

```
root.geometry('200x200+280+280')
```

```
root.title('计算器示例')
```

```
color = ('green','yellow') #定义背景色
```

```
for Bi in range(1,10):      #Grid 网格布局
```

```
    Li = tk.Button(root, text=str(Bi), width=5, bg=color[Bi%2])
```

```
    Li.grid(row=(Bi-1)//3, column=(Bi-1)%3)
```

```
L0 = tk.Button(root, text='0')
```

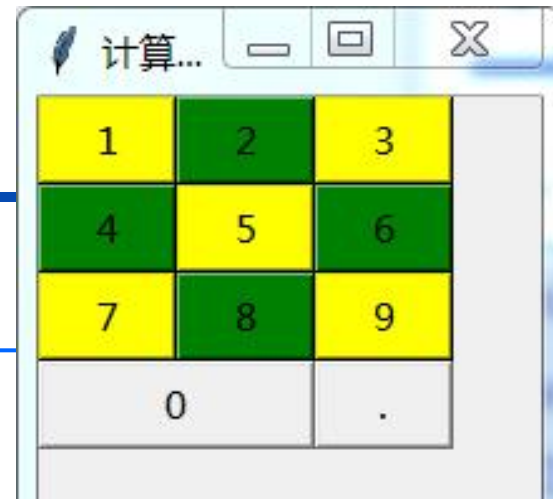
```
L0.grid(row = 3, column = 0,columnspan=2,sticky=E+W )    #跨2列，左右贴紧
```

```
Lp = tk.Button(root, text='.')
```

```
Lp.grid(row = 3, column = 2,sticky=E+W )                #左右贴紧
```

```
root.mainloop()
```

200x200代表了初始化时主窗口的大小，  
280, 280代表了初始化时窗口所在的位置



# 7.1.2 几何布局管理器

## 2. grid几何布局管理器

- grid几何布局函数的语法：子组件.grid (option)
- grid方法的参数选项如下表：

选项	描述	取值范围
sticky	组件紧贴所在单元格的某一边角，对应于东南西北以及四个角	n, s, e, w, nw, sw, se, ne, center (默认)
row	单元格行号	整数
column	单元格列号	整数
rowspan	行跨度	整数
columnspan	列跨度	整数
ipadx, ipady	组件内部在x/y方向上填充的空间大小	单位为c(厘米), m(毫米), i(英寸), p(打印机的点)
padx, pady	组件外部在x/y方向上填充的空间大小	单位为c(厘米), m(毫米), i(英寸), p(打印机的点)

## 7.1.2 几何布局管理器

### 3. place几何布局管理器

- place几何布局管理允许指定组件的大小与位置。place的优点是可以精确控制组件的位置，不足之处是改变窗口大小时，子组件不能随之灵活改变大小。
- place几何布局函数的语法：**子组件.place (option)**
- place方法的参数选项如下表：

选项	描述	取值范围
x, y	将组件放到指定位置的绝对坐标	从0开始的整数
relx, rely	将组件放到指定位置的相对坐标	取值范围0 ~ 1.0
height, width	高度和宽度，单位为像素	
sticky	组件紧贴所在单元格的某一边角，对应于东南西北以及四个角	n, s, e, w, nw, sw, se, ne, center (默认)

## 7.1.2 几何布局管理器

### 3. place几何布局管理器

➤ 实例:

```
import tkinter as tk
root = tk.Tk()
root.title("登录")
root['width']=250;root['height']=100      #root.geometry('250x100')
tk.Label(root,text = '用户名',width=6).place(x=1,y=5)      #绝对坐标 (1, 5)
tk.Entry(root,width=20).place(x=60,y=5)      #绝对坐标 (60, 5)
tk.Label(root,text = '密码',width=6).place(x=1,y=35)      #绝对坐标 (1, 35)
tk.Entry(root,width=20, show='*').place(x=60,y=35)      #绝对坐标 (60, 35)
tk.Button(root,text = '登录',width=8).place(x=60,y=65)      #绝对坐标 (60, 65)
tk.Button(root,text = '取消',width=8).place(x=150,y=65)      #绝对坐标 (150, 65)
root.mainloop()
```



# 目录

---

## 第7章 Tkinter图形界面设计

1	Python图形开发库
2	常用Tkinter组件的使用
3	图形绘制
4	Tkinter字体
5	Python事件处理
6	图形界面程序设计的应用

## 7.2 常用Tkinter 组件的使用

### 7.2.1 Tkinter 组件

- Tkinter提供各种组件（控件），如按钮、标签和文本框，供GUI应用程序使用。目前有15种常用的Tkinter组件，如下表所示：

控件名称	描述
Button	按钮控件，在程序中显示按钮
Canvas	画布控件，显示图形元素，如线条、文本及多边形等
Checkbutton	多选框控件，用于在程序中提供多项选择框
Entry	输入控件，定义一个简单的文字输入字段
Frame	框架控件，定义一个矩形区域，以作为其他控件的容器
Label	标签控件，定义一个文字或图片标签
Listbox	列表框控件，定义一个下拉方块
Menu	菜单控件，定义一个菜单栏、下拉菜单和弹出菜单
Menubutton	菜单按钮控件，用于显示菜单项



# 7.2 常用Tkinter 组件的使用

## 7.2.1 Tkinter 组件（续）

控件名称	描述
Message	消息控件，用来显示多行文本，与Label类似
Radiobutton	单选按钮控件，定义一个单选按钮，显示按钮状态
Scale	范围控件，定义一个滑动条，以帮助用户设置数值
Scrollbar	滚动条控件，当内容超过可视化区域时使用，如列表框
Text	文本控件，定义一个文本框，用于显示多行文本
Toplevel	此控件与Frame控件类似，可以作为其他控件的容器。但是此控件有自己的最上层窗口，可以提供窗口管理接口
Spinbox	输入控件；与Entry类似，但是可以输入指定范围值
PanedWindow	一个窗口布局管理的插件，可以包含一个或者多个子控件
LabelFrame	一个简单的容器控件，常用于复杂的窗口布局
tkMessageBox	用于显示你应用程序的消息框

## 7.2 常用Tkinter 组件的使用

### 7.2.1 Tkinter 组件

- 通过组件类的构造函数可以创建其对象实例，如

```
import tkinter as tk
root = tk.Tk()
button1 = tk.Button(root, text = "确定").pack(side = tk.RIGHT)
button2 = tk.Button(root, text = "取消").pack(side = tk.RIGHT)
root.mainloop()
```

## 7.2 常用Tkinter 组件的使用

### 7.2.2 标准属性

- 所有组件（控件）会有一些**共同的属性**，如大小，字体和颜色等等。这些属性也叫组件的标准属性，常用的标准属性如表所示。

属性	描述
dimension	控件大小
color	控件颜色
font	控件字体
anchor	锚点（停靠位置），对应于东南西北及四个角
relief	控件样式
bitmap	位图，内置位图有：error, gray75, gray50, gray25, gray12, hourglass, info, questhead, question, 和 warning.自定义位图是.xbm格式文件
cursor	光标
text	显示文本内容
state	设置组件状态；正常(normal)、激活(active)、禁用(disable)

## 7.2.3 Label标签组件

Label组件用于在窗口中显示文本或位图

```
import tkinter as tk
win = tk.Tk();                #创建窗口对象
win.title("我的窗口")        #设置窗口标题
lab1 = tk.Label(win, text = '你好', anchor= 'nw')  #创建文字是你好的Label组件
lab1.pack()                  #显示Label组件
lab2 = tk.Label(win, bitmap = 'question') #创建显示疑问图标Label组件
lab2.pack()                  #显示Label组件(内置的位图)
bm = tk.PhotoImage(file = r'J:\2021-2022-1 Python程序开发\第7章 图形界面设计\test.png') #显示自选的图片
lab3 = tk.Label(win, image = bm)
lab3.pack()                  #显示Label组件
win.mainloop()
```



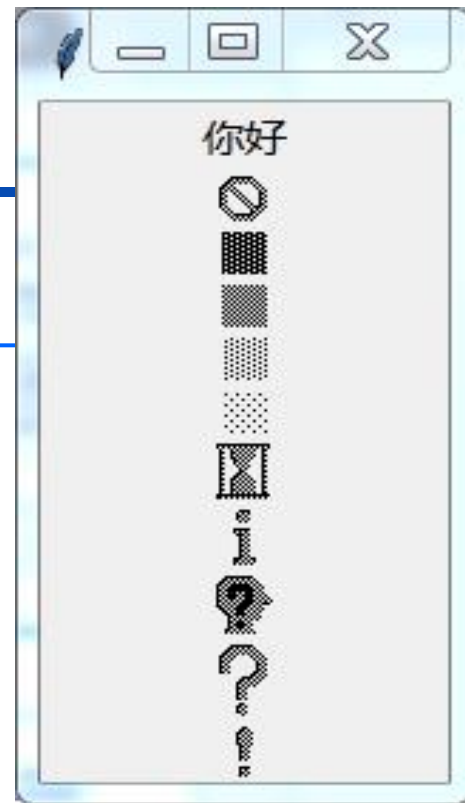
## 7.2.3 Label标签组件

Label组件用于在窗口中显示文本或位图

```
import tkinter as tk
win = tk.Tk();                #创建窗口对象
win.title("显示所有内置位图") #设置窗口标题
lab1 = tk.Label(win, text = '你好', anchor = 'nw')
lab1.pack()                   #显示Label组件
```


#显示内置的位图

```
bitmaps = ['error', 'gray75', 'gray50', 'gray25', 'gray12', 'hourglass', 'info',
            'questhead', 'question', 'warning']
for bit_map in bitmaps:
    lab2 = tk.Label(win, bitmap = bit_map).pack() #创建显示内置图标
win.mainloop()
```



# 7.2.3 Label标签组件

## Label组件常用属性

属性	描述									
width	宽度									
height	高度									
compound	指定文本与图像如何在Label上显示，缺省为None。当指定image/bitmap 时，文本(text)将被覆盖，只显示图像。可以使用的值如下: left, right, top, bottom, center(文字覆盖在图像上)									
wraplength	指定多少单位后开始换行，用于多行显示文本									
justif	指定多行的对齐方式，可以使用的值为left(左对齐)或right(右对齐)									
anchor	指定文本(text)或图像(bitmap/image)在Label 中的显示位置。对应于东南西北以及四个角，可用值如图所示: <div><table><tr><td>nw</td><td>n</td><td>ne</td></tr><tr><td>w</td><td>center</td><td>e</td></tr><tr><td>sw</td><td>s</td><td>se</td></tr></table></div>	nw	n	ne	w	center	e	sw	s	se
nw	n	ne								
w	center	e								
sw	s	se								
image和bm	显示自定义图片，如png, gif									
bitmap	显示内置的位图									

## 7.2.4 Button按钮组件

---

- Button组件（控件）是一个标准的Tkinter部件，用于实现各种按钮
  - 按钮可以包含文本或图像；
  - 可以通过`command`属性将Python函数或方法关联到按钮上，当Tkinter的按钮被按下时，会自动调用该函数或方法。
  - 该按钮只能显示单一字体的文本，但文本可以跨越多行。
  - 此外，字符可以有下划线，例如标记的键盘快捷键。

## 7.2.4 Button按钮组件

### Tkinter Button按钮属性和方法

```
import tkinter as tk
```

```
root = tk.Tk()
```

```
root.title("Button Test")
```

```
def callback():                #定义一个按钮触发函数
```

```
    tk.messagebox.showinfo("Python command", "人生苦短、我用Python")
```

```
# 创建5个Button按钮、并设置width, height, relief, bg, bd, fg, state, bitmap, command, anchor
```

```
tk.Button(root, text="外观装饰边界附近的标签", width=19, relief=GROOVE,  
          bg="red").pack()
```

