第3章

Python图形用户界面

本章主要介绍图形界面。在介绍 Python 图形界面编程前，首先简单介绍一下 Python 的图形界面库。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：Tkinter 模块(Tk 接口)是 Python 的标准 Tk GUI 工具包的接口，Tk 和 Tkinter 可以在大多数的 UNIX 平台下使用，同样可以应用在 Windows 和 macOS 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 程序员很方便地创建完整的、功能健全的 GUI。

(3) Jython：Jython 程序可以和 Java 无缝集成。除了一些标准模块，Jython 还使用 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。例如，Jython 的用户界面将使用 Swing、AWT 或者 SWT。Jython 可以被动态或静态地编译成 Java 字节码。

其中，Tkinter 是 Python 的标准 GUI 库。Python 使用 Tkinter 可以快速地创建 GUI 应用程序。

由于 Tkinter 是内置到 Python 的安装包中，只要安装好 Python 之后就能导入 Tkinter 库，而且 IDLE 也是用 Tkinter 编写而成的，对于简单的图形界面，Tkinter 能应付自如。导入 Tkinter 库的方法为：

```
import tkinter
```

在开始 GUI 编程之前，需要先了解几个概念：窗体控件、事件驱动、布局管理。

(1) 窗体控件：包括窗体、标签、按钮、列表框、滚动条等。

(2) 事件驱动：单击按钮及释放、鼠标移动、按回车键等。

(3) 布局管理：Tk 有 3 种布局管理器，分别为 Placer、Packer、Grid。

3.1 布局管理

GUI 编程就相当于搭积木，每个积木块应该放在哪里，每个积木块显示为多大，也就是对大小和位置都需要进行管理，而布局管理器正是负责管理各组件的大小和位置的。此外，当用户调整了窗口的大小后，布局管理器还会自动调整窗口中各组件的大小和位置。

3.1.1 Pack 布局管理器

如果使用 Pack 布局，那么当程序向容器中添加组件时，这些组件会依次向后排列，排列方

向既可以是水平的,也可以是垂直的。

【例 3-1】 利用 Pack 布局管理器制作钢琴的布局。

```
"""
Pack 布局
"""
from tkinter import *
from tkinter import messagebox
import random
import matplotlib.pyplot as plt                    # plt 用于显示图片
#matplotlib inline
class Application(Frame):
    def __init__(self, mester = None):
        super().__init__(mester)
        self.mester = mester
        self.pack()
        self.creatWidget()
    def creatWidget(self):
        f1 = Frame(root)
        f1.pack()
        f2 = Frame(root)
        f2.pack()
        btnText = ('公平', '民主', '开放', '自由', '法治', '正义')
        for txt in btnText:
            Button(f1, text = txt).pack(side = 'left', padx = '10')
        for i in range(1, 15):
            Label(f2, width = 4, height = 10, borderwidth = 1, relief = 'solid',
                  bg = 'black' if i % 2 == 0 else 'white').pack(side = 'left', padx = '1')
if __name__ == '__main__':
    root = Tk()
    root.geometry('400x200 + 200 + 200')
    app = Application(mester = root)
    root.mainloop()
```

运行程序,效果如图 3-1 所示。

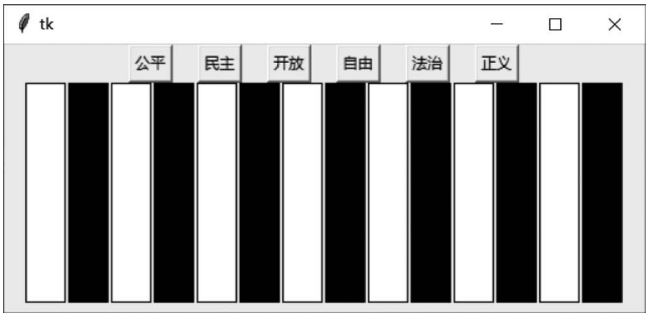

图 3-1 钢琴布局

pack()方法通常可支持如下选项。

- anchor:当可用空间大于组件所需求的大小时,该选项决定组件被放置在容器的何处。该选项支持 N(北,代表上)、E(东,代表右)、S(南,代表下)、W(西,代表左)、NW(西北,代表左上)、NE(东北,代表右上)、SW(西南,代表左下)、SE(东南,代表右下)、CENTER(中,默认值)这些值。

- expand：该 bool 值指定当父容器增大时是否拉伸组件。
- fill：设置组件是否沿水平或垂直方向填充。该选项支持 NONE、X、Y、BOTH 四个值，其中，NONE 表示不填充，BOTH 表示沿着两个方向填充。
- ipadx：指定组件在 x 方向(水平)上的内部留白(padding)。
- ipady：指定组件在 y 方向(垂直)上的内部留白(padding)。
- padx：指定组件在 x 方向(水平)上与其他组件的间距。
- pady：指定组件在 y 方向(垂直)上与其他组件的间距。
- side：设置组件的添加位置，可以设置为 TOP、BOTTOM、LEFT 或 RIGHT 这四个值的其中之一。

3.1.2　Grid 布局管理器

Grid 布局管理器可以说是 Tkinter 这三个布局管理器中最灵活多变的。使用一个 Grid 就可以简单地实现用很多个框架和 Pack 搭建起来的效果。

Grid 把组件空间分解成一个网格进行维护，即按照行、列的方式排列组件，组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组件会被依次上下排列，列号相同而行号不同的几个组件会被依次左右排列。

在 Python 程序中，调用 grid()方法进行 Grid 布局。在调用 grid()方法时可传入多个选项，该方法支持的 ipadx、ipady、pady 与 pack()方法的这些选项相同。而 grid()方法额外增加了如下选项。

- column：单元格的列号为从 0 开始的正整数。
- columnspan：跨列，跨越的列数，正整数。
- row：单元格的行号为从 0 开始的正整数。
- rowspan：跨行，跨越的行数，正整数。
- sticky：组件紧贴所在单元格的某一角，该选项支持 N(北，代表上)、E(东，代表右)、S(南，代表下)、W(西，代表左)、NW(西北，代表左上)、NE(东北，代表右上)、SW(西南，代表左下)、SE(东南，代表右下)、CENTER(中，默认值)这些值。

【例 3-2】 利用 Grid 布局管理器设计一个计算器界面。

```
"""Grid 布局管理器"""
# coding:utf-8
from tkinter import *
from tkinter import messagebox
import random

class Application(Frame):
    """一个经典的 GUI 程序类写法"""
    def __init__(self, master=None):
        super().__init__(master) # super 代表的是父类的定义，而不是父类的对象
        self.master = master
        self.pack()
        self.createWidget()

    def createWidget(self):
        btnText = (('MC', 'M+', 'M-', 'MR'), ('C', '±','÷','×'), ('7','8','9','-'), ('4','5',
'6','+'), ('1','2','3','='),('0','·'))
```

```
            Entry(self).grid(row = 0, column = 0, columnspan = 4)
            for rindex,r in enumerate(btnText):
                  for cindex,c in enumerate(r):
                        if c == '=':
                              Button(self, text = c, width = 2).grid(row = rindex + 1, column =
cindex, rowspan = 2, sticky = NSEW)
                        elif c == '0':
                              Button(self, text = c, width = 2).grid(row = rindex + 1, column =
cindex, columnspan = 2, sticky = NSEW)
                        elif c == '·':
                              Button(self, text = c, width = 2).grid(row = rindex + 1, column =
cindex + 1, sticky = NSEW)
                        else:
                               Button(self, text = c, width = 2).grid(row = rindex + 1, column =
cindex, sticky = NSEW)
if __name__ == "__main__":
      root = Tk()
      root.geometry("170x220 + 200 + 300")
      root.title('canvas')
      app = Application(master = root)
      root.mainloop()
```

运行程序,效果如图 3-2 所示。

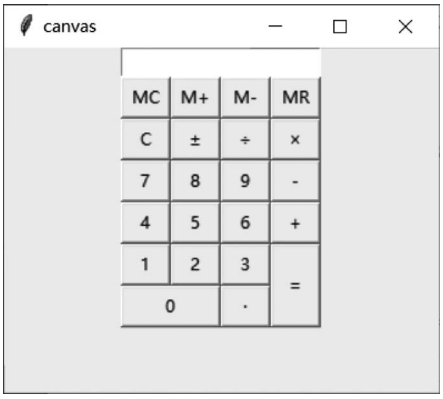

图 3-2　计算器界面

3.1.3　Place 布局管理器

Place 布局就是其他 GUI 编程中的“绝对布局”,这种布局方式要求程序显式指定每个组件的绝对位置或相对于其他组件的位置。

如果要使用 Place 布局,调用相应组件的 place()方法即可。在使用该方法时同样支持一些详细的选项,关于这些选项的介绍如下。

- x:指定组件的 X 坐标。x 为 0,代表位于最左边。
- y:指定组件的 Y 坐标。y 为 0,代表位于最右边。
- relx:指定组件的 X 坐标,以父容器总宽度为单位 1,该值应该为 0.0～1.0,其中,0.0 代表位于窗口最左边,1.0 代表位于窗口最右边,0.5 代表位于窗口中间。
- rely:指定组件的 Y 坐标,以父容器总宽度为单位 1,该值应该为 0.0～1.0,其中,0.0 代表位于窗口最上边,1.0 代表位于窗口最下边,0.5 代表位于窗口中间。

- width：指定组件的宽度，以 pixel 为单位。
- height：指定组件的高度，以 pixel 为单位。
- relwidth：指定组件的宽度，以父容器总宽度为单位 1，该值应该为 0.0～1.0，其中，1.0 代表整个窗口宽度，0.5 代表窗口的一半宽度。
- relheight：指定组件的高度，以父容器总高度为单位 1，该值应该为 0.0～1.0，其中，1.0 代表整个窗口高度，0.5 代表窗口的一半高度。
- bordermode：该属性支持“inside”或“outside”属性值，用于指定当设置组件的宽度、高度时是否计算该组件的边框宽度。

