# 第10课　Tkinter

# Tkinter简介

- Python有许多图形用户界面（GUI）模块可用于用开发GUI程序。
- 实验2中的turtle模块，可以绘制几何图形，但是我们不能使用turtle创建图形用户界面。
- Tkinter能够开发GUI项目，是“T k interface”的缩写（发音为T-K-Inter）。
- Tk是许多编程语言用来在Windows、Mac和UNIX上开发GUI程序的GUI库。
- Tkinter为Python程序员提供了一个使用Tk GUI库的接口，它实际上是用Python开发GUI程序的标准。
- 使用Tkinter开发GUI程序，同时也是学习面向对象编程的优秀工具

# Tkinter简介

- tkinter模块包含用于创建GUI的类。
- Tk类创建一个窗口，用于保存GUI小部件（即可视化组件）。

```
from tkinter import *  #导入tkinter模块

root = Tk()            #创建一个窗口
label = Label(root, text = "Welcome to Python")
                       #创建一个标签
button = Button(root, text = "Click Me")
                       #创建一个按钮

label.pack()           #在窗口中显示标签
button.pack()          #在窗口中显示按钮

root.mainloop()        #创建事件循环
```

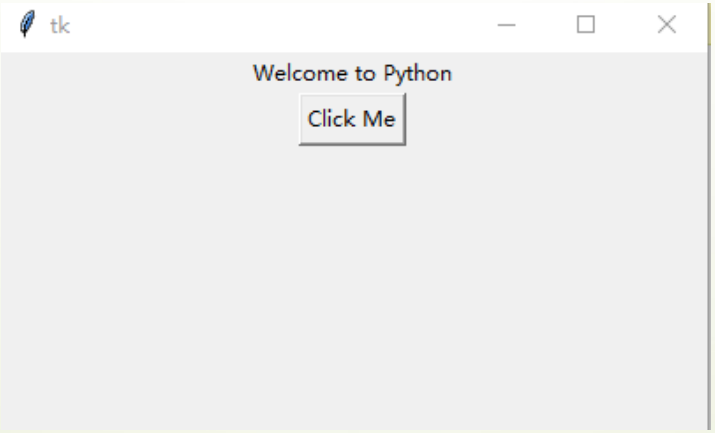
# Tkinter简介

```
from tkinter import *
root = Tk()
label = Label(root, text = "Welcome to Python")
button = Button(root, text = "Click Me")
```

#Label和Button是用于创建标签和按钮的Python Tkinter小部件类。小部件类的第一个参数始终是父“容器”（即小部件将被放置在其中的容器）。

```
label.pack()
button.pack()
```

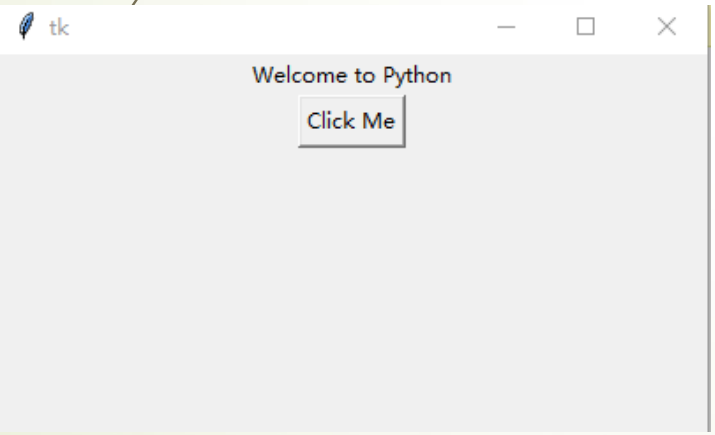
#使用pack管理器在容器中放置标签。在本例中， pack管理器逐行放置窗口中的小部件。

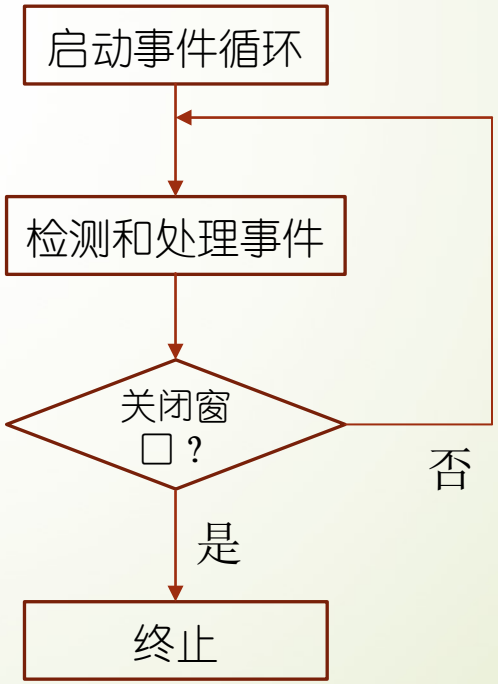


# Tkinter简介

Tkinter GUI编程是事件驱动的。显示用户界面后，程序等待用户交互，如鼠标单击和按键。

```
root.mainloop()        #创建事件循环
```