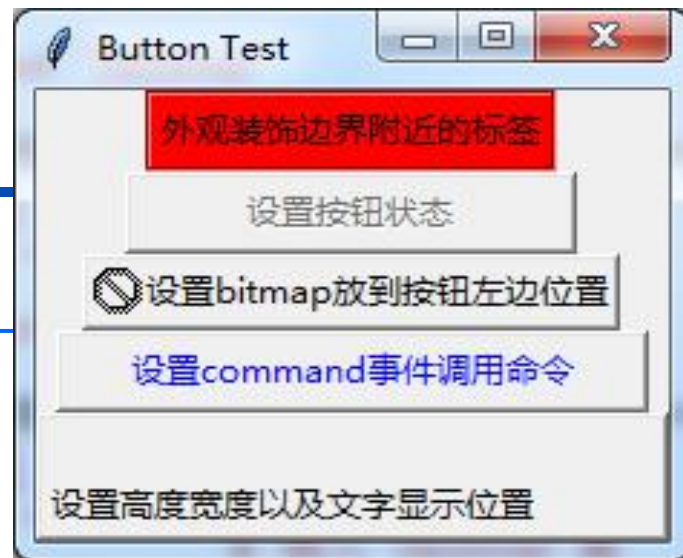
```
tk.Button(root, text="设置按钮状态", width=21, state=DISABLED).pack()
```

```
tk.Button(root, text="设置bitmap放到按钮左边位置", compound="left",  
          bitmap="error").pack()
```

```
tk.Button(root, text="设置command事件调用命令", fg="blue", bd=10, width=28,  
          command=callback).pack()
```

```
tk.Button(root, text="设置高度宽度以及文字显示位置", anchor='sw', width=30,  
          height=2).pack()
```

```
root.mainloop()
```





# 7.2.4 Button按钮组件

## Tkinter Button按钮组件常用属性

属性	描述
text	显示文本内容
command	指定Button的事件处理函数
compound	指定文本与图像的位置关系
bitmap	指定位图
focus_set	设置当前组件得到的焦点
master	代表了父窗口
bg	设置背景颜色
fg	设置前景颜色
font	设置字体大小
height	设置显示高度，如果未设置此项，其大小以适应内容标签
relief	指定外观装饰边界附近的标签，默认是平的，可以设置的参数： flat、groove、raised、ridge、solid、sunken

## 7.2.4 Button按钮组件

### Tkinter Button按钮组件常用属性-续

属性	描述
width	设置显示宽度，如果未设置此项,其大小以适应内容标签
wrplength	将此选项设置为所需的数址限制每行的字符数,默认为0
state	设置组件状态:正常(normal)、激活(active)、禁用(disabled)
anchor	设置Button文本在控件上的显示位置，可用值:
bd	设置Button的边框大小; bd(bordwidth)缺省为1或2个像素
textvariable	设置Button可变的文本内容对应的变量

### Tkinter Button按钮方法

属性	描述
flash()	按钮在active color and normal color颜色之间闪烁几次，disabled状态无效
invoke()	调用按钮的command指定的回调函数

## 7.2.5 单行文本框Entry和多行文本框text

实例：输入摄氏度，转换成华氏度，并显示出转换结果。

```
import tkinter as tk

def btnHelloClicked():          #事件函数
    cd = float(entryCd.get())    #获取文本框内输入的内容转换成浮点数
    labelHello.config(text = "%.2f °C = %.2f °F" % (cd, cd*1.8+32))

root = tk.Tk()
root.title("Entry Test")
labelHello=tk.Label(root, text="转换°C to °F", height=5, width=20, fg="blue")
labelHello.pack()
entryCd = tk.Entry(root).pack() #创建Entry组件, 并显示Entry组件
btnCal = tk.Button(root, text="转换温度", command=btnHelloClicked) #按钮
btnCal.pack()
root.mainloop()
```

## 7.2.5 单行文本框Entry和多行文本框text

实例：输入摄氏度，转换成华氏度，并显示出转换结果。

```
import tkinter as tk
```

```
def btnHelloClicked():
```

```
    cd = float(entryCd.get())
```

```
    labelHello
```

```
root = tk.
```

```
root.title()
```

```
labelHello
```

```
labelHello
```

```
entryCd =
```

```
btnCal = tk.Button(root, text="转换温度", command=btnHelloClicked) #按钮
```

```
btnCal.pack()
```

```
root.mainloop()
```

#事件函数

#获取文本框中输入的内容并转换成浮点数

°C = %

转换°C

#创建E

=20, fg="blue")

组件



## 7.2.5 单行文本框Entry和多行文本框text

实例：输入摄氏度，转换成华氏度，并显示出转换结果。

```
import tkinter as tk

def btnHelloClicked():          #事件函数
    cd = float(entryCd.get())    #获取文本框内输入的内容转换成浮点数
    labelHello.config(text = "%.2f °C = %.2f °F" %(cd, cd*1.8+32))

#组件属性除了可以用.config修改，还可以用以下方法修改：
labelHello['text'] = "%.2f °C = %.2f °F" %(cd, cd*1.8+32)
```

## 7.2.5 单行文本框Entry和多行文本框text

### 1. 创建和显示Entry对象

- 创建Entry对象的基本方法: `Entry对象 = Entry(Windows窗口对象)`
- 显示Entry对象的方法: `Entry对象.pack()/.grid()/.place()`

### 2. 获取Entry组件内容

- 获取文本框内输入的内容: `Entry对象.get()`

### 3. 设置Entry组件的内容

- 可以通过StringVar()对象来设置或者获取Entry组件内容, 例如:

```
>>> s = StringVar()
```

```
>>> s.set( "使用StringVar设置Entry内容" )
```

```
>>> entryCd = Entry(root, text = s)
```

```
>>> print(s.get())
```

## 7.2.5 单行文本框Entry和多行文本框text

### 4. Entry常用属性

属性	描述
show	如果设置为字符*, 则输入文本框内显示为*, 用于密码输入
insertbackground	插入光标的颜色, 默认为黑色“black”
selectbackground 和 selectforeground:	选中文本的背景色和前景色
width	组件的宽度 (所占字节个数)
fg	字体前景颜色
bg	背景颜色
state	设置组件状态, 默认是normal, 可设置为disable、readonly

- Python也提供了**多行文本框Text**, 用于输入多行内容和显示文本, 使用方法类似Entry, 可查阅Tkinter手册。

## 7.2.6 列表框组件Listbox

实例:

```
import tkinter as tk
root = tk.Tk()
root.title("使用Listbox组件的例子")    #设置窗口标题
m = tk.StringVar()                      #创建一个字符串变量对象

def callbutton1():
    tb.config(text = m.get()) #获取列表内容，并在label上显示

def callbutton2():
    listtext = ''
    for i in lb.curselection():        #获取列表选项内容(多选)
        listtext = listtext+lb.get(i)+' '
    tb.config(text = listtext)         #在button上显示选中项
```





## 7.2.6 列表框组件Listbox

实例：

```
lb = tk.Listbox(root, listvariable=m, selectmode=tk.MULTIPLE)
for item in ['北京','天津','上海']:
```

```
    lb.insert(tk.END, item)#在下拉列表中插入列表项
```

```
lb.pack()
```

```
tb = tk.Label(root, text= "显示Button状态", bg = "yellow")
```

```
tb.pack()
```

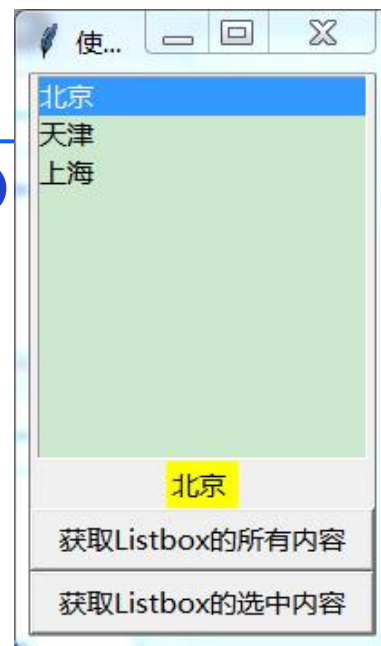
```
b1 = tk.Button (root, text = '获取Listbox的所有内容', command=callbutton1)
```

```
b1.pack()                #创建Button组件并显示Button组件
```

```
b2 = tk.Button (root, text = '获取Listbox的选中内容', command=callbutton2)
```

```
b2.pack()                #创建Button组件并显示Button组件
```

```
root.mainloop()
```



## 7.2.6 列表框组件Listbox

### 1. 创建和显示ListBox对象

- 创建ListBox对象的方法: `ListBox对象 = ListBox(Windows窗口对象)`
- 显示ListBox对象的方法: `ListBox对象.pack()/.grid()/.place()`

### 2. 插入文本项

- 使用insert()方法向列表框组件中插入文本项:

`Listbox对象.insert(index, item)`

其中

- ❑ index是插入文本项的位置，如果在尾部插入文本项，则可以使用END；如果在当前选中处插入文本项，则可以使用ACTIVE
- ❑ item 是要插入的文本项。

## 7.2.6 列表框组件Listbox

### 3. 返回选中项索引

- 返回当前选中项目的索引，结果为元组（可多选）：

**Listbox对象. curselection()**

### 4. 删除文本项

- 使用delete()方法从列表框组件中删除文本项：

**Listbox对象. delete(first, last)**

如果删除不连续的多个文本项，如何操作？

- 删除指定范围(first, last)的项目，不指定last时，删除1个项目。

### 5. 获取项目内容

- 返回指定范围(first, last)的项目，不指定last时，仅返回1个项目：

**Listbox对象. get(first, last)**

## 7.2.6 列表框组件Listbox

### 6. 获取项目个数

Listbox对象.size()

### 7. 获取Listbox内容

- 需要使用listvariable属性为Listbox对象指定一个对应的变量，例如：

```
m = StringVar()
```

```
listb = tk.Listbox(root, listvariable = m, selectmode=tk.MULTIPLE)
```

- 指定后就可以使用m.get()方法，获取Listbox对象中的内容了
- 注意：如果允许用户选择多个项目，则将Listbox对象的selectmode属性设置为MULTIPLE表示多选，而设置为SINGLE表示单选。



## 7.2.6 列表框组件Listbox

**实例：** 从一个列表框选择内容添加到另一个列表框

```
import tkinter as tk
root = tk.Tk()
root.title("从一个列表框选择内容添加到另一个列表框")    #设置窗口标题
def callbutton1():
    for i in listb1.curselection():                          #遍历选中项
        listb2.insert(0, listb1.get(i))                     #添加到右侧列表框
def callbutton2():
    for i in listb2.curselection():                          #遍历选中项
        listb2.delete(i)                                    #从右侧列表框中删除
```



## 7.2.6 列表框组件Listbox

**实例：** 从一个列表框选择内容添加到另一个列表框

```
li = ['C', 'python', 'php', 'html', 'SQL', 'java']
```

```
listb1 = tk.Listbox(root) #创建两个列表框组件
```

```
listb2 = tk.Listbox(root)
```

```
for item in li:                #左侧列表框组件插入数据
```

```
    listb1.insert(0, item)
```

```
listb1.grid(row=0, column=0, rowspan=2) # 将列表框组件放置到窗口对象中
```

```
listb2.grid(row=0, column=2, rowspan=2)
```

```
b1 = tk.Button (root,text = '添加>>', command=callbutton1, width=20)
```

```
b2 = tk.Button (root,text = '删除<<', command=callbutton2, width=20)
```

```
b1.grid(row=0, column=1)        #显示Button组件
```

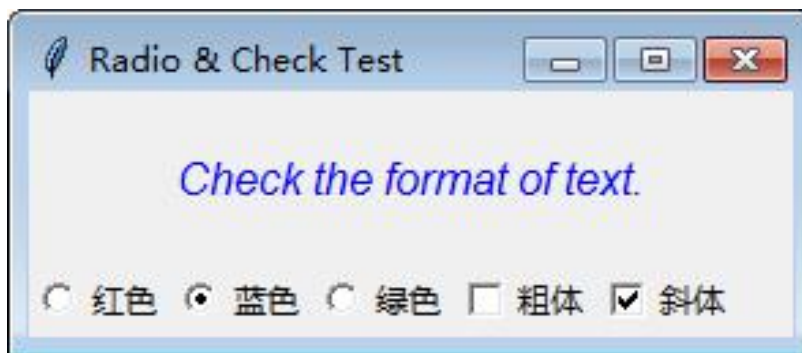
```
b2.grid(row=1, column=1)        #显示Button组件
```

```
root.mainloop()                # 进入消息循环
```