【例 3-3】 Place 布局管理实现。

```
"""
 Place 布局管理器
"""
from tkinter import *
from tkinter import messagebox
import random

class Application(Frame):
     def __init__(self, mester = None):
          super().__init__(mester)
          self.mester = mester
          self.pack()
          self.creatWidget()

     def creatWidget(self):
          self.photo1 = PhotoImage(file = '2.gif')
          self.puks01 = Label(root, image = self.photo1)
          self.puks01.place(x = 20, y = 50)
          self.photo2 = PhotoImage(file = '11.gif')
          self.pukse02 = Label(root, image = self.photo2)
          self.pukse02.place(x = 300, y = 50)
          self.puks01.bind_class('Label', '< Button - 1 >', self.chupai)
          self.pukse02.bind_class('Label', '< 1 >', self.chupai)
          # 为所有的 Label 增加事件处理

     def chupai(self, event):
          print(event.widget.winfo_geometry())
          print(event.widget.winfo_y())
          if event.widget.winfo_y() == 50:
               event.widget.place(y = 30)
          else:
               event.widget.place(y = 50)

if __name__ == '__main__':
     root = Tk()
     root.geometry('800x400 + 500 + 500')
     app = Application(mester = root)
     root.mainloop()
```

运行程序，效果如图 3-3 所示。

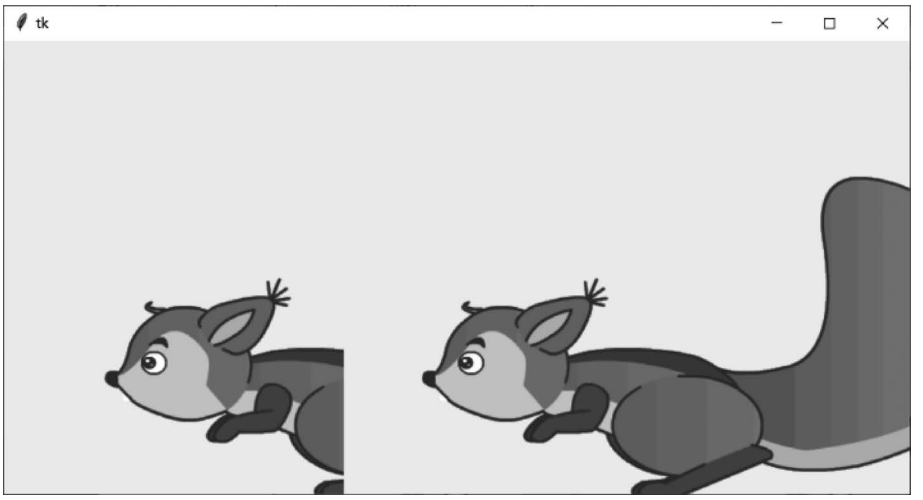

图 3-3　图片的上下管理

3.2　Tkinter 常用组件

Tkinter 中，每个组件都是一个类，创建某个组件其实就是将这个类实例化。在实例化的过程中，可以通过构造函数给组件设置一些属性，同时还必须给该组件指定一个父容器，意即该组件放置何处。最后，还需要给组件设置一个几何管理器（布局管理器）。解决了放哪里的问题，还需要解决怎么放的问题，而布局管理器就是解决怎么放问题的，即设置子组件在父容器中的放置位置。

3.2.1　Variable 类

Tkinter 支持将很多 GUI 组件与变量进行双向绑定，执行这种双向绑定后编程非常方便。

- 如果程序改变变量的值，GUI 组件的显示内容或值会随之改变。
- 当 GUI 组件的内容发生改变时（如用户输入），变量的值也会随之改变。

为了让 Tkinter 组件与变量进行双向绑定，只要为这些组件指定 variable（通常绑定组件的 value）、textvariable（通常绑定组件显示的文本）等属性即可。但这种双向绑定有一个限制，就是 Tkinter 不允许将组件和普通变量进行绑定，只能和 Tkinter 包下 Variable 类的子类进行绑定。该类包含如下几个子类。

- StringVar()：用于包装 str 值的变量。
- IntVar()：用于包装整型值的变量。
- DoubleVar()：用于包装浮点值的变量。
- BooleanVar()：用于包装 bool 值的变量。

对于 variable 变量而言，如果要设置其保存的变量值，则使用它的 set()方法；如果要得到其保存的变量值，则使用它的 get()方法。

【例 3-4】 将 Entry 组件与 StringVar 进行双向绑定。

程序既可以通过该 StringVar 改变 Entry 输入框显示的内容，也可以通过该 StringVar 获取 Entry 输入框中的内容。

```
from tkinter import *
from tkinter import ttk                                    #导入 ttk
class App:
```

```
    def __init__(self, master):
        self.master = master
        self.initWidgets()
    def initWidgets(self):
        self.st = StringVar()
        #创建 Entry 组件,将其 textvariable 绑定到 self.st 变量
        ttk.Entry(self.master, textvariable = self.st,
            width = 24,
            font = ('StSong', 20, 'bold'),
            foreground = 'red').pack(fill = BOTH, expand = YES)
        #创建 Frame 作为容器
        f = Frame(self.master)
        f.pack()
        #创建两个按钮,将其放入 Frame 中
        ttk.Button(f, text = '改变', command = self.change).pack(side = LEFT)
        ttk.Button(f, text = '获取', command = self.get).pack(side = LEFT)
    def change(self):
        books = ('图形用户界面', 'Tkinter 组件', 'Variable 类')
        import random
        #改变 self.st 变量的值,与之绑定的 Entry 的内容随之改变
        self.st.set(books[random.randint(0, 2)])
    def get(self):
        from tkinter import messagebox
        #获取 self.st 变量的值,实际上就是获取与之绑定的 Entry 中的内容
        #并使用消息框显示 self.st 变量的值
        messagebox.showinfo(title = '输入内容', message = self.st.get() )
root = Tk()
root.title("variable 测试")
App(root)
root.mainloop()
```

运行程序,界面如图 3-4 所示。单击界面中的“改变”按钮,将可以看到输入框中的内容会随之改变;如果单击界面上的“获取”按钮,将会看到程序弹出一个消息框,显示了用户在 Entry 输入框中输入的内容。

图 3-4　Entry 组件双向绑定

3.2.2　compound 选项

在 Python 中,可以为按钮或 Label 等组件同时指定文本(text)与图片(image)两个选项,其中,text 用于指定该组件上的文本;image 用于显示该组件上的图片,当同时指定这两个选项时,通常 image 会覆盖 text。但在某些时候,我们希望该组件能同时显示文本和图片,此时就需要通过 compound 选项进行控制。

compound 选项支持如下属性值。

- None:图片覆盖文字。
- LEFT 常量(值为'left'字符串):图片在左,文本在右。
- RIGHT 常量(值为'right'字符串):图片在右,文本在左。

- TOP 常量(值为'top'字符串):图片在上,文本在下。
- BOTTOM 常量(值为'bottom'字符串):图片在下,文本在上。
- CENTER 常量(值为'center'字符串):文本在图片上方。

【例 3-5】 利用 compound 选项同时使用图片和文字。

```
from tkinter import *
root = Tk()
root.title("Label 测试")
img = PhotoImage(file = "2.gif")
stext = "Python 简单、易学"
#图片位于文字左侧
label1 = Label(root, image = img, text = stext, compound = "left", bg = "lightyellow")
label1.pack()
#图片位于文字右侧
label2 = Label(root, image = img, text = stext, compound = "right", bg = "lightcyan")
label2.pack()
#图片位于文字上方
label3 = Label(root, image = img, text = stext, compound = "top", bg = "lightgreen")
label3.pack()
#图片位于文字下方
label4 = Label(root, image = img, text = stext, compound = "bottom", bg = "lightgray")
label4.pack()
# 文字覆盖图片中央,可作背景图片
label5 = Label(root, image = img, text = stext, compound = "center", bg = "lightblue", fg =
"white", font = ("微软雅黑", 24))
label5.pack()
root.mainloop()
```

运行程序,效果如图 3-5 所示。

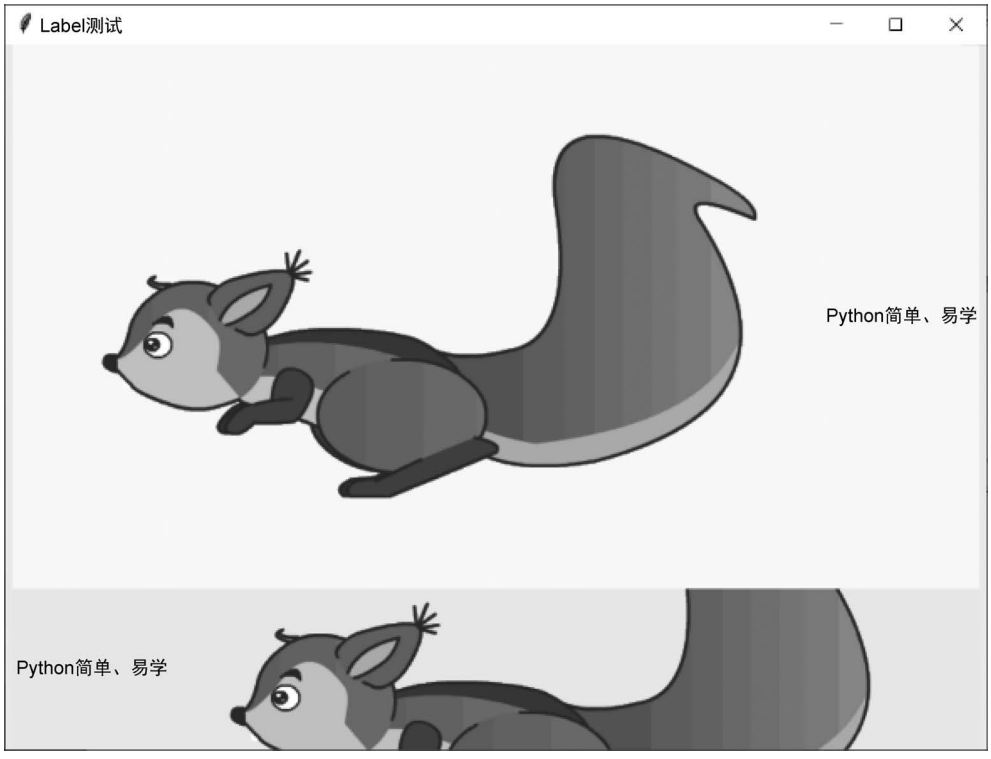

图 3-5 compound 选项

3.2.3　Entry 与 Text 组件

Entry 组件仅允许用于输入一行文本，如果用于输入的字符串长度比该组件可显示空间更长，那内容将被滚动。这意味着该字符串将不能被全部看到(可以用鼠标或键盘的方向键调整文本的可见范围)。如果希望接收多行文本的输入，可以使用 Text 组件。