# 处理事件

Tkinter小部件可以绑定到函数，该函数在事件发生时调用。

- 当用户单击按钮时，程序应处理此事件。
- 通过定义处理函数并将函数绑定到按钮，可以启用此操作。

```
def processOK():
    print("OK button is clicked")

def processCancel():
    print("Cancel button is clicked")

root = Tk()

btOK = Button(root, text = "OK", command = processOK)
btCancel = Button(root, text = "Cancel", command = processCancel)


btOK.pack()
btCancel.pack()

root.mainloop()
```

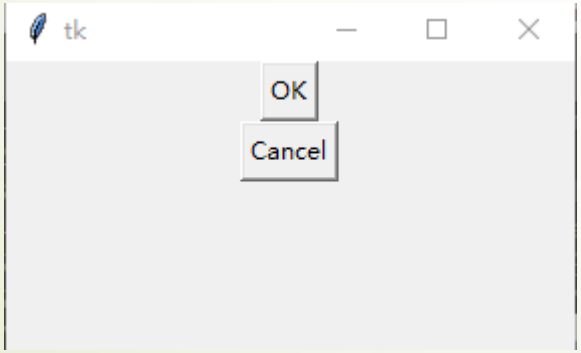
# 处理事件

```
def processOK():
    print("OK button is clicked")

def processCancel():
    print("Cancel button is clicked")
```

将OK按钮（Cancel按钮）绑定到processOK（processCancel ）函数，单击该按钮时将调用该函数。

```
btOK = Button(root, text = "OK", command = processOK)
btCancel = Button(root, text = "Cancel", command = processCancel)
```

程序定义了函数processOK和processCancel。这些函数在构造按钮时绑定到按钮。这些函数称为回调函数（callback function）或处理函数（handler）。




```
OK button is clicked
OK button is clicked
OK button is clicked
Cancel button is clicked
Cancel button is clicked
Cancel button is clicked
```

在命令窗口中监视正在处理的事件。

# 处理事件

面向对象的写法

```
class ProcessButtonEvent:
    def __init__(self):
        window = Tk()
        btOK = Button(window, text = "OK", command = self.processOK)

        btCancel = Button(window, text = "Cancel",
                          command = self.processCancel)

        btOK.pack()
        btCancel.pack()

        window.mainloop()

    def processOK(self):
        print("OK button is clicked")

    def processCancel(self):
        print("Cancel button is clicked")

ProcessButtonEvent()
```

程序定义了一个ProcessButtonEvent类，在构造函数__init__中创建GUI。函数processOK和processCancel现在是类中的实例方法，因此它们由self.processOK和self.processCancel调用。

# 处理事件

面向对象的写法

```
class ProcessButtonEvent:
    def __init__(self):
        window = Tk()
        btOK = Button(window, text = "OK", command = self.processOK)

        btCancel = Button(window, text = "Cancel"
                          command = self.processCancel)

        btOK.pack()
        btCancel.pack()

        window.mainloop()

    def processOK(self):
        print("OK button is clicked")

    def processCancel(self):
        print("Cancel button is clicked")

ProcessButtonEvent()
```

定义类用于创建GUI和处理GUI事件的**优点**：
（1）将来重用该类。
（2）所有函数定义为类的方法，使它们能够访问类中的实例数据。

# 小部件（Widget）类

Tkinter的GUI类定义常见的GUI小部件，如按钮、标签、单选按钮、检查按钮、条目、画布和其他。

| 小部件 | 解释 |
| --- | --- |
| Button | 一个简单的按钮，用于执行命令 |
| Canvas | 结构化图形，用于绘制图形和绘图、创建图形编辑器和实现自定义小部件 |
| Checkbutton | 单击复选按钮，可在值之间切换 |
| Entry | 文本输入字段，也称为文本字段或文本框 |
| Frame | 用于包含其他小部件的容器小部件 |
| Label | 显示文本或图像 |
| Menu | 菜单窗格，用于实现下拉菜单和弹出菜单 |
| Menubutton | 一种菜单按钮，用于实现下拉菜单 |
| Message | 显示文本。类似于label小部件，但可以自动将文本包装为给定的宽度或纵横比 |
| Radiobutton | 单击单选按钮可将变量设置为该值，并清除与同一变量关联的所有其他单选按钮 |
| Text | 格式化文本显示。显示和编辑具有各种样式和属性的文本。还支持嵌入图像和窗口 |