## 7.2.7 单选按钮和复选框

- 单选按钮 (Radiobutton) 和复选框 (Checkbutton) 分别用于实现选项的单选和复选功能。
  - ❑ Radiobutton用于同一组单选按钮中选择一个单选按钮(不能同时选定多个)
  - ❑ Checkbutton用于选择一项或多项
  - ❑ Radiobutton、Checkbutton可以显示文本，也可以显示图像。



## 7.2.7 单选按钮和复选框

### 实例：

```
import tkinter as tk
root = tk.Tk()
root.title("使用单选按钮的例子")      #设置窗口标题
r = tk.StringVar()                      #创建一个字符串对象
r.set('1')                              #设置初始值为'1'， 初始选中'中国'
radio=tk.Radiobutton(root,variable=r,value='1',text='中国')
radio.pack()
radio=tk.Radiobutton(root,variable=r,value='2',text='美国')
radio.pack()
radio=tk.Radiobutton(root,variable=r,value='3',text='日本')
radio.pack()
radio=tk.Radiobutton(root,variable=r,value='4',text='加拿大')
radio.pack()
radio=tk.Radiobutton(root,variable=r,value='5',text='韩国')
radio.pack()
root.mainloop()
print (r.get())                        #获取当前被选中单选按钮变量值
```





## 7.2.7 单选按钮和复选框

### 1. 创建和显示Radiobutton对象

- 创建Radiobutton对象的方法：

**Radiobutton对象 = Radiobutton(窗口对象, text = 单选按钮显示的文本)**

- 显示Radiobutton对象的方法：

**Radiobutton对象.pack()/.grid()/.place()**

- 可以使用variable属性，为Radiobutton组件指定一个对应的变量。

**radio=tk.Radiobutton(root, variable=r, value='1', text='中国')**

- ❑ 如果将多个Radiobutton组件绑定到同一个变量，则这些Radiobutton组件属于一个分组。
- ❑ 分组后需要使用value设置每个Radiobutton组件的值，以表示该项目是否被选中。

## 7.2.7 单选按钮和复选框

### 2. Radiobutton组件常用属性

属性	描述
<b>variable</b>	单选按钮索引变量，通过变量的值确定哪个单选按钮被选中。一组单选按钮使用同一个索引变量。
<b>value</b>	单选按钮选中时变量的值
<b>command</b>	单选按钮选中时执行的命令（函数）

### 3. Radiobutton组件的方法

属性	描述
<b>deselect()</b>	取消选择
<b>select()</b>	选择
<b>invoke()</b>	调用单选按钮command指定的回调函数。

## 7.2.7 单选按钮和复选框

---

### 4. 创建和显示Checkbutton对象

➤ 创建Checkbutton对象的方法：

□ **Checkbutton对象 = Checkbutton(窗口对象, text = 复选框按钮显示的文本, command = 单击复选框按钮所调用的回调函数)**

➤ 显示Checkbutton对象的方法：

□ **Checkbutton对象.pack()/.grid()/.place()**

## 7.2.7 单选按钮和复选框

### 5. Checkbutton组件常用属性

属性	描述
<b>variable</b>	复选框索引变量，通过变量的值确定哪些复选框被选中。每个复选框使用不同的变量，使复选框之间相互独立。
<b>onvalue</b>	复选框选中(有效)时变量的值
<b>offvalue</b>	复选框未选中(无效)时变量的值
<b>command</b>	复选框选中时执行的命令(函数)

## 7.2.7 单选按钮和复选框

### 6. 获取Checkbutton状态

- 为了获取Checkbutton组件是否被选中，需要使用variable属性为Checkbutton组件指定一个对应变量，例如：

```
c = tk.IntVar()
```

```
c.set(2)          #设置复选框的状态，1选中，2没选中
```

```
check=tk.Checkbutton(root, text='喜欢',variable=c, onvalue=1, offvalue=2)
```

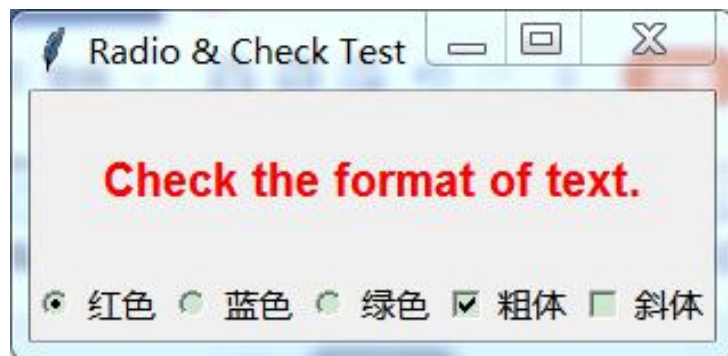
```
check.pack()
```

- 指定变量c后，可以使用 **c.get()** 获取复选框的状态值。
- 也可以使用 **c.set()** 设置复选框的状态。

## 7.2.7 单选按钮和复选框

**实例：**通过单选按钮、复选框设置标签文字的样式

```
import tkinter as tk
def colorChecked():
    label_1.config(fg = color.get())
def typeChecked():
    textType = typeBold.get() + typeItalic.get()
    if textType == 1:
        label_1.config(font = ("Arial", 12, "bold"))
    elif textType == 2:
        label_1.config(font = ("Arial", 12, "italic"))
    elif textType == 3:
        label_1.config(font = ("Arial", 12, "bold italic"))
    else :
        label_1.config(font = ("Arial", 12))
```



## 7.2.7 单选按钮和复选框

**实例：通过单选按钮、复选框设置标签文字的样式**

```
root = tk.Tk()
root.title("Radio & Check Test")
label_1 = tk.Label(root, text = "Check the format of text.", height = 3,
                    font=("Arial", 12))
label_1.config(fg = "blue")      #初始颜色蓝色
label_1.pack()

color = tk.StringVar()          #三个颜色Radiobutton定义了同样的变量color
color.set("blue")

typeBold = tk.IntVar()          #定义了typeBlod变量表示文字是否为粗体
typeItalic = tk.IntVar()        #定义了typeItalic变量表示文字是否为斜体
```

## 7.2.7 单选按钮和复选框

### 实例：通过单选按钮、复选框设置标签文字的样式

```
tk.Radiobutton(root, text = "红色", variable = color, value = "red",  
command = colorChecked).pack(side = tk.LEFT)
```

```
tk.Radiobutton(root, text = "蓝色", variable = color, value = "blue",  
command = colorChecked).pack(side = tk.LEFT)
```

```
tk.Radiobutton(root, text = "绿色", variable = color, value = "green",  
command = colorChecked).pack(side = tk.LEFT)
```

```
tk.Checkbutton(root, text = "粗体", variable = typeBold, onvalue = 1,  
offvalue = 0, command = typeChecked).pack(side = tk.LEFT)
```

```
tk.Checkbutton(root, text = "斜体", variable = typeItalic, onvalue = 2,  
offvalue = 0, command = typeChecked).pack(side = tk.LEFT)
```

```
root.mainloop()
```



## 7.2.8 菜单组件Menu

---

- 图形用户界面（GUI）应用程序通常提供菜单，菜单包含各种按照主题分组的基本命令。
- 图形用户界面应用程序包括2种类型的菜单。
  - **主菜单**：提供窗体的菜单系统。通过单击可下拉出子菜单，选择命令可执行相关的操作。常用的主菜单通常包括：文件、编辑、视图、帮助等。
  - **上下文菜单**(也称为快捷菜单)：通过鼠标右击某对象而弹出的菜单，一般为与该对象相关的常用菜单命令。例如：剪切、复制、粘贴等。

## 7.2.8 菜单组件Menu

### 1. 创建和显示Menu对象

➤ 创建Menu对象的方法:

□ **Menu对象** = **Menu(Windows窗口对象)**

➤ 显示Menu对象的方法:

□ **Windows窗口对象['menu'] = Menu对象**

□ **Windows窗口对象.mainloop()**



```
import tkinter as tk
root = tk.Tk()
def hello():          #菜单项事件函数，可以每个菜单项单独写
    print("你单击主菜单")
topmenu = tk.Menu(root)
for item in ['文件', '编辑', '视图']: #添加菜单项
    topmenu.add_command(label=item, command=hello)
root['menu'] = topmenu          #附加主菜单到窗口
root.mainloop()
```

## 7.2.8 菜单组件Menu

### 2. 添加下拉菜单

- 创建主菜单之后，默认情况不包含下拉菜单，若想加下拉菜单，可以将另一个Menu组件作为它的下拉菜单，方法如下：
  - **Menu对象1.add\_cascade(label = 菜单文本, menu = Menu对象2)**
  - 上述语法格式，就将Menu对象2设置为Menu对象1的下拉菜单。
- 在创建Menu对象2的时候，也要指明它是Menu对象1的子菜单，如下：
  - **Menu对象2 = Menu(Menu对象1)**

## 7.2.8 菜单组件Menu

### 2. 添加下拉菜单——实例

- 使用`add_cascade()`方法给“文件”、“编辑”菜单添加下拉菜单



## 7.2.8 菜单组件Menu

### 2. 添加下拉菜单——实例

- 使用add\_cascade()方法给“文件”、“编辑”菜单添加下拉菜单

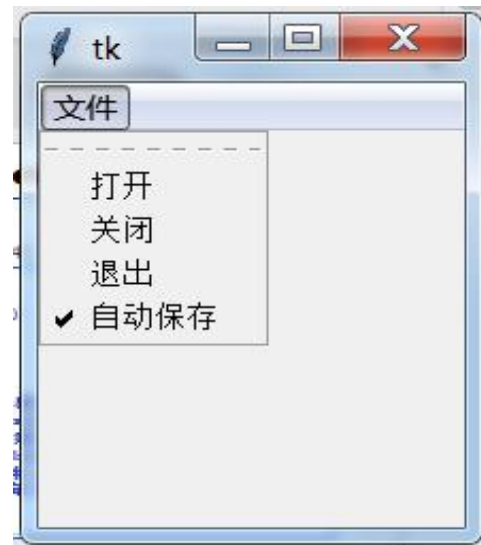


```
import tkinter as tk
def hello():
    print("I'm a child menu")
root = tk.Tk()
topmenu = tk.Menu(root)           #创建主菜单
filemenu = tk.Menu(topmenu)       #创建下拉菜单, 指明是topmenu的下拉菜单
editmenu = tk.Menu(topmenu)       #创建下拉菜单, 指明是topmenu的下拉菜单
for item in ['打开', '关闭', '退出']: #添加菜单项
    filemenu.add_command(label=item, command = hello)
for item in ['复制', '剪切', '粘贴']: #添加菜单项
    editmenu.add_command(label=item, command = hello)
topmenu.add_cascade(label='文件', menu = filemenu) #把filemenu作为文件下拉菜单
topmenu.add_cascade(label='编辑', menu = editmenu) #把editmenu作为编辑下拉菜单
root['menu'] = topmenu             #附加主菜单到窗口
root.mainloop()
```

## 7.2.8 菜单组件Menu

### 3. 在菜单中添加复选框 —— add\_checkbutton()

```
import tkinter as tk
def hello():
    print(v.get())
root = tk.Tk()
v = tk.StringVar()
topmenu = tk.Menu(root)
filemenu = tk.Menu(topmenu)
for item in ['打开','关闭','退出']:
    filemenu.add_command(label=item, command = hello)
topmenu.add_cascade(label='文件', menu = filemenu)
filemenu.add_checkbutton(label='自动保存',command = hello,variable = v)
root['menu'] = m
root.mainloop()
```



## 7.2.8 菜单组件Menu

### 4. 在菜单中的当前位置添加分隔符 —— add\_separator()

```
import tkinter as tk
def hello():
    print("I'm a child menu")
root = tk.Tk()

m = tk.Menu(root)
filemenu = tk.Menu(m)
filemenu.add_command(label='打开', command = hello)
filemenu.add_command(label='关闭', command = hello)
filemenu.add_separator()           #'关闭'和'退出'之间添加分隔符
filemenu.add_command(label='退出', command = hello)
m.add_cascade(label='文件', menu = filemenu)
root['menu'] = m
root.mainloop()
```