不管是 Entry 还是 Text 组件，程序都提供了 get()方法来获取文本框中的内容；但如果程序要改变文本框中的内容，则需要调用二者的 insert()方法来实现。

如果要删除 Entry 或 Text 组件中的部分内容，则可通过 delete(self,first,last=None)方法实现，该方法指定删除从 first 到 last 之间的内容。

但两者之间支持的索引是不同的，由于 Entry 是单行文本框组件，因此它的索引很简单，例如，要指定第 4～8 个字符，将索引指定为(3,8)即可。但 Text 是多行文本框组件，因此它的索引需要同时指定行号和列号，例如，1.0 代表第 1 行、第 1 列(行号从 1 开始，列号从 0 开始)，如果要指定第 2 行第 3 个字符到第 3 行第 7 个字符，索引应指定为(2.2,3.6)。

提示：Entry 支持双向绑定。

【例 3-6】 将 Entry 中用户输入的字符串在 Text 文本框中显示，其中触发不同按钮，用户输入的内容将插入在与之相应的不同位置。

```
import tkinter as tk
window = tk.Tk()
window.title('Entry 与 Text 测试')
window.geometry('200x200')
e = tk.Entry(window,show = '*')
# Entry 的第一个参数是父窗口,即这里的 window
#  *表示输入的文本变为星号,在 Entry 中为不可见内容,如果为 None 则表示输入文本以原形式可见
e.pack()
def insert_point():
    var = e.get()
    t.insert('insert',var)
def insert_end():
    var = e.get()
    t.insert('end',var)
#这里的 end 表示插入到结尾,可以换为 1.2,则插入在第 1 行第 2 位后面
b1 = tk.Button(window,text = '插入点',width = 15,height = 2,command = insert_point)
b1.pack()
b2 = tk.Button(window,text = '插入端',width = 15,height = 2,command = insert_end)
b2.pack()
t = tk.Text(window,height = 2)      #这里设置文本框高,可以容纳两行
t.pack()
window.mainloop()
```

运行程序，效果如图 3-6 所示。

图 3-6　Entry 与 Text 测试效果

3.2.4 Checkbutton 组件

Checkbutton(多选按钮)组件用于实现确定是否选择的按钮。Checkbutton 组件可以包含文本或图像,可以将一个 Python 的函数或方法与之相关联,当按钮被按下时,对应的函数或方法将被自动执行。

Checkbutton 组件仅能显示单一字体的文本,但文本可以跨越多行。另外,还可以为其中的个别字符加上下画线(例如,用于表示键盘快捷键)。默认情况下,Tab 键被用于在按钮间切换。

Checkbutton 组件被用于作为二选一的按钮(通常为选择“开”或“关”的状态),当希望表达“多选多”选项的时候,可以将一系列 Checkbutton 组合起来使用。但是处理“多选一”的问题时,还是交给 Radiobutton 和 Listbox 组件来实现更合理。

【例 3-7】 创建 Checkbutton 复选框。

```
from tkinter import *
tk = Tk()
label = Label(tk, text = "你最喜欢的计算机语言", bg = "yellow", fg = "blue", width = 30)
label.grid(row = 0)
var1 = IntVar()
cbtn1 = Checkbutton(tk, text = "Python", variable = var1)
cbtn1.grid(row = 1, sticky = W)
var2 = IntVar()
cbtn2 = Checkbutton(tk, text = "MATLAB", variable = var2)
cbtn2.grid(row = 2, sticky = W)
var3 = IntVar()
cbtn3 = Checkbutton(tk, text = "TensorFlow", variable = var3)
cbtn3.grid(row = 3, sticky = W)
tk.mainloop()
```

运行程序,效果如图 3-7 所示。

图 3-7 复选按钮

3.2.5 Radiobutton 组件

Radiobutton 组件跟 Checkbutton 组件的用法基本一致,唯一不同的是 Radiobutton 实现的是“单选”的效果。

要实现这种互斥的效果,同组内的所有 Radiobutton 只能共享一个 Variable 选项,并且需要设置不同的 Value 选项值。

【例 3-8】 利用 Radiobutton 组件创建单选按钮组。

```
import tkinter as tk
root = tk.Tk()
v = tk.IntVar()
v.set(1)
# 如果 set 的括号里是 1 的话,一开始的默认选项在 1;如果括号里是 2 的话,默认选项在 2;如果超过了
# 4,则没有默认选项
langs = [("One",1),("Two",2),("Three",3)]
for lang,num in langs:
    tk.Radiobutton(root,text = lang,variable = v,value = num).pack(anchor = tk.W)
tk.mainloop()
```

运行程序,效果如图 3-8 所示。

图 3-8 是一个单选按钮样式,如果将它的 indicatoron 选项设置为 False,Radiobutton 的样式就会变成普通按钮的样式了,如图 3-9 所示。

```
#在 for 循环中进行改动
for text in texts:
    tk.Radiobutton(root,text = text,variable = v,value = i,indicatoron = False).pack(fill = tk.X)
    i += 1
```

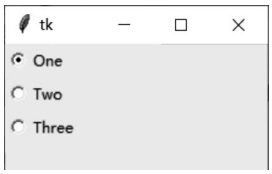

图 3-8　单选按钮

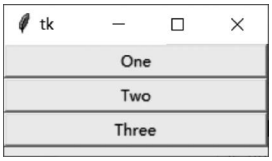

图 3-9　改成普通按钮形式

3.2.6　Listbox 和 Combobox 组件

Listbox 代表一个列表框，用户可通过列表框来选择一个列表项。ttk 模块下的 Combobox 则是 Listbox 的改进版，它既提供了单行文本框让用户直接输入（就像 Entry 一样），也提供了下拉列表框供用户选择（就像 Listbox 一样），因此它被称为复合框。

程序创建 Listbox 需要以下两步。

（1）创建 Listbox 对象，并为之执行各种选项。Listbox 除支持大部分通用选项之外，还支持 selectmode 选项，用于设置 Listbox 的选择模式。

（2）调用 Listbox 的 insert(self,index, * elements)方法来添加选项。从最后一个参数可以看出，该方法既可每次添加一个选项，也可传入多个参数，每次添加多个选项。index 参数指定选项的插入位置，它支持 END（结尾处）、ANCHOR（当前位置）和 ACTIVE（选中处）等特殊索引。

Listbox 的 selectmode 支持的选择模式有如下几种。

（1）'browse'：单选模式，支持按住鼠标键拖动来改变选择。

（2）'multiple'：多边模式。

（3）'single'：单边模式，必须通过鼠标键单击来改变选择。

（4）'extended'：扩展的多边模式，必须通过 Ctrl 或 Shift 键辅助实现多选。

【例 3-9】　利用 Listbox 和 Combobox 组件创建界面。

```
#用户界面
import os
from tkinter import *
from tkinter import ttk

root = Tk()
root.title("window")
root.geometry('500x500')
#创建标签
var1 = StringVar()
l = Label(root, bg = 'green', fg = 'pink',font = ('Arial', 12), width = 10, textvariable = var1)
l.pack()
#列表框单击事件
def print_lb1():
    value = lb1.get(lb1.curselection())
    var1.set(value)
#列表框单击按钮
b1 = Button(root,text = '打印选择的事件 lb',width = 18,height = 2,command = print_lb1)
```

```
b1.pack()
#创建 Listbox
var_lb1 = StringVar()
var_lb1.set(('C30','C35','C40'))
lb1 = Listbox(root,listvariable = var_lb1)
lb1.pack()

#组合框单击事件
def print_cb1():
    value = cb1.get()
    var1.set(value)
#组合框单击按钮
b2 = Button(root,text = '打印选择的事件 cb',width = 18,height = 2,command = print_cb1)
b2.pack()
#创建 Combobox
var_cb1 = StringVar()
var_cb1.set('请选择混凝土标号')
cb1 = ttk.Combobox(root,textvariable = var_cb1)
cb1['values'] = ['C30','C35','C40']
cb1.pack()
#事件循环
root.mainloop()
```

运行程序,效果如图 3-10 所示。

图 3-10　Listbox 和 Combobox 组件界面

3.2.7　Spinbox 组件

Spinbox 组件是一个带有两个小箭头的文本框,用户既可以通过两个小箭头上下调整该组件内的值,也可以直接在文本框内输入内容作为该组件的值。

Spinbox 本质上也相当于持有一个列表框,这一点类似于 Combobox,但 Spinbox 不会展开下拉列表供用户选择。Spinbox 只能通过向上、向下箭头来选择不同的选项。

在使用 Spinbox 组件时,既可通过 from(由于 from 是关键字,实际使用时写成 from_)、to、increment 选项来指定选项列表,也可通过 values 选项来指定多个列表项,该选项的值可以是 list 或 tuple。

Spinbox 同样可通过 textvariable 选项将它与指定变量绑定,这样程序即可通过该变量来获取或修改 Spinbox 组件的值。

Spinbox 还可通过 command 选项指定事件处理函数或方法：当用户单击 Spinbox 的向上、向下箭头时，程序就会触发 command 选项指定的事件处理函数或方法。

【例 3-10】 Spinbox 组件创建界面。

```
'''
打印 Spinbox 的当前内容
get:此方法取得当前显示的内容
'''
from tkinter import *
root = Tk()
def printSpin():
    #使用 get()方法得到当前的显示值
    print(sb.get())
sb = Spinbox(root, from_ = 0, to = 10, command = printSpin)
sb.pack()
root.mainloop()
#每次单击 Spinbox 按钮时就会调用 printSpin()函数,打印出 Spinbox
#的当前值
```

运行程序，效果如图 3-11 所示。

图 3-11　Spinbox 组件界面

3.2.8　Scale 和 LabeledScale 组件

Scale 组件代表一个滑动条，可以为该滑动条设置最小值和最大值，也可以设置滑动条每次调节的步长。Scale 组件支持如下选项。

- from：设置该 Scale 的最小值。
- to：设置该 Scale 的最大值。
- resolution：设置该 Scale 滑动时的步长。
- label：为 Scale 组件设置标签内容。
- length：设置轨道的长度。
- width：设置轨道的宽度。
- troughcolor：设置轨道的背景色。
- sliderlength：设置轨道的长度。
- sliderrelief：设置滑块的立体样式。
- showvalue：设置是否显示当前值。
- orient：设置方向。该选项支持 vertical 和 horizontal 两个值。
- digits：设置有效数字至少要有几位。
- variable：用于与变量进行绑定。
- command：用于为该 Scale 组件绑定事件处理、函数或方法。