# 小部件（Widget）类


```
check button is checked
check button is unchecked
check button is checked
check button is unchecked
```

```
class WidgetsDemo:
    def __init__(self):
        window = Tk()
        window.title("Widgets Demo") #标题

        # Frame：用于包含其他小部件的容器小部件
        frame1 = Frame(window)
        frame1.pack()

        # Checkbutton：单击复选按钮，可在值之间切换
        self.v1 = IntVar()
        cbtBold = Checkbutton(frame1, text = "Bold",
            variable = self.v1, command = self.processCheckbutton)

        cbtBold.grid(row = 1, column = 1)

        window.mainloop()


    def processCheckbutton(self):
        print("check button is "
            + ("checked " if self.v1.get() == 1 else "unchecked"))
```

网格几何图形管理器用于将复选按钮放置到frame1中。

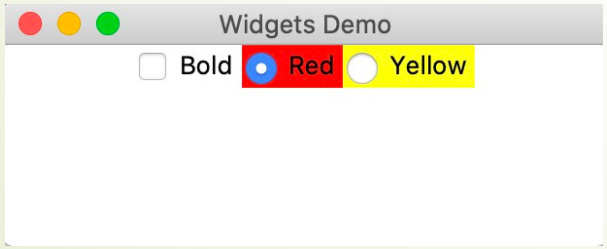
可以使用文本字段输入值。该值必须是IntVar、DoubleVar或StringVar的对象，分别表示整数、浮点数或字符串，它们在tkinter模块中定义。

如果选中复选按钮，v1被设为1

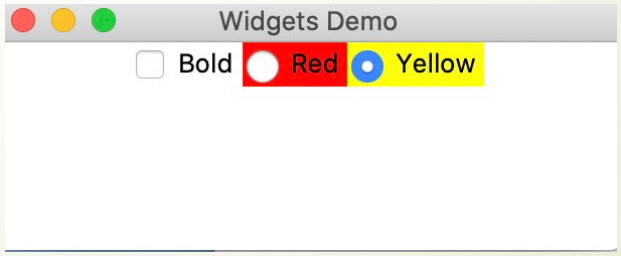
# 小部件（Widget）类

```
class WidgetsDemo:
    def __init__(self):
        …

        # Radiobutton单击单选按钮可将变量设置为该值，
        # 并清除与同一变量关联 所有 其他单选按钮
        self.v2 = IntVar()
        rbRed = Radiobutton(frame1, text = "Red", bg = "red",
                variable = self.v2, value = 1,
                command = self.processRadiobutton)
        rbYellow = Radiobutton(frame1, text = "Yellow",
                bg = "yellow", variable = self.v2, value = 2,
                command = self.processRadiobutton)

        rbRed.grid(row = 1, column = 2)
        rbYellow.grid(row = 1, column = 3)

        window.mainloop()

    def processRadiobutton(self):
        print(("Red" if self.v2.get() == 1 else "Yellow") + " is selected " )
```