## 7.2.8 菜单组件Menu

### 5. 创建上下文菜单

- 上下文菜单（也称为快捷菜单）是通过鼠标右击某对象而弹出的菜单，一般是与该对象相关的常用菜单命令。例如：剪切、复制、粘贴等。
- 创建上下文菜单的一般步骤：
  - 创建菜单（与创建主菜单相同），例如：

```
import tkinter as tk
root = tk.Tk()
menubar = tk.Menu(root)
menubar.add_command(label = '剪切', command = hello1)
menubar.add_command(label = '复制', command = hello2)
menubar.add_command(label = '粘贴', command = hello3)
```



## 7.2.8 菜单组件Menu

### 5. 创建上下文菜单

- 上下文菜单（也称为快捷菜单）是通过鼠标右击某对象而弹出的菜单，一般是与该对象相关的常用菜单命令。例如：剪切、复制、粘贴等。
- 创建上下文菜单的一般步骤：
  - ❑ 绑定鼠标右击事件，并在事件处理函数中弹出菜单。例如：

```
def popup(event)                                #事件处理函数
    menubar.post(event.x_root, event.y_root)    #在鼠标右键位置显示菜单

root.bind('<Button -3>', popup)                  #绑定事件
```

## 7.2.8 菜单组件Menu

### 5. 创建上下文菜单——实例

```
import tkinter as tk

def popup(event):                                #右键事件处理函数
    menubar.post( event.x_root, event.y_root)    #在鼠标右键位置显示菜单

def hello1():                                    #菜单事件处理函数
    print("我是剪切命令")

def hello2():
    print("我是复制命令")

def hello3():
    print("我是粘贴命令")
```

## 7.2.8 菜单组件Menu

### 5. 创建上下文菜单——实例

```
root = tk.Tk()
root.geometry("300x150")
menubar = tk.Menu(root)
menubar.add_command(label='剪切', command = hello1)
menubar.add_command(label='复制', command = hello2)
menubar.add_command(label='粘贴', command = hello3)
s=tk.StringVar()                                #一个StringVar()对象
s.set("大家好, 这是测试上下文菜单")
entryCd = tk.Entry(root, textvariable=s)        #创建Entry组件界面
entryCd.pack()
root.bind('<Button-3>',popup)                    #绑定右键事件
root.mainloop()
```

## 7.2.9 对话框

---

- 对话框用于与用户交互和检索信息。
- Tkinter模块中有一些子模块，如：messagebox（消息盒）、filedialog（文件对话框）、colorchooser（颜色对话框）、simpleDialog（简单对话框），这些子模块包含了一些通用的预定义对话框；
- 用户也可以**通过继承TopLevel创建自定义对话框**。

## 7.2.9 对话框

### 1. 文件对话框

- Tkinter的子模块filedialog包含用于打开文件对话框的函数

**askopenfilename()**

- 文件对话框供用户选择某文件夹下的文件。语法格式如下：

**askopenfilename(title = '标题', filetypes = [('所有文件', '\*.\*), ('文本文件'), '.txt']])**

- 其中, filetypes: 文件过滤器, 可以筛选某种格式文件

- title: 设置打开文件对话框的标题

- 还有文件保存对话框函数**asksavesfilename()**, 语法格式如下:

**asksavesfilename(title = '标题', initialdir = 'd:\mywork', initialfile = 'hello.py')**

- 其中, initialdir: 默认保存路径即文件夹, 如 'd:\mywork'

- initialfile: 默认保存的文件名, 如 'hello.py'

## 7.2.9 对话框

### 1. 文件对话框——实例

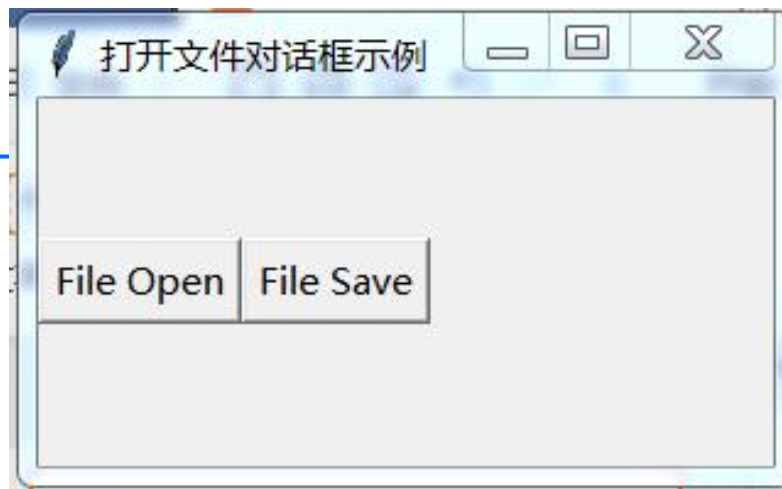
```
import tkinter as tk
from tkinter import filedialog
def openfile():                #按钮事件处理函数
                                #显示打开文件对话框，返回选中文件名以及路径
    r = tk.filedialog.askopenfilename(title='打开文件', filetypes=[('Python',
'*.*.py *.pyw'), ('All Files', '*.*')])
    print(r)
def savefile():                #按钮事件处理函数
                                #显示保存文件对话框
    r = tk.filedialog.asksaveasfilename(title='保存文件',
initialdir='d:\mywork', initialfile='hello.py')
    print(r)
```

## 7.2.9 对话框

### 1. 文件对话框——实例

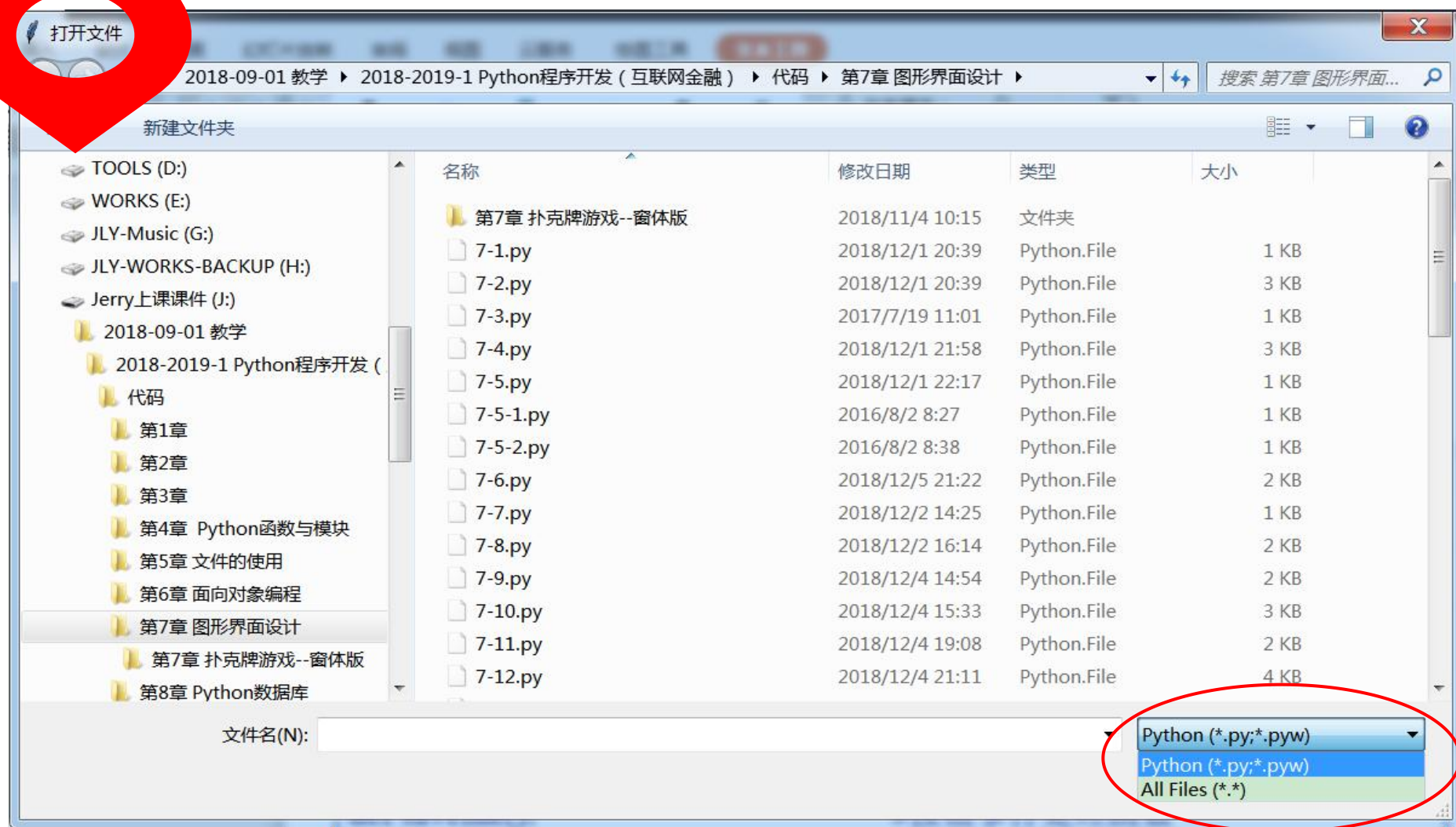
```
root = tk.Tk()
root.title('打开文件对话框示例')
root.geometry("300x150")
btn1 = tk.Button(root, text='File Open', command=openfile) #创建Button组件
btn2 = tk.Button(root, text='File Save', command=savefile) #创建Button组件
btn1.pack(side='left')
btn2.pack(side='left')
root.mainloop()
```