Scale 组件同样支持 variable 进行变量绑定，也支持使用 command 选项绑定事件处理函数或方法，这样每当用户拖动滑动条上的滑块时，都会触发 command 绑定的事件处理方法，不过 Scale 的事件处理方法除外，它可以额外定义一个参数，用于获取 Scale 的当前值。

而对于 LabeledScale 组件，如果使用 ttk.Scale 组件，则更接近操作系统本地的效果，但允许定制的选项少。ttk.LabeledScale 是平台化的滑动条，因此它允许设置的选项很少，只能设置 from、to 和 compound 等有限的几个选项，而且它总是生成一个水平滑动条（不能变成垂直的），其中，compound 选项控制滑动条的数值标签是显示在滑动条的上方，还是滑动条的下方。

【例 3-11】 利用 Scale 和 LabeledScale 组件创建界面。

```
from tkinter import *
root = Tk()
root.resizable(0,0)

def change(value):
    print(value, scale.get(), doubleVar.get())
'''获取 Scale 值
1.通过事件处理方法的参数来获取.
2.通过 Scale 组件提供的 get()方法来获取.
3.通过 Scale 组件绑定的变量来获取.
'''
doubleVar = DoubleVar()
scale = Scale(root,
              from_ = -100,                    #设置最小值
              to = 100,                        #设置最大值
              resolution = 5,                  #设置步长
              label = 'Sacle:',                #设置标签内容
              length = 400,                    #设置轨道的长度
              width = 30,                      #设置轨道的宽度
              troughcolor = 'lightblue',       #设置轨道的背景色
              sliderlength = 20,               #设置滑块的长度
              sliderrelief = SUNKEN,           #设置滑块的立体样式
              showvalue = YES,                 #设置显示当前值
              orient = HORIZONTAL,             #设置水平方向
              digits = 5,                      #设置五位有效数字
              command = change,                #绑定事件处理函数
              variable = doubleVar             #绑定变量
        )
scale.pack()
scale.set(20)
f = Frame(root)
f.pack(fill = X, expand = YES, padx = 10)

def valueshow():
    scale['showvalue'] = showVar.get()
Label(f, text = '是否显示值:').pack(side = LEFT)
showVar = IntVar()
i = 0
for s in ('不显示', '显示'):
    Radiobutton(f, text = s, value = i,variable = showVar,command = valueshow).pack(side = LEFT)
    i += 1
showVar.set(1)
f = Frame(root)
f.pack(fill = X, expand = YES, padx = 10)

def scaleorient():
    scale['orient'] = VERTICAL if orientVar.get() else HORIZONTAL
Label(f, text = '方向:').pack(side = LEFT)
orientVar = IntVar()
i = 0
for direction in ('水平', '垂直'):
    Radiobutton(f, text = direction, value = i,variable = orientVar,command = scaleorient).pack
(side = LEFT)
    i += 1
orientVar.set(0)
```

```
from tkinter import ttk
f = Frame(root)
f.pack(fill = X, expand = YES, padx = 10)
Label(f, text = 'LabeledScale:').pack(anchor = NW)
labeledscale = ttk.LabeledScale(f,
                                from_ = -100,        #设置最小值
                                to = 100,            #设置最大值
                                compound = BOTTOM    #显示数值在滑动条下方
                                )
labeledscale.value = -20
labeledscale.pack(fill = X, expand = YES)
mainloop()
```

运行程序,效果如图 3-12 所示。

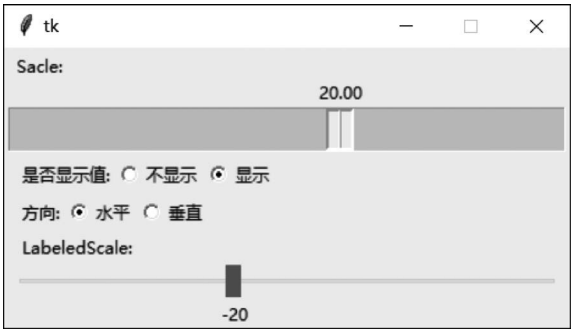

图 3-12　Scale 和 LabeledScale 组件界面图

3.2.9　LabelFrame 组件

LabelFrame 组件是 Frame 组件的变体。默认情况下,LabelFrame 会在其子组件的周围绘制一个边框以及一个标题。当想要将一些相关的组件分为一组的时候,可以使用 LabelFrame 组件,如一系列 Radiobutton(单选按钮)组件。

【例 3-12】 创建两个 LabelFrame 组件,并为其添加内容。

```
import tkinter as tk

win = tk.Tk()
#定义第一个容器
frame_left = tk.LabelFrame(win, text = "优点", labelanchor = "n")
frame_left.place(relx = 0.2, rely = 0.2, relwidth = 0.3, relheight = 0.6)
label_1 = tk.Label(frame_left, text = "简单")
label_1.place(relx = 0.2, rely = 0.2)
label_2 = tk.Label(frame_left, text = "易学")
label_2.place(relx = 0.6, rely = 0.2)
label_3 = tk.Label(frame_left, text = "易读")
label_3.place(relx = 0.2, rely = 0.6)
label_4 = tk.Label(frame_left, text = "易维护")
label_4.place(relx = 0.6, rely = 0.6)

#定义第二个容器
frame_right = tk.LabelFrame(win, text = "缺点", labelanchor = "n")
frame_right.place(relx = 0.5, rely = 0.2, relwidth = 0.3, relheight = 0.6)
label_1 = tk.Label(frame_right, text = "速度慢")
label_1.place(relx = 0.2, rely = 0.2)
```

```
label_2 = tk.Label(frame_right, text = "强制缩进")
label_2.place(relx = 0.6, rely = 0.2)
label_3 = tk.Label(frame_right, text = "单行语句")
label_3.place(relx = 0.2, rely = 0.6)
label_4 = tk.Label(frame_right, text = "不加密")
label_4.place(relx = 0.6, rely = 0.6)
win.mainloop()
```

运行程序,效果如图 3-13 所示。

图 3-13 LabelFrame 组件创建界面

3.2.10 PanedWindow 组件

当我们需要提供一个可供用户调整的多空间框架的时候,可以使用 PanedWindow 组件。PanedWindow 组件会为每一个子组件生成一个独立的窗格,用户可以自由调整窗格的大小。

【例 3-13】 使用 PanedWindow 组件创建三个窗格。

```
import tkinter as tk

window = tk.Tk()
window.title('你好 Tkinter')
height = window.winfo_screenheight()
width = window.winfo_screenwidth()
window.geometry('400x300 + %d + %d' %((width - 400)/2,(height - 300)/2))
#创建三个窗口,在窗口上可以拖动鼠标调整大小
m1 = tk.PanedWindow(bd = 1,bg = 'blue')
m1.pack(fill = "both", expand = 1)

left = tk.Label(m1, text = "左窗格")
m1.add(left)
m2 = tk.PanedWindow(orient = "vertical",bd = 1,bg = 'red')
m1.add(m2)
top = tk.Label(m2, text = "顶部窗格")
m2.add(top)
bottom = tk.Label(m2, text = "底部窗格")
m2.add(bottom)
window.mainloop()
```

运行程序,效果如图 3-14 所示。

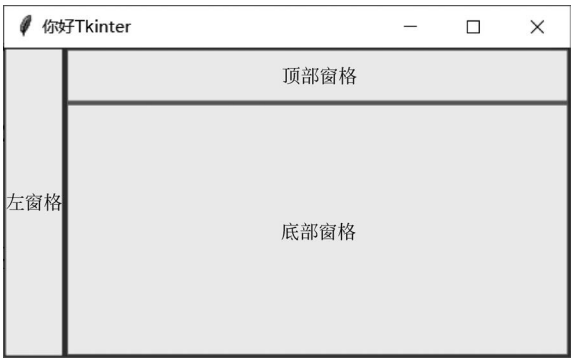

图 3-14　PanedWindow 组件创建界面

3.2.11　OptionMenu 组件

OptionMenu 组件用于构建一个带菜单的按钮，该菜单可以在按钮的四个方向上展开，展开方向可通过 direction 选项控制。

使用 OptionMenu 比较简单，直接调用它的如下构造函数即可。

```
_init_(self,master,variable,value, * values, ** kwargs)
```

其中，master 参数的作用与所有的 Tkinter 组件一样，指定将该组件放入哪个容器中。其他参数的含义如下。

- variable：指定该按钮上的菜单与哪个变量绑定。
- value：指定默认选择菜单中的哪一项。
- values：Tkinter 将收集为此参数传入的多个值，为每个值创建一个菜单项。
- kwargs：用于为 OptionMenu 配置选项。除前面介绍的选项之外，还可通过 direction 选项控制菜单的展开方向。

【例 3-14】　利用 OptionMenu 创建菜单。

```
from tkinter import *
from tkinter import messagebox

class Application(Frame):
    def __init__(self,master = None):
        super().__init__(master)
        self.master = master
        self.pack()
        self.createwidget()

    def createwidget(self):
        option = ["青菜", "白菜", "菠菜", "黄瓜"]                    #所有的选项列表
        variable = StringVar()
        variable.set(option[0])                                      #默认选项
        op = OptionMenu(self, variable, * option)
        op["width"] = 10                                             #选项框的长度
        op.pack(padx = 10, pady = 10)                                #选项框的位置
        #设置按钮，按下后弹出窗口，提示选的蔬菜名称，即获取变量内容
        Button(self, text = "确定",command = lambda : self.print_fruit(variable)).pack(pady = 20)

    def print_fruit(self,variable):
        messagebox.showinfo("蔬菜", "您选的蔬菜是:{}".format(variable.get()))
```

```
if __name__ == '__main__':
    root = Tk()
    root.title("optionmenu 测试")
    root.geometry("300x200")
    app = Application(root)
    root.mainloop()
```

运行程序，选择其中一个菜单并单击“确定”按钮，得到如图 3-15 所示的效果。

图 3-15　OptionMenu 组件创建菜单按钮

3.3　对话框

对话框也是图形界面编程中很常用的组件，通常用于向用户生成某种提示信息，或者请求用户输入某些简单的信息。

对话框看上去有点类似于顶级窗口，但对于对话框有如下两点需要注意。

(1) 对话框通常依赖其他窗口，因此程序在创建对话框时同样需要指定 master 属性(该对话框的属主窗口)。

(2) 对话框有非模式(non-modal)和模式(modal)两种，当某个模式对话框被打开后，该模式对话框总是位于它依赖的窗口之上；在模式对话框被关闭前，它依赖的窗口无法获得焦点。