如果选中红色单选按钮，则v2设置为1；
如果选中黄色单选按钮，则v2设置为2。




Red is selected

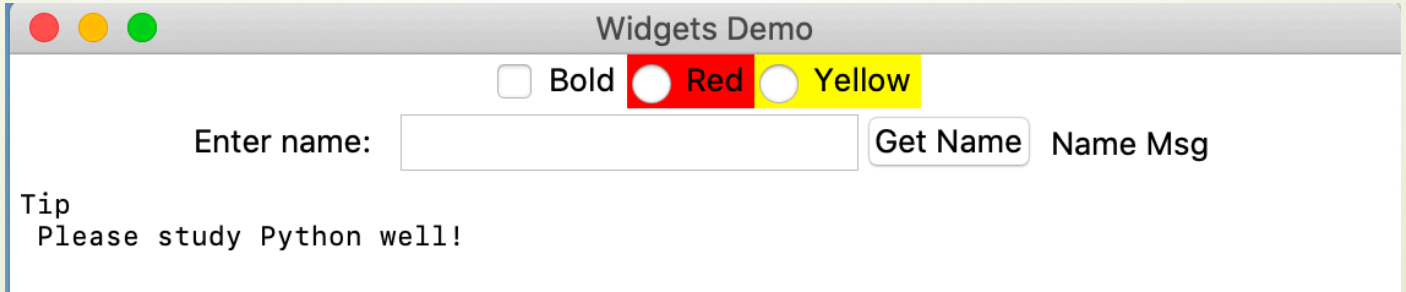



Yellow is selected

# 小部件（Widget）类

```
class WidgetsDemo:
    def __init__(self):
        …
        frame2 = Frame(window) # 在window创建新的frame2
        frame2.pack()

        label = Label(frame2, text = "Enter name: ")
        self.name = StringVar()
        entryName = Entry(frame2, textvariable = self.name)
        btGetName = Button(frame2, text = "Get Name",
            command = self.processButton)
        message = Message(frame2, text = ” Name Msg")

        label.grid(row = 1, column = 1)
        entryName.grid(row = 1, column = 2)
        btGetName.grid(row = 1, column = 3)
        message.grid(row = 1, column = 4)

        text = Text(window) # Create a text add to the window
        text.pack()
        text.insert(END, "Tip\n Please study Python well!")
```

END选项指定将文本插入到当前内容的结尾。



# 小部件（Widget）类

```
class WidgetsDemo:
    def __init__(self):
        window = Tk()
        window.title("Widgets Demo")

        frame1 = Frame(window)
        frame1.pack()

        self.v2 = IntVar()
        self.rbRed = Radiobutton(frame1, text = "Red", bg = "white",
                variable = self.v2, value = 1,
                command = self.processRadiobutton)
        self.rbYellow = Radiobutton(frame1, text = "Yellow",
                bg = "white", variable = self.v2, value = 2,
                command = self.processRadiobutton)

        self.rbRed.grid(row = 1, column = 1)
        self.rbYellow.grid(row = 1, column = 2)
```

如何更改小部件（颜色、字体和文本等）？

思考processRadiobutton方法实现的功能。

# 小部件（Widget）类

```
class WidgetsDemo:
    def __init__(self):
        window = Tk()
        window.title("Widgets Demo")

        frame1 = Frame(window)
        frame1.pack()

        self.v2 = IntVar()
        self.rbRed = Radiobutton(frame1, text = "Red", bg = "white",
                variable = self.v2, value = 1,
                command = self.processRadiobutton)
        self.rbYellow = Radiobutton(frame1, text = "Yellow",
                bg = "white", variable = self.v2, value = 2,
                command = self.processRadiobutton)

        self.rbRed.grid(row = 1, column = 1)
        self.rbYellow.grid(row = 1, column = 2)

    def processRadiobutton(self):
        if self.v2.get() == 1:
            self.rbRed["bg"] = "red"
            self.rbYellow["bg"] = "white"
        elif self.v2.get() == 2:
            self.rbYellow["bg"] = "yellow"
            self.rbRed["bg"] = "white"
```

如何更改小部件（颜色、字体和文本等）？

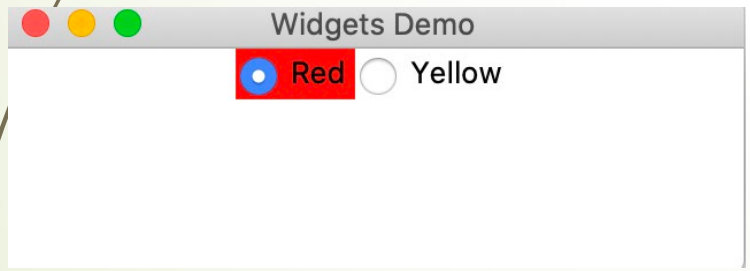
思考processRadiobutton方法实现的功能。

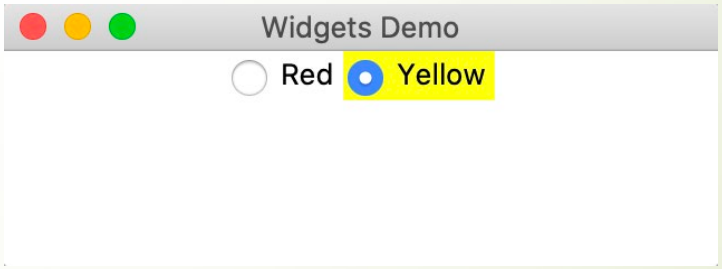
# 小部件（Widget）类

```
def processRadiobutton(self):
        if self.v2.get() == 1:
            self.rbRed["bg"] = "red"
            self.rbYellow["bg"] = "white"
        elif self.v2.get() == 2:
            self.rbYellow["bg"] = "yellow"
            self.rbRed["bg"] = "white"
```