# title属性用来指定标题



# 7.2.9 对话框

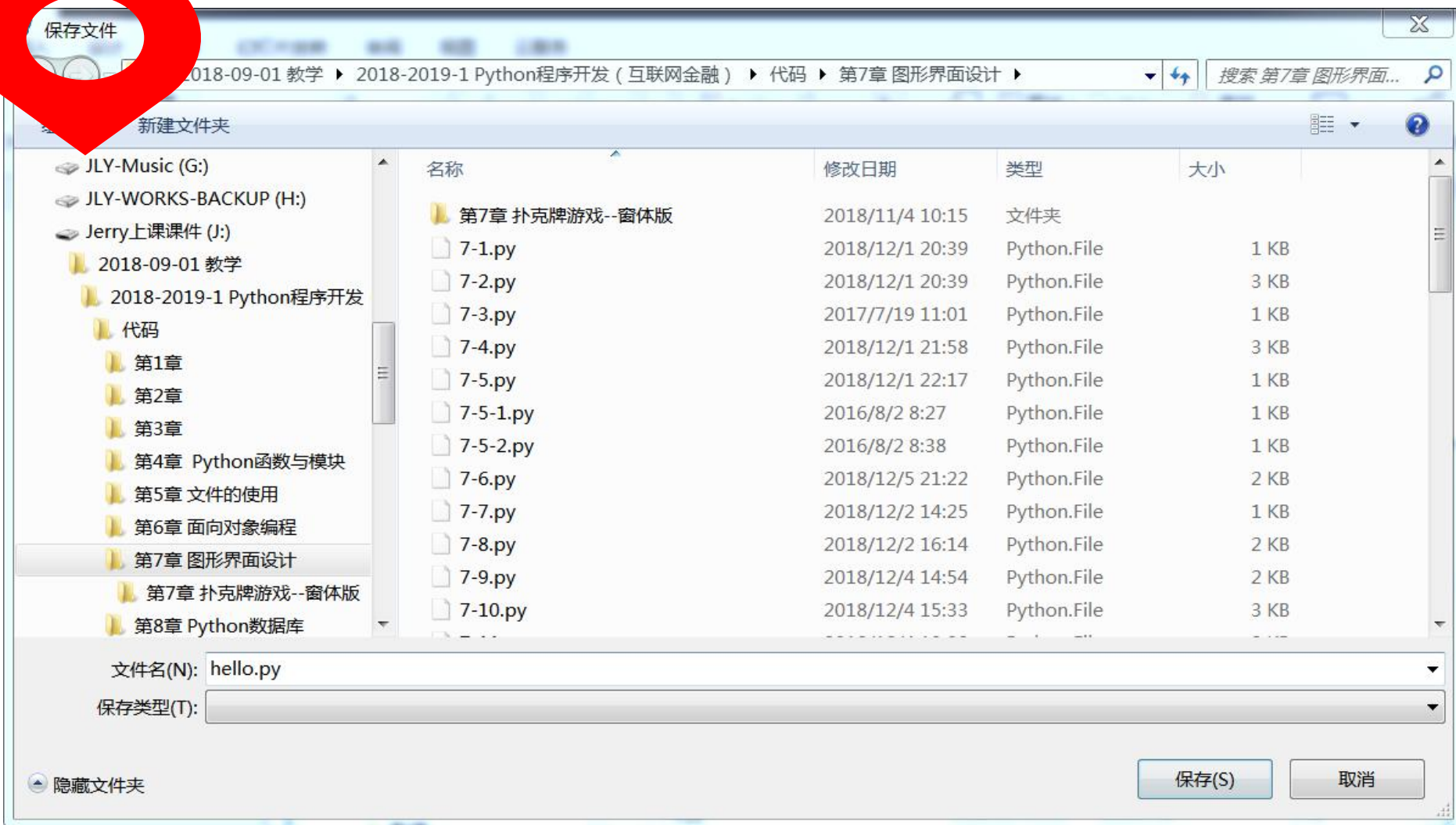
## 1. 文件对话框——实例





# 7.2.9 对话框

## 1 文件对话框——实例



# 7.2.9 对话框

## 2. 颜色对话框

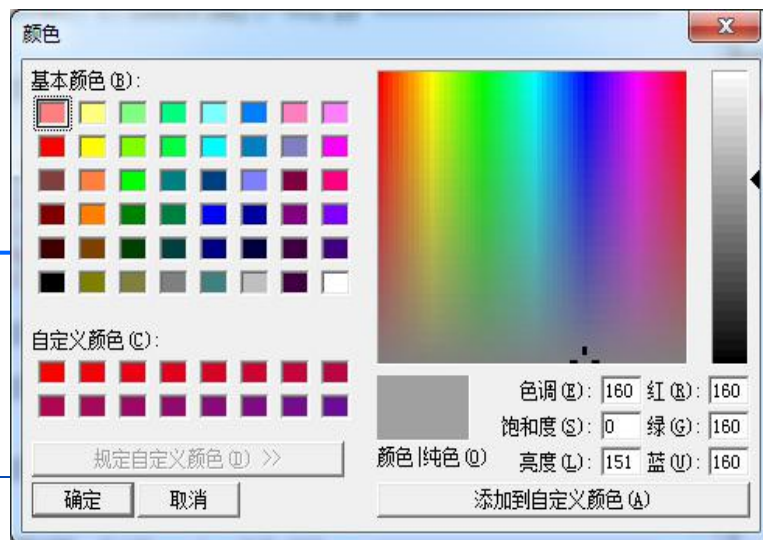
- Tkinter的子模块**colorchooser**包含用于打开颜色对话框的函数**askcolor()**
- 颜色对话框供用户选择某颜色。

```
import tkinter as tk
from tkinter import colorchooser
root = tk.Tk()
#调用askcolor返回选中颜色的(R,G,B)值和'#RRGGBB'的十六进制表示
color = colorchooser.askcolor()
print(color)
root.mainloop()
```

- 弹出颜色对话框，点击确定，关闭对话框，返回颜色值
- 如果点击取消，关闭对话框，返回 (None, None)

输出结果：

((205.80078125, 5.01953125, 50.1953125),  
'#cd0532')



## 7.2.9 对话框

### 3. 简单对话框

- Tkinter的子模块`simpdialog`包含用于打开输入对话框的函数
  - `askfloat(title, prompt, 选项)`: 打开输入对话框, 输入并返回浮点数
  - `askinteger(title, prompt, 选项)`: 打开输入对话框, 输入并返回整数
  - `askstring(title, prompt, 选项)`: 打开输入对话框, 输入并返回字符串
- 其中,
  - `title`为窗口标题
  - `prompt`为提示文本信息
  - 选项包括: `initialvalue` (初始值)、`minvalue` (最小值)、`maxvalue` (最大值)

## 7.2.9 对话框

### 3. 简单对话框

```
import tkinter as tk
from tkinter import simpledialog
def inputStr():
```

```
    r = simpledialog.askstring('Python Tkinter', 'Input String', initialvalue =
    'Python Tkinter')
```

```
    print(r)
```

```
def inputInt():
```

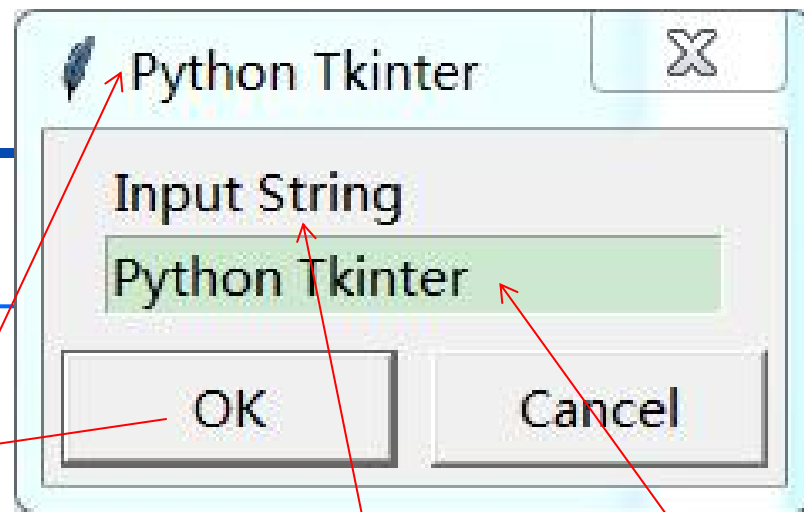
```
    r = simpledialog.askinteger('Python Tkinter', 'Input Integer')
```

```
    print(r)
```

```
def inputFloat():
```

```
    r = simpledialog.askfloat('Python Tkinter', 'Input Float')
```

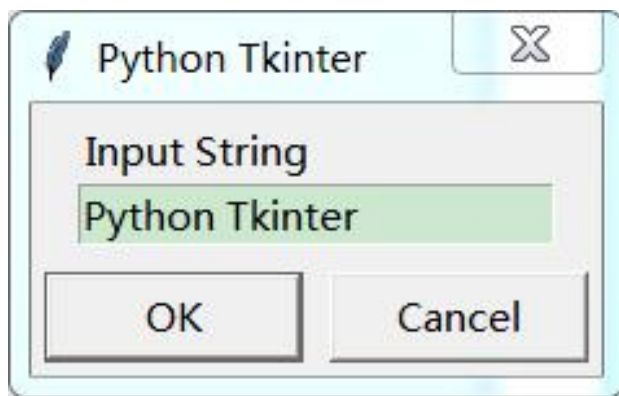
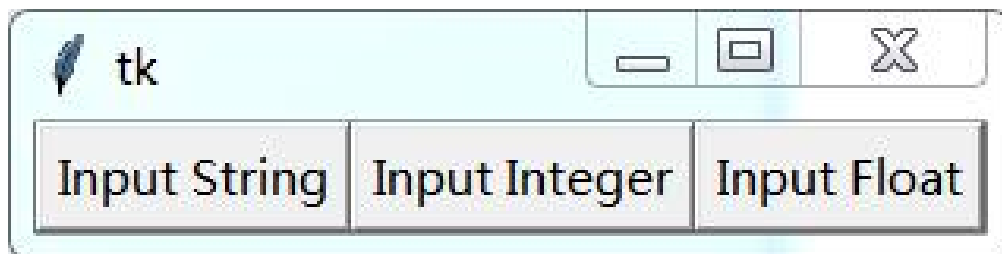
```
    print(r)
```



## 7.2.9 对话框

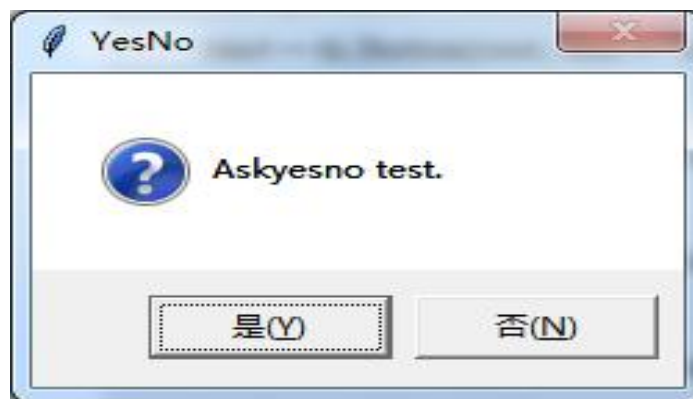
### 3. 简单对话框

```
root = tk.Tk()
btn1 = tk.Button(root, text='Input String', command=inputStr)
btn2 = tk.Button(root, text='Input Integer', command=inputInt)
btn3 = tk.Button(root, text='Input Float', command=inputFloat)
btn1.pack(side='left')
btn2.pack(side='left')
btn3.pack(side='left')
root.mainloop()
```



## 7.2.10 消息窗口（消息框）

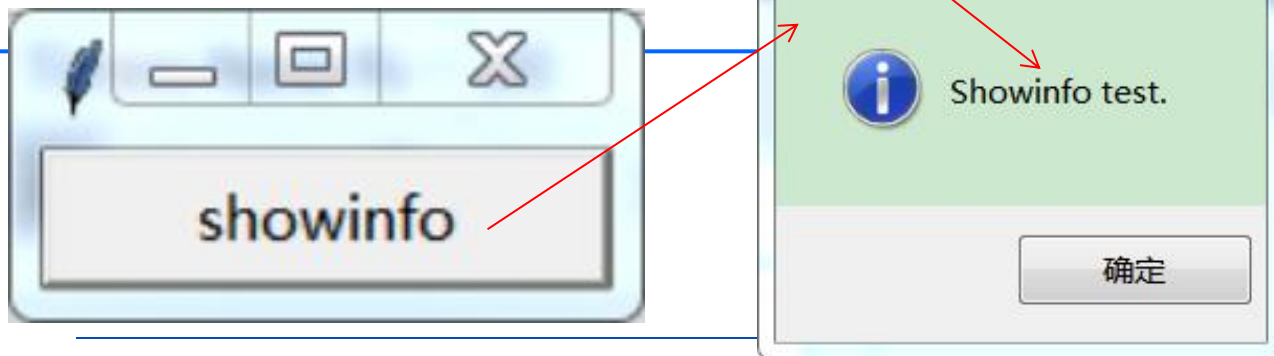
- 消息窗口（messagebox）用于弹出提示框向用户进行告警，或让用户选择下一步如何操作。
- 消息框包括很多类型，常用的有info、warning、error、yesno、okcancel等，包含不同的图标、按钮以及弹出提示音。



## 7.2.10 消息窗口（消息框）

### Info消息窗口

```
import tkinter as tk
from tkinter import messagebox as msgbox
def btn1_clicked():
    msgbox.showinfo("Info", "Showinfo test.")
root = tk.Tk()
root.title("MsgBox Test")
btn1 = tk.Button(root, text = "showinfo", command = btn1_clicked)
btn1.pack(fill = tk.X)
root.mainloop()
```



## 7.2.10 消息窗口（消息框）

### Warning消息窗口

```
import tkinter as tk
from tkinter import messagebox as msgbox
def btn2_clicked():
    msgbox.showwarning("Warning", "Showwarning test.")
root = tk.Tk()
root.title("MsgBox Test")
btn2 = tk.Button(root, text = "showwarning", command = btn2_clicked)
btn2.pack(fill = tk.X)
root.mainloop()
```