3.3.1　普通对话框

Tkinter 在 simpledialog 和 dialog 模块下分别提供了 SimpleDialog 类和 Dialog 类，它们都可作为普通对话框使用，而且用法也差不多。

在使用 simpledialog.SimpleDialog 创建对话框时，可指定如下选项。

- title：指定该对话框的标题。
- text：指定该对话框的内容。
- button：指定该对话框下方的几个按钮。
- default：指定该对话框中默认第几个按钮得到焦点。
- cancel：指定当用户通过对话框右上角的×按钮关闭对话框时，该对话框的返回值。

如果使用 dialog.Dialog 创建对话框，除可使用 master 指定对话框的属主窗口之外，还可通过 dict 来指定如下选项。

- title：指定该对话框的标题。
- text：指定该对话框的内容。
- strings：指定该对话框下方的几个按钮。
- default：指定该对话框中默认第几个按钮得到焦点。

- bitmap：指定该对话框上的图标。

【例 3-15】　分别使用 SimpleDialog 和 Dialog 来创建对话框。

```
from tkinter import *
from tkinter import simpledialog
from tkinter import dialog

root = Tk()
root.title('普通对话框')
root.geometry("250x250+30+30")
msg = '先定义简单对话框的显示文本,再定义简单对话框函数,最后创建一个按钮引用此函数'
def open_simpledialog():
    d = simpledialog.SimpleDialog(root,title='Simpledialog',text=msg,buttons=["确定","取消","退出"],default=0,cancel=3)
    #获取用户单击对话框的哪个按钮或关闭对话框返回 cancel 指定的值
    print(d.go())
Button(root,text='打开 Simpledialog',command=open_simpledialog).pack(side='left',ipadx=10,padx=10)
def open_dialog():
    d = dialog.Dialog(root, {'title': 'Dialog', 'text': msg,'strings': ('确定','取消','退出'),'default':0,
                             'bitmap': 'question'})
    #打印该对话框 num 属性的值,该返回值会获取用户单击了对话框的哪个按钮
    print(d.num)
Button(root,text='打开 Dialog',command=open_dialog).pack(side='left',ipadx=10,padx=10)
root.mainloop()
```

运行程序，单击界面上的按钮，得到如图 3-16 所示的效果。

图 3-16　普通对话框

在如图 3-16 所示的 Dialog 对话框中的左侧还显示了一个问号图标，这是 Python 内置的 10 个位图之一，可以直接使用。共有如下几个常量可用于设置位图："error""gray75""gray50""gray25""gray12""hourglass""info""questhead""question""warning"。

3.3.2　非模式对话框

对话框分为模式对话框和非模式对话框。例如，常用的"登录"和"注册"对话框就是模式对话框。从主窗口打开非模式对话框后，不关闭对话框，也能操作主窗口。例如，文本编辑器中的"查找和替换"对话框就是非模式对话框，查找一般从光标处开始，如完成查找后，希望从开始再查找一次，必须在主窗口中将光标移到开始处。

Python Tkinter 中定义了多个模式对话框类，包括 messagebox 类（通用消息对话框类）、filedialog 类（文件对话框类）、colorchooser 类（颜色选择对话框类）、simpledialog 类（简单对话框类）和 dialog 类（对话框类）。但这些类只能满足比较简单的应用，像“登录”和“注册”对话框这样的对话框，需要输入用户名和密码，并要对输入的数据格式做检查，例如，注册时密码要求多少位、是否要求包括数字和字符、用户名是否唯一等。登录时要确定密码是否正确，若不正确则要求重新输入。对这些比较复杂的对话框，要使用 Toplevel 类生成自定义对话框。Toplevel 类可以在主窗体外创建一个独立的窗口，它和用 Tk()方法创建出来的主窗口一样有标题栏、边框等部件，它们有相似的方法和属性。在 Toplevel 窗口中，能像主窗口一样放入 Button、Label 和 Entry 等组件。Toplevel 窗口和主窗口可以互相使用对方的变量和方法。一般用于创建自定义模式和非模式对话框。

【例 3-16】 用 Toplevel 类创建的对话框默认是非模式对话框。

```
import tkinter as tk
def openDialog():
    global f1,e1                    #在 Toplevel 窗口和主窗口可以互相使用对方的变量和方法
    f1 = tk.Toplevel(root)          #用 Toplevel 类创建独立主窗口的新窗口
    f1.grab_set()                   #将 f1 设置为模式对话框,f1 不关闭无法操作主窗口
    #f1.transient(root) #该函数使 f1 总是在父窗口前边,如父窗口最小化,f1 被隐藏.模式对话
                        #框不用使用这条语句
    e1 = tk.Entry(f1)               #可在 e1 中输入数据,单击“确定”按钮将数据显示在主窗口 label1
    e1.pack()
    b1 = tk.Button(f1,text = '确定',command = showInput)
    b1.pack()
def showInput():                    #在此函数中,可检查数据格式是否正确
    label1['text'] = e1.get()       #显示 e1 中输入的数据
    f1.destroy()                    #关闭对话框
root = tk.Tk()
root.geometry('200x200 + 50 + 50')
tk.Button(root, text = "打开模式对话框", command = openDialog).pack()
label1 = tk.Label(root,text = '初始字符')
label1.pack()
root.mainloop()
```

运行程序，效果如图 3-17 所示。

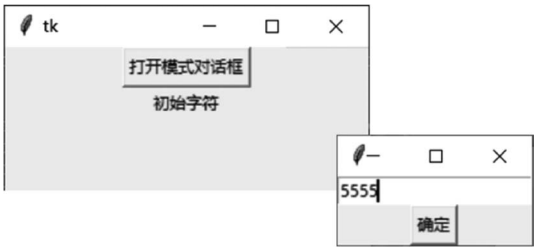

图 3-17 非模式对话框

3.3.3 输入对话框

Python 的 Tkinter 模块中，有一个子模块 simpledialog，这个子模块中包含三个函数：askinteger()、askfloat()、askstring()。它们通过 GUI 窗口的方式，让用户输入一个整数、浮点数或者字符串，并且自带输入合法性检测，使用非常方便。

1. askinteger()函数

通过对话框，让用户输入一个整数：

```
import tkinter as tk
from tkinter.simpledialog import askinteger,askfloat,askstring
root = tk.Tk() #需要一个根窗口来显示下面的示例
print(askinteger('askinteger','请输入一个整数:'))
```

代码中的askinteger()函数的第1个参数是对话框的Title，第2个参数是输入条上面的一行信息，如图3-18所示。

在图3-18中输入整数，单击OK按钮，返回整数；单击Cancel按钮，直接关闭此对话框，返回None。如果输入非法数据，会有如图3-19所示的对话框弹出。

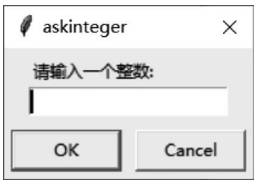

图3-18　输入整数对话框

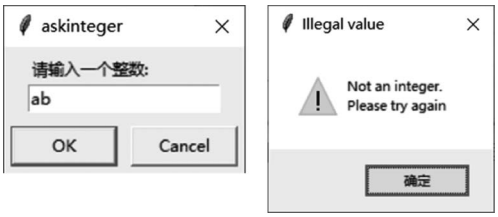

图3-19　自带输入合法性检查

askinteger()函数还支持设置初始值，设置可以接收的最大值和最小值，这极大地方便了应用的开发。

```
askinteger('askinteger','please give me an integer:',
                    initialvalue = 12345, minvalue = 100, maxvalue = 20000)
```

2. askfloat()函数

这是一个接收浮点数的对话框：

```
print(askfloat('askfloat','请输入一个浮点数:'))
```

运行程序，在弹出的如图3-20所示的对话框中输入小数(浮点数)，单击OK按钮，返回小数；单击Cancel按钮，直接关闭此对话框，返回None。同样自带输入合法性检查。但是，askfloat()函数可以接收一个整数，返回的是此整数对应的float。

跟askinteger()函数一样，askfloat()函数也支持设置初始值、最大值和最小值。

3. askstring()函数

接收字符串的对话框，这个对话框可以接收的数据就很广了，输入整数或浮点数，都会被当作字符串返回。

```
print(askstring('askstring','请输入一个字符串:'))
```

运行程序，效果如图3-21所示。

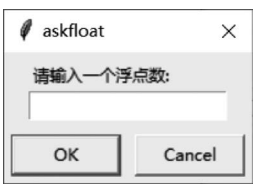

图3-20　浮点数对话框

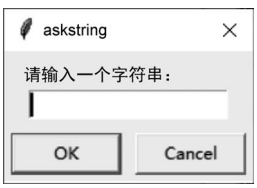

图3-21　字符串对话框

输入有效浮点数后，单击 OK 按钮即完成输入。

单击 Cancel 按钮和直接关闭对话框，都是返回 None。

askstring()函数也支持设置初始值、最大值和最小值，不过要注意，askstring()函数接收的是字符串，内部比较大小，也是字符串之间比较大小。

3.3.4 生成对话框

在 filedialog 模块下提供了各种用于生成文件对话框的工具函数。这些工具函数有些返回用户所选择文件的路径，有些直接返回用户所选择文件的输入/输出流。

- askopenfilename(** options)：返回打开的文件名。
- askopenfilenames(** options)：返回打开的多个文件名列表。
- askopenfile(** options)：返回打开的文件对象。
- askopenfiles(** options)：返回打开的文件对象的列表。
- askdirectory(** options)：返回目录名。
- asksaveasfile(** options)：返回保存的文件对象。
- asksaveasfilename(** options)：返回保存的文件名。

上面用于生成打开文件的对话框的工具函数支持如下选项。

- defaultextension 指定默认扩展名。当用户没有输入扩展名时，系统会默认添加该选项指定的扩展名。
- filetypes：指定在该文件对话框中能查看的文件类型。该选项值是一个序列，可指定多个文件类型。可以通过“*”指定浏览所有文件。
- initialdir：指定初始打开的目录。
- initialfile：指定所选择的文件。
- parent：指定该对话框的属主窗口。
- title：指定对话框的标题。
- multiple：指定是否允许多选。

对于打开目录的对话框，还额外支持一个 mustexist 选项，该选项指定是否只允许打开已存在的目录。

【例 3-17】 测试文件对话框。

```
from tkinter import *
from tkinter.filedialog import *
root = Tk()
root.geometry('400x300')

def test01():
    f1 = askopenfilename(title='上传文件',
                         initialdir='c:file',
                         filetype=[('文本文件', '.txt')])
    file01['text'] = f1
Button(root, text='选择文件', command=test01).pack()
file01 = Label(root, width=40, height=3, bg='pink')
file01.pack()
root.mainloop()
```

运行程序，效果如图 3-22 所示。

图 3-22　打开文件对话框

3.3.5　颜色选择对话框

在 colorchooser 模块下提供了用于生成颜色选择对话框的 askcolor()函数，可为该函数指定如下选项。

- parent：指定该对话框的属主窗口。
- title：指定该对话框的标题。
- color：指定该对话框初始选择的颜色。

【例 3-18】　演示颜色选择对话框的用法。

```
from tkinter import *
import tkinter.colorchooser as cc
main = Tk()
def CallColor():
    Color = cc.askcolor()
    print(Color)
Button(main,text = "选择颜色",command = CallColor).grid()
mainloop()
```

运行程序，单击界面上的“选择颜色”按钮，将可以看到如图 3-23 所示的对话框。

图 3-23　颜色选择对话框

当用户选择指定颜色，并单击颜色选择对话框中的“确定”按钮后，askcolor()函数会返回用户所选择的颜色，因此可以在控制台看到用户所选择的颜色。

3.3.6 消息框

在 messagebox 模块下提示了大量工具函数用来生成各种消息框，在默认情况下，开发者在调用 messagebox 的工具箱函数时只要设置提示区的字符串即可，图标区的图标、按钮区的按钮都有默认设置。如果有必要，则完全可通过如下两个选项来定制图标和按钮。

- icon：定制图标的选项。该选项支持“error”“info”“question”“waring”这几个选项值。
- type：定制按钮的选项。该选项支持“abortretryignore”（取消、重试、忽略）、“ok”（确定）、“okcancel”（确定、取消）、“retrycancel”（重试、取消）、“yesno”（是、否）、“yesnocancel”（是、否、取消）这些选项。

【例 3-19】 演示 messagebox 工具的用法。

```
import tkinter as tk
import tkinter.messagebox
window = tk.Tk()
window.title('消息框')
window.geometry('200x200')
def hit_me():
    tk.messagebox.showinfo(title = 'Hi',message = '显示消息框信息')
    tk.messagebox.showerror(title = 'Hi',message = '提示消息框错误')
    tk.messagebox.showwarning(title = 'Hi', message = '提示消息框警告')
    print(tk.messagebox.askokcancel(title = 'Hi',message = '是、取消'))
    print(tk.messagebox.askquestion(title = 'Hi', message = '确定'))
    print(tk.messagebox.askyesno(title = 'Hi', message = '是、否'))
    print(tk.messagebox.askretrycancel(title = 'Hi', message = '重试、取消'))
    print(tk.messagebox.askyesnocancel(title = 'Hi', message = '是、否、取消'))
tk.Button(window,text = '请单击',command = hit_me).pack()
window.mainloop()
```

运行程序，得到相应的对话框界面，如图 3-24 所示。

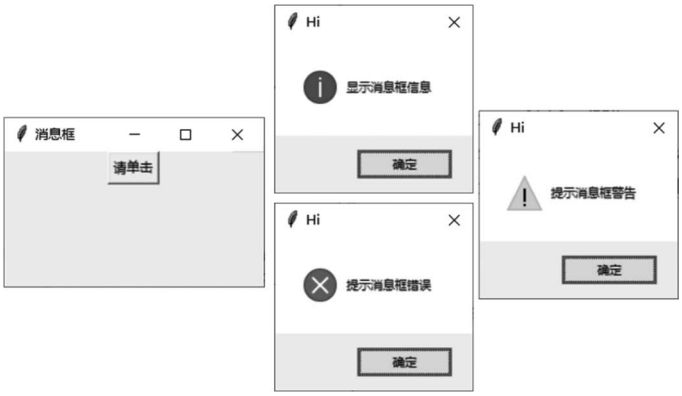

图 3-24 各种消息框

3.4 菜单

Tkinter 为菜单提供了 Menu 类。该类既可代表菜单条，也可代表菜单，还可代表上下文菜单（右键菜单）。简单来说，Menu 类就可以搞定所有菜单相关内容。

- add_command()：添加菜单项。
- add_checkbutton()：添加复选框菜单项。

- add_radiobutton()：添加单选按钮菜单。
- add_separator()：添加菜单分隔条。

上面的前三个方法都用于添加菜单项，因此都支持如下常用选项。

- label：指定菜单项的文本。
- command：指定为菜单项绑定的事件处理方法。
- image：指定菜单项的图标。
- compound：指定在菜单项中图标位于文字的哪个方位。

菜单有两种用法，分别为在窗口上方通过菜单条管理菜单、通过鼠标右键触发右键菜单（上下文菜单）。

3.4.1　窗口菜单

Menu（菜单）组件通常被用于实现应用程序上的各种菜单，由于该组件是底层代码实现，所以不建议自行通过按钮和其他组件来实现菜单功能。

【例 3-20】 创建一个顶级菜单，需要先创建一个菜单实例，然后使用 add()方法将命令和其他子菜单添加进去。

```
import tkinter as tk

root = tk.Tk()
def callback():
          print("～被调用～")
#创建一个顶级菜单
menubar = tk.Menu(root)
menubar.add_command(label = "文件", command = callback)
menubar.add_command(label = "退出", command = root.quit)
#显示菜单
root.config(menu = menubar)
root.mainloop()
```

运行程序，效果如图 3-25 所示。

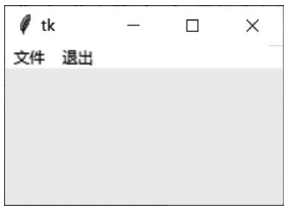

图 3-25　顶级菜单

如果要创建一个下拉菜单（或者其他子菜单），方法与例 3-20 的大同小异，最主要的区别是它们最后需要添加到主菜单上（而不是窗口上），例如：

```
import tkinter as tk

root = tk.Tk()
def callback():
     print("～被调用～")
#创建一个顶级菜单
menubar = tk.Menu(root)
#创建一个下拉菜单"文件",然后将它添加到顶级菜单中
filemenu = tk.Menu(menubar, tearoff = False)
filemenu.add_command(label = "打开", command = callback)
filemenu.add_command(label = "保存", command = callback)
filemenu.add_separator()
filemenu.add_command(label = "退出", command = root.quit)
menubar.add_cascade(label = "文件", menu = filemenu)
```

```
#创建另一个下拉菜单"编辑",然后将它添加到顶级菜单中
editmenu = tk.Menu(menubar, tearoff = False)
editmenu.add_command(label = "剪切", command = callback)
editmenu.add_command(label = "复制", command = callback)
editmenu.add_command(label = "粘贴", command = callback)
menubar.add_cascade(label = "编辑", menu = editmenu)
#显示菜单
root.config(menu = menubar)
root.mainloop()
```

运行程序,效果如图 3-26 所示。

图 3-26　下拉菜单

3.4.2　右键菜单

实现右键菜单很简单,程序只要先创建菜单,然后为目标组件的右键单击事件绑定处理函数,当用户单击鼠标右键时,调用菜单的 post()方法即可在指定位置弹出右键菜单。

【例 3-21】　创建弹出式菜单。

```
import tkinter
def makeLabel():
    global baseFrame
    tkinter.Label(baseFrame, text = "Python 简单、易用、易维护").pack()

baseFrame = tkinter.Tk()
menubar = tkinter.Menu(baseFrame)
for x in ['平等', '自由', '民主']:
    menubar.add_separator()
    menubar.add_command(label = x)
menubar.add_command(label = "和谐", command = makeLabel)
#事件处理函数一定要至少有一个参数,且第一个参数表示的是系统事件
def pop(event):
    #注意使用 event.x 和 event.x_root 的区别
    menubar.post(event.x_root, event.y_root)
baseFrame.bind("<Button-3>", pop)
baseFrame.mainloop()
```

运行程序,效果如图 3-27 所示。

图 3-27　右键菜单

3.5　在 Canvas 中绘图

Canvas 为 Tkinter 提供了绘图功能,其提供的图形组件包括线形、圆形、图片甚至其他控件。Canvas 控件为绘制图形图表、编辑图形、自定义控件提供了可能。

3.5.1　Canvas 的绘图

Canvas 组件的用法与其他 GUI 组件一样简单，程序只要创建并添加 Canvas 组件，然后调用该组件的方法来绘制图形即可。

【例 3-22】　演示最简单的 Canvas 绘图。

```
from tkinter import *
#创建窗口
root = Tk()
#创建并添加 Canvas
cv = Canvas(root, background = 'white')
cv.pack(fill = BOTH, expand = YES)
cv.create_rectangle(30, 30, 200, 200,
outline = 'yellow',                #边框颜色
stipple = 'question',              #填充的位图
fill = "red",                      #填充颜色
width = 5                          #边框宽度
)
cv.create_oval(250, 32, 340, 210,
outline = 'red',                   #边框颜色
fill = 'pink',                     #填充颜色
width = 4                          #边框宽度
)
root.mainloop()
```

运行程序，效果如图 3-28 所示。

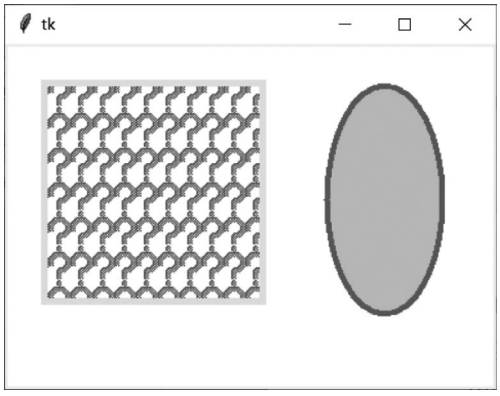

图 3-28　简单的 Canvas 绘图

从上面的程序可看出，Canvas 提供了 create_rectangle()方法绘制矩形和 create_oval()方法绘制椭圆(包括圆，圆是椭圆的特例)。实际上，Canvas 还提供了如下方法来绘制各种图形。