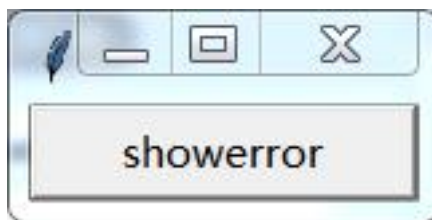




## 7.2.10 消息窗口（消息框）

### Error消息窗口

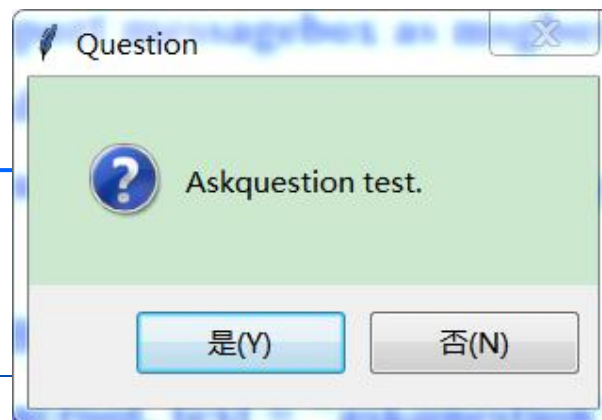
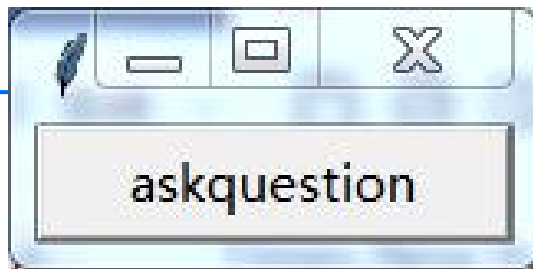
```
import tkinter as tk
from tkinter import messagebox as msgbox
def btn3_clicked():
    msgbox.showerror("Error", "Showerror test.")
root = tk.Tk()
root.title("MsgBox Test")
btn3 = tk.Button(root, text = "showerror", command = btn3_clicked)
btn3.pack(fill = tk.X)
root.mainloop()
```



## 7.2.10 消息窗口（消息框）

### Question消息窗口

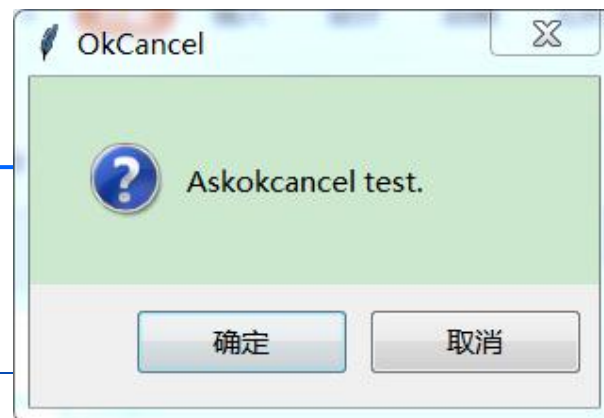
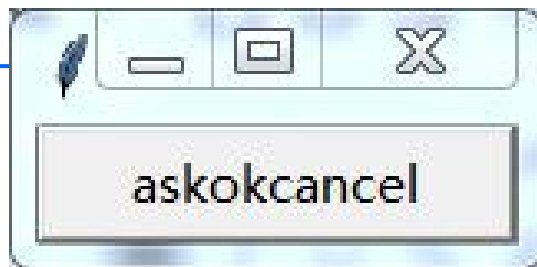
```
import tkinter as tk
from tkinter import messagebox as msgbox
def btn4_clicked():
    result = msgbox.askquestion("Question", "Askquestion test.")
    print(result)  #点击【是】 result = 'yes', 点击【否】 result = 'no'
root = tk.Tk()
root.title("MsgBox Test")
btn4 = tk.Button(root, text = "askquestion", command = btn4_clicked)
btn4.pack(fill = tk.X)
root.mainloop()
```



## 7.2.10 消息窗口（消息框）

### OkCancel消息窗口

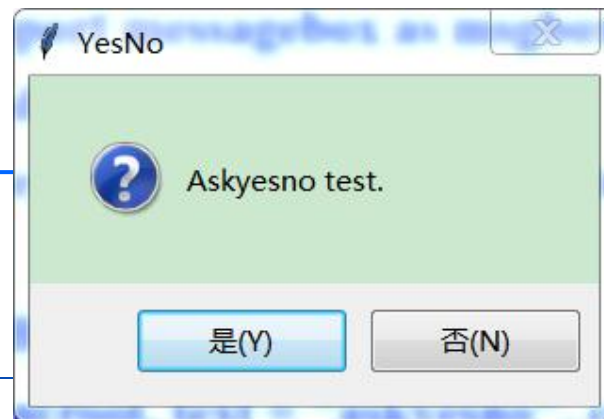
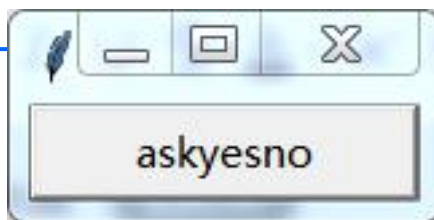
```
import tkinter as tk
from tkinter import messagebox as msgbox
def btn5_clicked():
    result = msgbox.askokcancel("OkCancel", "Askokcancel test.")
    print(result)  #result = True/False
root = tk.Tk()
root.title("MsgBox Test")
btn5 = tk.Button(root, text = "askokcancel", command = btn5_clicked)
btn5.pack(fill = tk.X)
root.mainloop()
```



## 7.2.10 消息窗口（消息框）

### YesNo消息窗口

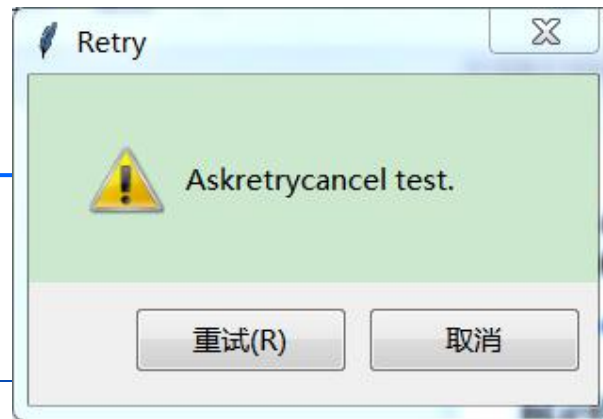
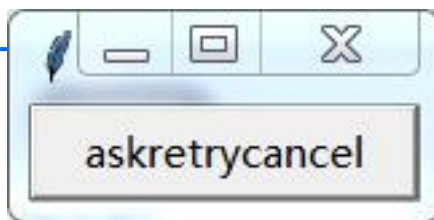
```
import tkinter as tk
from tkinter import messagebox as msgbox
def btn6_clicked():
    result = msgbox.askyesno("YesNo", "Askyesno test.")
    print(result)           #result = True/False
root = tk.Tk()
root.title("MsgBox Test")
btn6 = tk.Button(root, text = "askyesno", command = btn6_clicked)
btn6.pack(fill = tk.X)
root.mainloop()
```



## 7.2.10 消息窗口（消息框）

### Retry消息窗口

```
import tkinter as tk
from tkinter import messagebox as msgbox
def btn7_clicked():
    m = msgbox.askretrycancel("Retry", "Askretrycancel test.")
    print(m)      #点击重试返回'True', 点击取消返回'False'
root = tk.Tk()
root.title("MsgBox Test")
btn7 = tk.Button(root, text = "askretrycancel", command = btn7_clicked)
btn7.pack(fill = tk.X)
root.mainloop()
```



## 7.2.11 Frame框架组件

- Frame组件是框架组件，可以分组组织其他子组件，负责安排其他组件的位置。
- Frame组件在屏幕上显示为一个矩形区域，作为显示其他组件的容器。



## 7.2.11 Frame框架组件

### 1. 创建和显示Frame对象

- 创建Frame对象的方法：

**Frame对象 = Frame(窗口对象, height = 高度, width = 宽度, bg = 背景色, .....)**

- 显示Frame对象的方法：**Frame对象.pack()**

### 2. 向Frame组件中添加子组件

- 在创建子组件的时候，指定它的容器为Frame组件即可，例如：

**Lable(Frame对象, text = 'Hello').pack()**

### 3. 带有标题的Frame组件

- LableFrame组件是有标题的Frame组件，例如：

**LableFrame(窗口对象, height = 高度, width = 宽度, text = 标题).pack()**

## 7.2.11 Frame框架组件

### 实例

```
import tkinter as tk
root = tk.Tk()
root.title("使用Frame组件的例子")
f1 = tk.Frame(root)
f1.pack()
f2 = tk.Frame(root)
f2.pack()
f3 = tk.LabelFrame(root, text = '第3个Frame')
f3.pack( side = tk.BOTTOM )
```

#创建窗口对象  
#设置窗口标题  
#创建第1个Frame组件  
#创建第2个Frame组件  
#创建有标签的Frame框架  
#放置在窗口底部



## 7.2.11 Frame框架组件

### 实例

```
redbutton = tk.Button(f1, text="Red", fg="red")
redbutton.pack( side = tk.LEFT)
brownbutton = tk.Button(f1, text="Brown", fg="brown")
brownbutton.pack( side = tk.LEFT )
bluebutton = tk.Button(f1, text="Blue", fg="blue")
bluebutton.pack( side = tk.LEFT )
blackbutton = tk.Button(f2, text="Black", fg="black")
blackbutton.pack()
greenbutton = tk.Button(f3, text="Green", fg="Green")
greenbutton.pack()
root.mainloop()
```



## 7.2.11 Frame框架组件

### 4. 刷新Frame对象

- 用Python做GUI图形界面，可以使用 **after** 方法每隔几秒刷新GUI图形界面，例如如下代码实现移动电子广告效果，使用after不断移动label。

```
import tkinter as tk
colors = ('red','orange','yellow','green','blue','purple')
root = tk.Tk()                                #创建窗口对象
f = tk.Frame(root, height = 100, width = 200)
color = 0
f['bg'] = colors[color]
lab1 = tk.Label(f, text = r'人生苦短，我用Python')
x = 0
```

## 7.2.11 Frame框架组件

### 4. 刷新Frame对象

- 用Python做GUI图形界面，可以使用 **after** 方法每隔几秒刷新GUI图形界面，例如如下代码实现移动电子广告效果，使用after不断移动label。

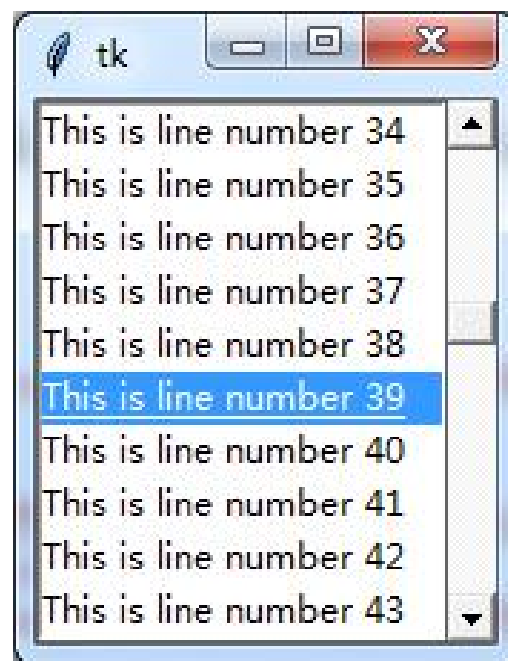
```
def foo():  
    global x  
    global color  
    x = x+10  
    if x>200:  
        x = 0  
    f['bg'] = colors[color]  
    color = (color+1)%(len(colors))  
    lab1.place(x = x, y = 30)  
    f.after(500, foo)           #嵌套调用
```

```
f.pack()  
foo()  
root.mainloop()
```



## 7.2.12 Scrollbar滚动条组件

- Scrollbar组件用于滚动一些组件的可见范围，根据方向可分为垂直滚动条和水平滚动条。Scrollbar 组件常常被用于实现文本、画布和列表框的滚动
- Scrollbar 组件通常与 Text 组件、Canvas 组件和 Listbox 组件一起使用，水平滚动条还能跟 Entry 组件配合。