- create_arc：绘制弧。
- create_bitmap：绘制位图。
- create_image：绘制图片。
- create_line()：绘制直线。
- create_polygon：绘制多边形。
- create_text：绘制文字。
- create_window：绘制组件。

Canvas 的坐标系统是绘图的基础，其中，点(0,0)位于 Canvas 组件的左上角，X 轴水平向

右延伸，Y 轴垂直向下延伸。

绘制上面这些图形时需要简单的几何基础。

- 在使用 create_line()绘制直线时，需要指定两个点的坐标，分别作为直线的起点和终点。
- 在使用 create_rectangle()绘制矩形时，需要指定两个点的坐标，分别作为矩形左上角点和右下角点的坐标。
- 在使用 create_oval()绘制椭圆时，需要指定两个点的坐标，分别作为左上角点和右下角点的坐标来确定一个矩形，而该方法则负责绘制该矩形的内切椭圆，如图 3-29 所示。

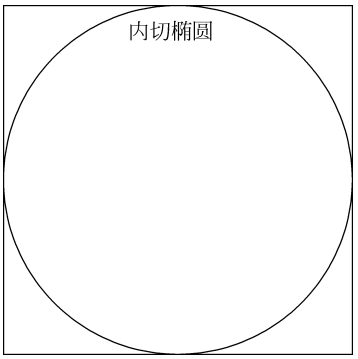

图 3-29　内切椭圆

从图 3-29 可以看出，只要矩形确定下来，该矩形的内切椭圆就能确定下来，而 create_oval()方法所需要的两个坐标正是用于指定该矩形的左上角点和右下角点的坐标。

- 在使用 create_arc()绘制弧时，和 create_oval()的用法相似，因为弧是椭圆的一部分，因此同样也是指定左上角和右下角两个点的坐标，默认总是绘制从 3 点(0)开始，逆时针旋转 90°的那一段弧。程序可通过 start 改变起始角度，也可通过 extent 改变转过的角度。
- 在使用 create_polygon()绘制多边形时，需要指定多个点的坐标来作为多边形的多个定点。
- 在使用 create_bitmap()、create_image()、create_text()、create_window()等方法时，只要指定一个坐标点，用于指定目标元素的绘制位置即可。

在绘制这些图形时可指定如下选项。

- fill：指定填充颜色(不指定，默认不填充)。
- outline：指定边框颜色。
- width：指定边框宽度，边框宽度默认为 1。
- dash：指定边框使用虚线。该属性值既可为单独的整数，用于指定虚线中线段的长度；也可为形如(4,1,3)格式的元素，此时 4 指定虚线中线段的长度，1 指定间隔长度，3 指定虚线长度。
- stipple：使用位图平铺进行填充。该选项可与 fill 选项结合使用。
- style：指定绘制弧的样式(仅对 create_arc()方法起作用)。支持 PIESLICE(扇形)、CHORD(弓形)、ARC(仅绘制弧)选项值。
- start：指定绘制弧的起始角度(仅对 create_arc()方法起作用)。
- extent：指定绘制弧的角度(仅对 create_arc()方法起作用)。
- arrow：指定绘制直线时两端是否有箭头。支持 NONE(两端无箭头)、FIRST(开始端有箭头)、LAST(结束端有箭头)、BOTH(两端都有箭头)选项值。
- arrowshape：指定箭头形状。该选项是一个形如“30 20 10”的字符串，字符串中的三个整数依次指定填充长度、箭头长度、箭头宽度。
- joinstyle：指定直接连接点的风格。仅对绘制直线和多边形有效。支持 METTER、ROUND、BEVEL 选项值。
- anchor：指定绘制文字、GUI 组件的位置。仅对 create_text()、create_window()方法有效。

- justify：指定文字的对齐方式(仅对 create_text()方法有效)。支持 CENTER、LEFT、RIGHT 常量值。

【例 3-23】 通过不同的方法来绘制不同的图形，这些图形分别使用不同的边框、不同的填充效果。

```
from tkinter import *

#创建窗口
root = Tk()
root.title('绘制图形项')
#创建并添加 Canvas
cv = Canvas(root, background = 'white', width = 830, height = 830)
cv.pack(fill = BOTH, expand = YES)
columnFont = ('黑体', 17)
titleFont = ('黑体', 19, 'bold')
#使用循环绘制文字
for i, st in enumerate(['默认', '指定边宽', '指定填充', '边框颜色', '位图填充']):
    cv.create_text((130 + i * 140, 20),text = st,
    font = columnFont,
    fill = 'gray',
    anchor = W,
    justify = LEFT)
#绘制文字
cv.create_text(10, 60, text = '绘制矩形',
    font = titleFont,
    fill = 'magenta',
    anchor = W,
    justify = LEFT)
#定义列表,每个元素的 4 个值分别指定边框宽度、填充色、边框颜色、位图填充
options = [(None, None, None, None),
    (4, None, None, None),
    (4, 'pink', None, None),
    (4, 'pink', 'green', None),
    (4, 'pink', 'green', 'error')]
#采用循环绘制 5 个矩形
for i, op in enumerate(options):
    cv.create_rectangle(130 + i * 140, 50, 240 + i * 140, 120,
        width = op[0],              #边框宽度
        fill = op[1],               #填充颜色
        outline = op[2],            #边框颜色
        stipple = op[3])            #使用位图填充
#绘制文字
cv.create_text(10, 160, text = '绘制椭圆',
    font = titleFont,
    fill = 'magenta',
    anchor = W,
    justify = LEFT)
#定义列表,每个元素的 4 个值分别指定边框宽度、填充色、边框颜色、位图填充
options = [(None, None, None, None),
    (4, None, None, None),
    (4, 'pink', None, None),
    (4, 'pink', 'green', None),
    (4, 'pink', 'green', 'error')]
#采用循环绘制 5 个椭圆
for i, op in enumerate(options):
    cv.create_oval(130 + i * 140, 150, 240 + i * 140, 220,
```

```
            width = op[0],              #边框宽度
            fill = op[1],               #填充颜色
            outline = op[2],            #边框颜色
            stipple = op[3])            #使用位图填充
#绘制文字
cv.create_text(10, 260, text = '绘制多边形',
     font = titleFont,
     fill = 'magenta',
     anchor = W,
     justify = LEFT)
#定义列表,每个元素的 4 个值分别指定边框宽度、填充色、边框颜色、位图填充
options = [(None, "", 'black', None),
     (4, "", 'black', None),
     (4, 'pink', 'black', None),
     (4, 'pink', 'green', None),
     (4, 'pink', 'green', 'error')]
#采用循环绘制 5 个多边形
for i, op in enumerate(options):
     cv.create_polygon(130 + i * 140, 320, 185 + i * 140, 250, 240 + i * 140, 320,
            width = op[0],              #边框宽度
            fill = op[1],               #填充颜色
            outline = op[2],            #边框颜色
            stipple = op[3])            #使用位图填充
#绘制文字
cv.create_text(10, 360, text = '绘制扇形',
     font = titleFont,
     fill = 'magenta',
     anchor = W,
     justify = LEFT)
#定义列表,每个元素的 4 个值分别指定边框宽度、填充色、边框颜色、位图填充
options = [(None, None, None, None),
     (4, None, None, None),
     (4, 'pink', None, None),
     (4, 'pink', 'green', None),
     (4, 'pink', 'green', 'error')]
#采用循环绘制 5 个扇形
for i, op in enumerate(options):
     cv.create_arc(130 + i * 140, 350, 240 + i * 140, 420,
            width = op[0],              #边框宽度
            fill = op[1],               #填充颜色
            outline = op[2],            #边框颜色
            stipple = op[3])            #使用位图填充
#绘制文字
cv.create_text(10, 460, text = '绘制弓形',
     font = titleFont,
     fill = 'magenta',
     anchor = W,
     justify = LEFT)
#定义列表,每个元素的 4 个值分别指定边框宽度、填充色、边框颜色、位图填充
options = [(None, None, None, None),
     (4, None, None, None),
     (4, 'pink', None, None),
     (4, 'pink', 'green', None),
     (4, 'pink', 'green', 'error')]
#采用循环绘制 5 个弓形
for i, op in enumerate(options):
     cv.create_arc(130 + i * 140, 450, 240 + i * 140, 520,
```

```
            width = op[0],              #边框宽度
            fill = op[1],               #填充颜色
            outline = op[2],            #边框颜色
            stipple = op[3],            #使用位图填充
            start = 30,                 #指定起始角度
            extent = 60,                #指定逆时针转过角度
            style = CHORD)              # CHORD 指定绘制弓
#绘制文字
cv.create_text(10, 560, text = '仅绘弧',
     font = titleFont,
     fill = 'magenta',
     anchor = W,
     justify = LEFT)
#定义列表,每个元素的 4 个值分别指定边框宽度、填充色、边框颜色、位图填充
options = [(None, None, None, None),
     (4, None, None, None),
     (4, 'pink', None, None),
     (4, 'pink', 'green', None),
     (4, 'pink', 'green', 'error')]
#采用循环绘制 5 条弧
for i, op in enumerate(options):
     cv.create_arc(130 + i * 140, 550, 240 + i * 140, 620,
          width = op[0],              #边框宽度
          fill = op[1],               #填充颜色
          outline = op[2],            #边框颜色
          stipple = op[3],            #使用位图填充
          start = 30,                 #指定起始角度
          extent = 60,                #指定逆时针转过角度
          style = ARC)                #ARC 指定仅绘制弧
#绘制文字
cv.create_text(10, 660, text = '绘制直线',
     font = titleFont,
     fill = 'magenta',
     anchor = W,
     justify = LEFT)
#定义列表,每个元素的 5 个值分别指定边框宽度、线条颜色、位图填充、箭头风格、箭头形状
options = [(None, None, None, None, None),
     (6, None, None, BOTH, (20, 40, 10)),
     (6, 'pink', None, FIRST, (40, 40, 10)),
     (6, 'pink', None, LAST, (60, 50, 10)),
     (8, 'pink', 'error', None, None)]
#采用循环绘制 5 条直线
for i, op in enumerate(options):
     cv.create_line(130 + i * 140, 650, 240 + i * 140, 720,
          width = op[0],              #边框宽度
          fill = op[1],               #填充颜色
          stipple = op[2],            #使用位图填充
          arrow = op[3],              #箭头风格
          arrowshape = op[4])         #箭头形状
root.mainloop()
```

程序中演示了 Canvas 中不同的 create_xxx()方法的功能和用法,它们可用于创建矩形、椭圆、多边形、扇形、弓形、弧、直线、位图、图片和组件等。在绘制不同的图形时可指定不同的选项,从而实现丰富的绘制效果。运行程序,效果如图 3-30 所示。

掌握了上面的绘制方法后,已经可以实现一些简单的游戏了。

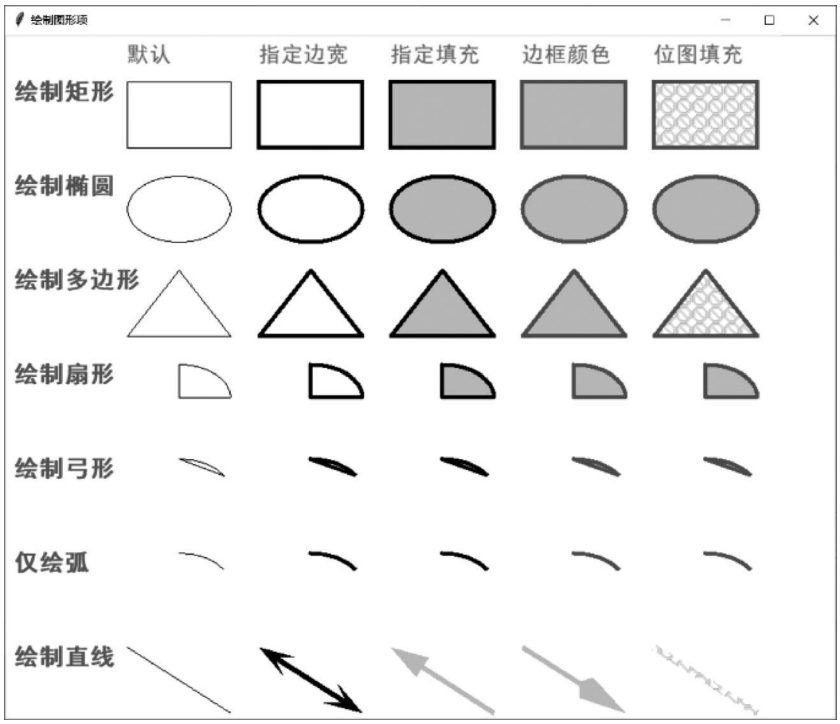

图 3-30　绘制图形

【例 3-24】 利用 Canvas 制作五子棋的图形界面。

该五子棋还需要根据用户的鼠标动作来确定下棋坐标，因此程序会为游戏界面的<Button-1>(左键单击)、<Motion>(鼠标移动)、<Leave>(鼠标移出)事件绑定事件处理函数。

```
from tkinter import *
import random

BOARD_WIDTH = 536
BOARD_HEIGHT = 538
BOARD_SIZE = 15
#定义棋盘坐标的像素值和棋盘数组之间的偏移距
X_OFFSET = 21
Y_OFFSET = 23
#定义棋盘坐标的像素值和棋盘数组之间的比率
X_RATE = (BOARD_WIDTH - X_OFFSET * 2) / (BOARD_SIZE - 1)
Y_RATE = (BOARD_HEIGHT - Y_OFFSET * 2) / (BOARD_SIZE - 1)
BLACK_CHESS = "●"
WHITE_CHESS = "○"
board = []
#把每个元素赋为"┼",代表无棋
for i in range(BOARD_SIZE) :
    row = ["┼"] * BOARD_SIZE
    board.append(row)
#创建窗口
root = Tk()
#禁止改变窗口大小
root.resizable(width = False, height = False)
#设置窗口标题
```

```
root.title('五子棋')
#创建并添加 Canvas
cv = Canvas(root, background = 'white',
     width = BOARD_WIDTH, height = BOARD_HEIGHT)
cv.pack()
bm = PhotoImage("board.png")
cv.create_image(BOARD_HEIGHT/2 + 1, BOARD_HEIGHT/2 + 1, image = bm)
selectedbm = PhotoImage("selected.gif")
#创建选中框图片,但该图片默认不在棋盘中
selected = cv.create_image( - 100, - 100, image = selectedbm)
def move_handler(event):
    #计算用户当前的选中点,并保证该选中点在 0～14 中
    selectedX = max(0, min(round((event.x - X_OFFSET) / X_RATE), 14))
    selectedY = max(0, min(round((event.y - Y_OFFSET) / Y_RATE), 14))
    #移动红色选择框
    cv.coords(selected,(selectedX * X_RATE + X_OFFSET,
        selectedY * Y_RATE + Y_OFFSET))
black = PhotoImage("black.gif")
white = PhotoImage("white.gif")
def click_handler(event):
    #计算用户的下棋点,并保证该下棋点在 0～14 中
    userX = max(0, min(round((event.x - X_OFFSET) / X_RATE), 14))
    userY = max(0, min(round((event.y - Y_OFFSET) / Y_RATE), 14))
    #当下棋点没有棋子时,用户才能下棋子
    if board[userY][userX] == "┼":
        cv.create_image(userX * X_RATE + X_OFFSET, userY * Y_RATE + Y_OFFSET,
            image = black)
        board[userY][userX] = "●"
        while(True):
            comX = random.randint(0, BOARD_SIZE - 1)
            comY = random.randint(0, BOARD_SIZE - 1)
            #如果计算机要下棋的点没有棋子时,才能让计算机下棋
            if board[comY][comX] == "┼": break
        cv.create_image(comX * X_RATE + X_OFFSET, comY * Y_RATE + _OFFSET,
            image = white)
        board[comY][comX] = "○"
def leave_handler(event):
    #将红色选中框移出界面
    cv.coords(selected, - 100, - 100)
#为鼠标移动事件绑定事件处理函数
cv.bind('< Motion >', move_handler)
#为鼠标单击事件绑定事件处理函数
cv.bind('< Button - 1 >', click_handler)
#为鼠标移出事件绑定事件处理函数
cv.bind('< Leave >', leave_handler)
root.mainloop()
```

运行以上程序后,弹出如图 3-31 所示的界面。当用户鼠标在棋盘上移动时,该选择框显示用户鼠标当前停留在哪个下棋点上。

程序在绘制黑色棋子和白色棋子的同时,也改变了底层代表棋盘状态的 board 列表的数据,这样既可记录下棋状态,从而让程序在后面可以根据 board[]列表来判断胜负,也可加入人工智能,根据 board[]列表来决定计算机的下棋点。

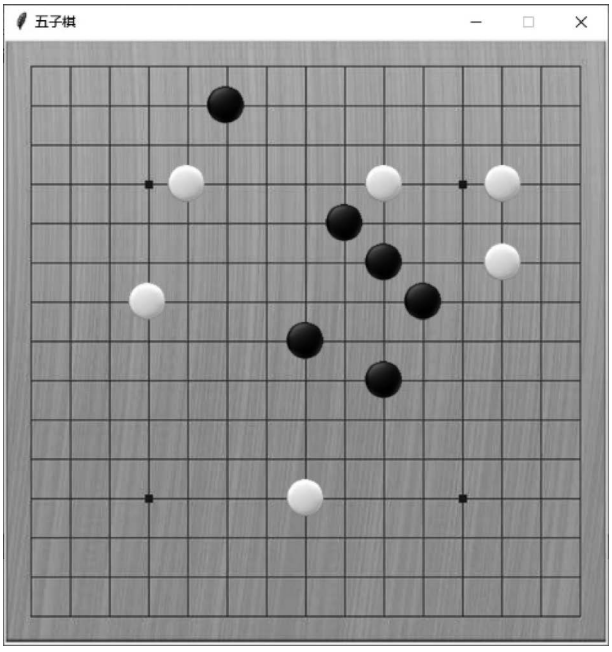

图 3-31　五子棋

3.5.2　绘制动画

在 Canvas 中实现动画时要添加一个定时器，周期性地改变界面上图形面的颜色、大小、位置等选项，用户看上去就是所谓的“动画”。

【例 3-25】 以一个简单的桌面弹球游戏来介绍使用 Canvas 绘制动画。使用 Tkinter Canvas 控件生成一个小球，并在画布上反复随机滚动，并交替更换颜色。

```
from tkinter import *
import time
from random import randint, seed

class Ball():
    def __init__(self, canvas, x1, y1, x2, y2, max_x,max_y):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
        self.max_x = max_x
        self.max_y = max_y
        self.center_x = 0
        self.center_y = 0
        self.canvas = canvas
        self.ball_color = 1
        #(x1,y1)和(x2,y2)分别对应圆的左上角和右下角坐标,也就是定义了圆的大小
        self.ball = canvas.create_oval(self.x1, self.y1, self.x2, self.y2, fill = "blue")

    def move_ball(self):
        deltax = randint(0,10)
        deltay = randint(0,15)
        self.center_x += deltax
        self.center_y += deltay
```

```
            if self.center_x > self.max_x or self.center_y > self.max_y:
                print('ball disapear')
                canvas.delete(self.ball)
                if self.ball_color == 1:
                    self.ball = canvas.create_oval(self.x1, self.y1, self.x2, self.y2, fill = "red")
                    self.ball_color = 2
                else:
                    self.ball = canvas.create_oval(self.x1, self.y1, self.x2, self.y2, fill = "blue")
                    self.ball_color = 1
                self.center_x = 0
                self.center_y = 0
            else:
                self.canvas.move(self.ball, deltax, deltay)
            self.canvas.after(200, self.move_ball)                    #定期自动更新
root = Tk()
root.title("循环球")
root.resizable(False,False)
canvas = Canvas(root, width = 300, height = 300)
canvas.pack()
seed(time.time())
ball = Ball(canvas, 0, 0, 20, 20,300,300)                             #创建 ball 实体
ball.move_ball()                                                      #开始移动球
root.mainloop()
```

运行程序，效果如图 3-32 所示。

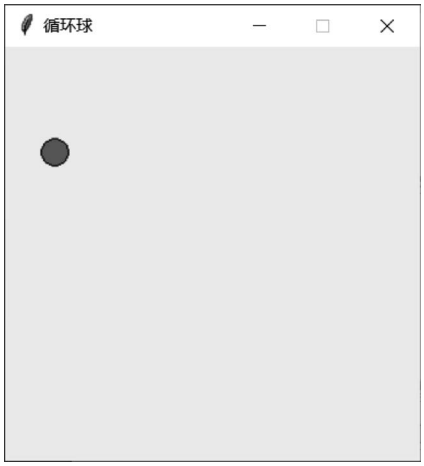

图 3-32　小球循环运动