## 7.2.12 Scrollbar滚动条组件

### 添加滚动条

➤ 在某个组件上添加垂直滚动条，需要2个步骤

- (1) 设置该组件的yscrollbarcommand选项为Scrollbar组件的set方法；
- (2) 设置Scrollbar组件的command选项为该组件的yview方法

```
import tkinter as tk
def print_item(event):    #鼠标松开事件打印出当前选中项内容
    print (mylist.get(mylist.curselection()))
root = tk.Tk()
mylist = tk.Listbox(root) #创建列表框
mylist.bind('<ButtonRelease-1>', print_item)    #绑定鼠标松开事件
for line in range(100):    #列表框内追加100项内容
    mylist.insert(tk.END, "This is line number " + str(line))
```

## 7.2.12 Scrollbar滚动条组件

### 添加滚动条

- 在某个组件上添加垂直滚动条，需要2个步骤
  - (1) 设置该组件的yscrollbarcommand选项为Scrollbar组件的set方法；
  - (2) 设置Scrollbar组件的command选项为该组件的yview方法

```
mylist.pack( side = LEFT, fill = BOTH )  
scrollbar = tk.Scrollbar(root)  
scrollbar.pack( side = tk.RIGHT, fill=tk.Y )  
mylist.configure(yscrollcommand = scrollbar.set)  
scrollbar.config( command = mylist.yview )  
root.mainloop()
```

## 第7章 Tkinter图形界面设计

1	Python图形开发库
2	常用Tkinter组件的使用
3	图形绘制
4	Tkinter字体
5	Python事件处理
6	图形界面程序设计的应用

## 7.3 图形绘制

### 7.3.1 Canvas画布组件

- Canvas (画布)是一个长方形的区域，用于图形绘制或复杂的图形界面布局。可以在画布上绘制图形、文字，放置各种组件和框架。

- 创建Canvas对象的语法格式：

**Canvas对象 = Canvas(窗口对象, 选项, .....)**

- 显示Canvas对象的方法：

**Canvas对象.pack()**

```
import tkinter as tk
root = tk.Tk()
cv = tk.Canvas(root, bg = 'white', width = 300, height = 120)
cv.create_line(10, 10, 100, 80, width=2, dash=7)
cv.pack()
root.mainloop()
```



## 7.3 图形绘制

### 7.3.2 Canvas上的图形对象

➤ Canvas画布上可以绘制各种图形对象。通过调用如下绘制函数实现：

- `create_arc()`            绘制圆弧。
- `create_line()`           绘制直线。
- `create_bitmap()`         绘制位图。
- `create_image()`          绘制位图图像。
- `create_oval()`           绘制椭圆。
- `create_polygon()`       绘制多边形。
- `create_window()`         绘制子窗口。
- `create_text()`           创建一个文字对象

## 7.3 图形绘制

### 7.3.2 Canvas上的图形对象

- Canvas上每个绘制对象都会返回一个对象ID，例如：

`id1 = cv.create_line(10, 10, 100, 80, width = 2, dash = 7)` #绘制直线

- 移动指定图形对象：

`Canvas对象.move(图形对象ID, x坐标偏移量, y坐标偏移量)`

- 删除指定的图形对象

`Canvas对象.delete(图形对象ID)`

- 缩放指定的图形对象

`Canvas对象.scale(图形对象ID, x轴偏移量, y轴偏移量,  
x轴缩放比例, y轴缩放比例)`

## 7.3 图形绘制

### 7.3.2 Canvas上的图形对象

- 创建图形对象时可以使用属性tags设置图形的标签，拥有相同标签的图形可以设置共同的属性，如边框颜色等

```
import tkinter as tk
root = tk.Tk()
# 创建一个Canvas，设置其背景色为白色，
cv = tk.Canvas(root, bg = 'white', width = 200, height = 200)
#使用tags给第一个矩形指定3个tag
cv.create_rectangle(10,10,110,110, tags = ('r1','r2','r3'))
cv.pack()
cv.create_rectangle(20,20,80,80, tags = 'r3') #使用tags给第2个矩形指定1个tag
for item in cv.find_withtag('r3'): # 将所有与tag('r3')绑定的item边框颜色设置为蓝色
    cv.itemconfig(item, outline = 'blue')
root.mainloop()
```

# 目录

---

## 第7章 Tkinter图形界面设计

1	Python图形开发库
2	常用Tkinter组件的使用
3	图形绘制
4	Tkinter字体
5	Python事件处理
6	图形界面程序设计的应用

# 7.4 Tkinter字体

## 7.4.1 通过元组表示字体

- 通过组件的font属性，可以设置其显示文本的字体。
- 设置组件字体前首先要能表示一个字体，字体可以通过3个元素的元组来表示：**(font family, size, modifiers)** 其中，
  - ❑ font family是字体名；
  - ❑ size为字体大小单位为point；
  - ❑ modifiers为包含粗体、斜体、下划线的样式修饰符。



```
import tkinter as tk
```

```
root = tk.Tk()
```

```
for ft in ('Arial', ('Courier New',19,'italic'), ('Comic Sans MS',), 'Fixdsys', ('MS Sans  
Serif',), ('MS Serif',), 'Symbol','System', ('Times New Roman',), 'Verdana'):
```

```
    tk.Label(root, text = 'hello sticky', font = ft ).grid()
```

```
root.mainloop()
```

## 7.4 Tkinter字体

### 7.4.2 通过Font对象表示字体

➤ 也可以使用tkFont.Font来创建字体。格式如下：

**ft = tkFont.Font(family = '字体名',size ,weight ,slant, underline, overstrike)**

➤ 其中：

- ❑ size为字体大小；
- ❑ weight为字形——'bold'或'normal'；
- ❑ slant='italic'或'normal', 'italic'为斜体；
- ❑ underline=1或0, 1为下划线；
- ❑ overstrike=1或0, 1为删除线。

➤ 例如：

**ft = Font(family="Helvetica",size=36,weight="bold")**

## 7.4 Tkinter字体

### 7.4.2 通过Font对象表示字体

```
import tkinter as tk
import tkinter.font as ft          # 引入字体模块
root = tk.Tk()

# 指定字体名称、大小、样式
ft = ft.Font(family = 'Fixdsys',size = 20,weight = 'bold')
tk.Label(root, text = 'hello sticky', font = ft ).grid()      # 创建一个Label
root.mainloop()
```



# 目录

---

## 第7章 Tkinter图形界面设计

1	Python图形开发库
2	常用Tkinter组件的使用
3	图形绘制
4	Tkinter字体
5	Python事件处理
6	图形界面程序设计的应用



## 7.5 Python事件处理

---

- 所谓事件 (event) 就是程序上发生的事。例如用户敲击键盘上某一个键或是单击、移动鼠标。而对于这些事件，程序需要做出反应。
- Tkinter提供的组件通常都有自己可以识别的事件。例如当按钮被单击时执行特定操作或是当一个输入栏成为焦点，而你又敲击了键盘上的某些按键，你所输入的内容就会显示在输入栏内。
- 程序可以使用事件处理函数来指定当触发某个事件时所做的反应（操作）。

## 7.5.1 事件类型

### 事件类型的通用格式

➤ `<[modifier-]...type[-detail]>`

- ❑ 事件类型必须放置于尖括号<>内。
- ❑ modifier用于组合键定义，例如Control、Alt
- ❑ type描述了类型，例如键盘按键、鼠标单击。
- ❑ detail用于明确定义是哪一个键或按钮的事件，例如1表示鼠标左键、2表示鼠标中键、3表示鼠标右键。

➤ **Python中事件主要有：**

- ❑ **键盘事件**
- ❑ **鼠标事件**
- ❑ **窗体事件**

# 7.5.1 事件类型

## 鼠标事件

### ➤ 鼠标单击事件

<Button-1>	<Button-2>	<Button-3>	<Button-4>	<Button-5>
单击鼠标左键	单击鼠标中键	单击鼠标右键	向上滚动滑轮	向下滚动滑轮

### ➤ 鼠标双击事件

<Double-Button-1>	<Double-Button-2>	<Double-Button-3>
鼠标左键双击	鼠标中键双击	鼠标右键双击

### ➤ 鼠标释放事件

<ButtonRelease-1>	<ButtonRelease-2>	<ButtonRelease-3>
鼠标左键释放	鼠标中键释放	鼠标右键释放

## 7.5.1 事件类型

### 鼠标事件（续）

#### ➤ 鼠标按下并移动事件（即拖动）

<B1-Motion>	<B2-Motion>	<B3-Motion>
鼠标左键拖动	鼠标中键拖动	鼠标右键拖动

#### ➤ 鼠标其他操作事件

<Enter>	<FocusIn>	<Leave>	<FocusOut>
鼠标进入控件（放到控件上面）	控件获得焦点	鼠标移出控件	控件失去焦点

# 7.5.1 事件类型

## 键盘事件

### ➤ 键盘按下事件

- ❑ **<Key>**: 键盘按下, 事件event中的keycode, char都可以获取按下的键值
- ❑ **<KeyPress>**: 按下键盘某键时触发, 可以在detail部分指定是哪个键
- ❑ **<KeyRelease>**: 释放键盘某键时触发, 可以在detail部分指定是哪个键
- ❑ **<Return>**: 键位绑定, 回车键, 其它还有<BackSpace>, <Escape>, <Left>, <Up>, <Right>, <Down>等等

### ➤ 控件属性改变事件

- ❑ **<Configure>**: 控件大小改变, 新的控件大小会存储在事件event对象中的width 和 height 属性传递, 部分平台上该事件也代表控件位置改变。

## 7.5.1 事件类型

Python中事件主要有：键盘事件，鼠标事件，窗体事件

➤ 事件类型的通用格式：<[modifier-]...type[-detail]>

➤ 键盘事件

名称	描述
KeyPress	按下键盘某键时触发，可以在detail部分指定是哪个键
KeyRelease	释放键盘某键时触发，可以在detail部分指定是哪个键

## 7.5.1 事件类型

Python中事件主要有：键盘事件，鼠标事件，窗体事件

➤ 事件类型的通用格式：<[modifier-]...type[-detail]>

➤ 鼠标事件

名称	描述
ButtonPress或Button	按下鼠标某键时触发，可以在detail部分指定是哪个键
ButtonRelease	释放鼠标某键时触发，可以在detail部分指定是哪个键
Motion	点中组件的同时拖曳组件移动时触发
Enter	当鼠标指针移进某个组件时触发
Leave	当鼠标指针移出某个组件时触发
MouseWheel	当鼠标滚轮滚动时触发

## 7.5.1 事件类型

Python中事件主要有：键盘事件，鼠标事件，窗体事件

### ➤ 窗体事件

名称	描述
visibility	当组件变为可视状态时触发
Unmap	当组件由显示状态变为隐藏状态时触发
Map	当组件由隐藏状态变为显示状态时触发
Expose	当组件由原本被其他组件遮盖的状态中暴露出来时触发
FocusIn	组件获得焦点时触发
FocusOut	组件失去焦点时触发
Configure	当改变组件大小时触发。例如拖曳窗体边缘
Property	当窗体的属性被删除或改变时触发，属于Tk的核心事件
Destroy	当组件被销毁时触发
Activate	与组件中的状态选项有关，表示由不可用转为可用时触发
Deactivate	与组件中的状态选项有关，表示由可用转为不可用时触发



## 7.5.2 事件绑定

### 1. 创建组件对象时指定

- 程序建立一个处理某一事件的事件处理函数，称之为**绑定**。
- 创建组件对象实例时，可通过其命名参数command指定事件处理函数。例

```
import tkinter as tk
from tkinter import messagebox
root = tk.Tk()

def callback(): #定义一个事件处理函数
    messagebox.showinfo("Python command","人生苦短、我用Python")

Bu1=tk.Button(root, text="设置事件调用命令",command=callback)
Bu1.pack()
root.mainloop()
```

## 7.5.2 事件绑定

### 2. 实例绑定 (**最常用事件绑定方式**)

- 调用组件对象实例方法bind可为指定组件实例绑定事件。

**组件对象实例名.bind("<事件类型>", 事件处理函数)**

- 例如假设声明了一个名为canvas的Canvas组件对象，想在canvas上按下鼠标左键时画上一条线，可以这样实现：

**canvas.bind("<Button-1>", drawline)**

- 其中，第一个参数是事件描述符，第二个参数是事件处理函数，这里只是函数声明，不需要后面的圆括号，Tkinter会把此函数填入参数然后调用运行。

## 7.5.2 事件绑定

### 2. 实例绑定

➤ 例如:

```
import tkinter as tk
def drawline(event):
    cv.create_line(10,10,200,100,width =3,dash=7)
root = tk.Tk()
cv = tk.Canvas(root, bg = 'white', width = 200, height = 120)
cv.bind("<Button-1>",drawline)
cv.create_rectangle(20,20,110,110,outline='red',stipple='gray12',fill='green')
cv.pack()
root.mainloop()
```

## 7.5.2 事件绑定

### 3. 类绑定

- 将事件与一组件类绑定。调用任意组件实例的.bind\_class()函数为特定组件类绑定事件。

组件实例名.bind\_class ("组件类", "<事件类型>", 事件处理函数)

例如：

```
import tkinter as tk
def drawline(event):
    cv.create_line(10,10,200,100,width =3,dash=7)
root = tk.Tk()
cv = tk.Canvas(root, bg = 'white', width = 200, height = 120)
cv.create_rectangle(20,20,110,110,outline='red',stipple='gray12',fill='green')
cv.bind_class("Canvas", "<Button-1>", drawline)
cv.pack()
root.mainloop()
```

## 7.5.2 事件绑定

---

### 4. 程序界面绑定

- 当无论在哪一组件实例上触发某一事件，程序都作出相应的处理。
- 例如，可以将PrintScreen键与程序中的所有组件对象绑定，这样整个程序界面就能处理打印屏幕的事件了。调用任意组件实例的.bind\_all()函数为程序界面绑定事件。

组件实例名.bind\_all("<事件类型>", 事件处理函数)

- 例如可以这样实现打印屏幕：

`widget.bind_all("<Key-Print>", printScreen)。`

## 7.5.2 事件绑定

### 4. 程序界面绑定

- 调用任意组件实例的.bind\_all()函数为程序界面绑定事件。

组件实例名.bind\_all("<事件类型>", 事件处理函数)

- 例如可以这样实现打印屏幕：

widget.bind\_all("<Key-Print>", printScreen)。

```
import tkinter as tk
def drawline(event):
    cv.create_line(10,10,200,100,width =3,dash=7)
root = tk.Tk()
cv = tk.Canvas(root, bg = 'white', width = 200, height = 120)
cv.create_rectangle(20,20,110,110,outline='red',stipple='gray12',fill='green')
cv.bind_all("<Button-1>",drawline)
cv.pack()
root.mainloop()
```

## 7.5.2 事件绑定

### 5. 标识绑定

- 在Canvas画布中绘制各种图形，将图形与事件绑定可以使用标识绑定tag\_bind()函数。预先为图形定义标识tag后，通过标识tag来绑定事件。例如：`cv.tag_bind('r1','<Button-1>',printRect)`

```
import tkinter as tk
def drawline(event):
    cv.create_line(10,10,200,100,width =3,dash=7)
root = tk.Tk()
cv = tk.Canvas(root, bg = 'white', width = 200, height = 120)
cv.create_rectangle(20,20,110,110,outline='red',stipple='gray12',fill='green',
tags = 'rt')
cv.tag_bind("rt", "<Button-1>", drawline)
cv.pack()
root.mainloop()
```

## 7.5.3 事件处理函数

### 1. 定义事件处理函数

- 事件处理函数往往带有一个event参数。触发事件调用事件处理函数时，将传递Event对象实例。

```
def callback(event):    #事件处理函数
    showinfo("Python command","人生苦短、我用Python")
```



## 7.5.3 事件处理函数

### 2. Event事件处理参数的属性

➤ Event对象实例可以获取各种相关的参数。Event事件对象主要参数属性如

名称	描述
.x, .y	鼠标相对于组件对象左上角的坐标
.x_root, .y_root	鼠标相对于屏幕对象左上角的坐标
.keysym	字符串命名按键
.keysym_num	数字代码命名按键
.keycode	键码，但是它不能反映事件前缀Alt, Control, Shift, Lock, 并且不区分大小写按键
.time	时间
.type	事件类型
.widget	触发事件的对应组件
.char	字符

# 7.5.3 事件处理函数

## 2. Event事件处理参数的属性

➤ Event事件对象按键详细信息说明

.keysym	.keycode	.keysym_num	说明
Alt_L	64	65513	左侧Alt键
Alt_R	113	65514	右侧Alt键
BackSpace	22	65288	BackSpace键
Cancel	110	65387	Pause Break键
F1~F11	67~77	65470~65480	功能键F1~F11
Print	111	65377	打印屏幕键

## 7.5.3 事件处理函数

### 2. 定义事件处理函数——实例

➤ 触发keyPress键盘事件的例子

```
import tkinter as tk
def printkey(event):
    print('你按下了:' + event.char)
root = tk.Tk()
entry = tk.Entry(root)

#给输入框绑定按键监听事件<KeyPress>以监听任何按键
entry.bind('<KeyPress>', printkey)
entry.pack()
root.mainloop()
```

#导入tkinter  
#定义的函数监听键盘事件  
#实例化tk  
#实例化一个单行输入框

# 目录

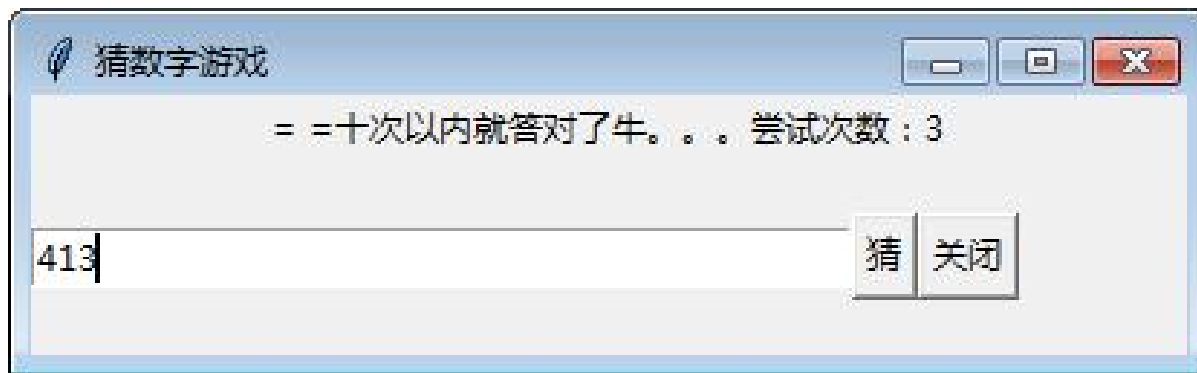
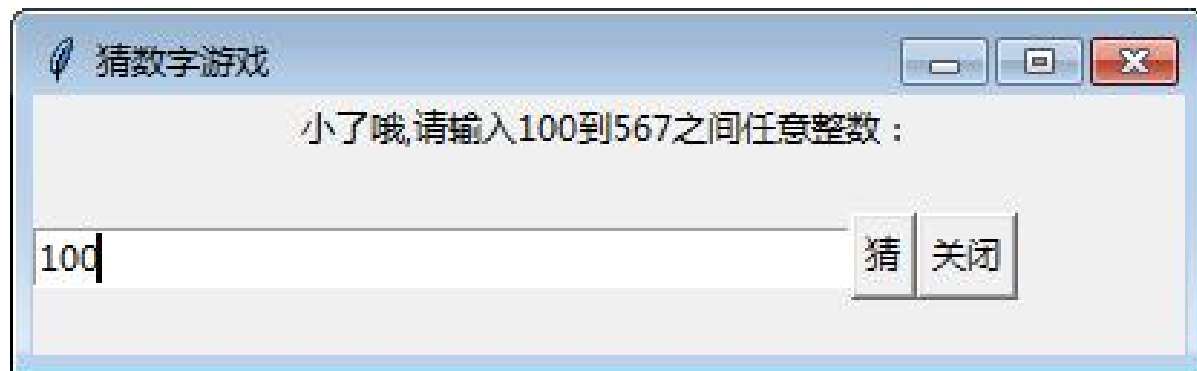
## 第7章 Tkinter图形界面设计

1	Python图形开发库
2	常用Tkinter组件的使用
3	图形绘制
4	Tkinter字体
5	Python事件处理
6	图形界面程序设计的应用

## 7.6.1 开发猜数字游戏

使用tkinter开发猜数字游戏，运行效果如图所示。

- 游戏中电脑随机生成1024以内数字，玩家去猜，如果猜得数字过大过小都会提示，程序要统计玩家猜的次数。



## 7.6.1 开发猜数字游戏

使用tkinter开发猜数字游戏，运行效果如图所示。

```
import tkinter as tk
```

```
import sys
```

```
import random
```

```
import re
```

```
number = random.randint(0,1024)
```

**#玩家要猜的数字**

```
running = True
```

```
num = 0
```

**#猜的次数**

```
nmaxn = 1024
```

**#提示猜测范围的最大数**

```
nminn = 0
```

**#提示猜测范围的最小数**

```
def eBtnClose(event):
```

```
    root.destroy()
```

## 7.6.1 开发猜数字游戏

使用tkinter开发猜数字游戏，运行效果如图所示。

```
def eBtnGuess(event):
    global nmaxn
    global nminn
    global num
    global running
    if running:
        val_a = int(entry_a.get())    #获取猜的数字输入并转换成数字
        if val_a == number:
            labelqval("恭喜答对了！")
            num+=1
            running = False
            numGuess()                #显示猜的次数
        elif val_a < number:           #猜小了
            if val_a > nminn:
                nminn = val_a         #修改提示猜测范围的最小数
                num+=1
                labelqval("小了哦,请输入"+str(nminn)+"到"+str(nmaxn)+"之间任意整数：")
            else:
                if val_a < nmaxn:
                    nmaxn = val_a     #修改提示猜测范围的最大数
                    num+=1
                    labelqval("大了哦,请输入"+str(nminn)+"到"+str(nmaxn)+"之间任意整数：")
        else:
            labelqval('你已经答对啦...')
```

## 7.6.1 开发猜数字游戏

使用tkinter开发猜数字游戏，运行效果如图所示。

**#显示猜的次数**

**def numGuess():**

**if num == 1:**

**labelqval('我靠！一次答对！')**

**elif num < 10:**

**labelqval('==十次以内就答对了牛。。。尝试次数: '+str(num))**

**else:**

**labelqval('好吧，您都试了超过10次了。。。尝试次数: '+str(num))**

**def labelqval(vText):**

**label\_val\_q.config(label\_val\_q,text=vText)** **#修改提示标签文字**



## 7.6.1 开发猜数字游戏

使用tkinter开发猜数字游戏，运行效果如图所示。

```
root = tk.Tk(className="猜数字游戏")
root.geometry("400x90+200+200")
label_val_q = tk.Label(root,width="80")           #提示标签
label_val_q.pack(side = "top")
entry_a = tk.Entry(root,width="40")               #单行输入文本框
btnGuess = tk.Button(root,text="猜")              #猜按钮
entry_a.pack(side = "left")
entry_a.bind('<Return>',eBtnGuess)                 #绑定事件
btnGuess.bind('<Button-1>',eBtnGuess)              #猜按钮
btnGuess.pack(side = "left")
btnClose = tk.Button(root,text="关闭")             #关闭按钮
btnClose.bind('<Button-1>',eBtnClose)
btnClose.pack(side="left")
labelqval("请输入0到1024之间任意整数: ")
entry_a.focus_set()
print(number)
root.mainloop()
```

## 7.6.2 扑克牌发牌程序窗体图形版

### 游戏初步——扑克牌发牌程序窗体图形版

- 4名牌手打牌，电脑随机将52张牌（不含大小鬼）发给4名打牌，在屏幕上显示每位牌手的牌。程序的运行效果如图所示。