如何更改小部件（颜色、字体和文本等）？

思考processRadiobutton方法实现的功能：更改背景颜色(类似字典）

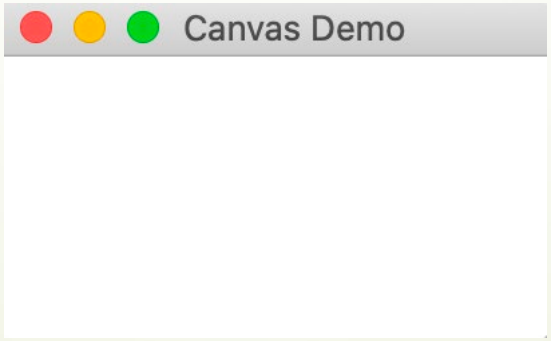


# Canvas

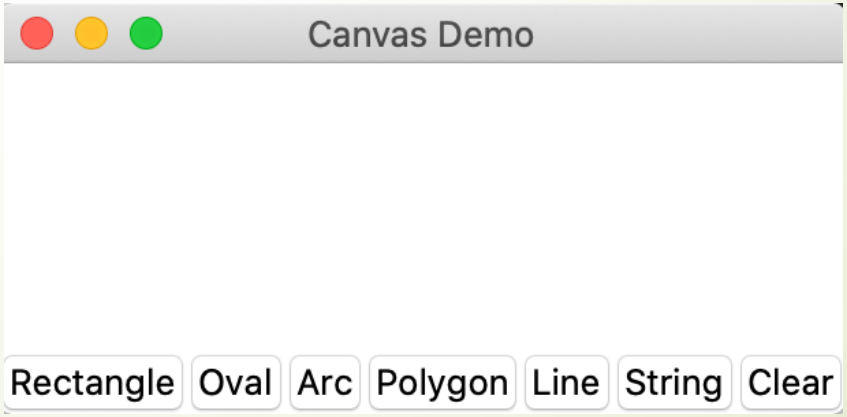
使用画布小部件显示形状，如矩形、椭圆形、圆弧，多边形等。

```
class CanvasDemo:
    def __init__(self):
        window = Tk()
        window.title("Canvas Demo")

        self.canvas = Canvas(window, width = 200, height = 100, bg = "white")
        self.canvas.pack()

        frame = Frame(window)
        frame.pack()


        window.mainloop()
```



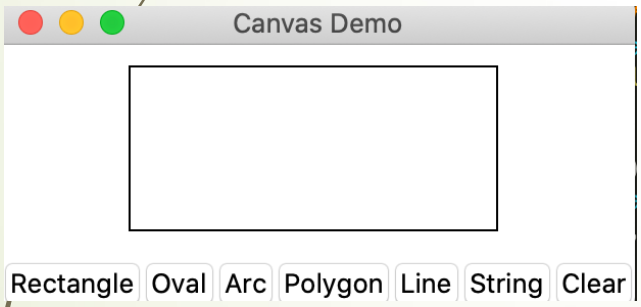
# Canvas

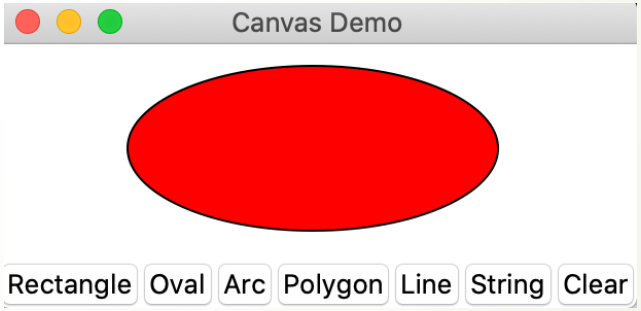
```
class CanvasDemo:
    def __init__(self):
        …
        frame = Frame(window)
        frame.pack()

        btRectangle = Button(frame, text = "Rectangle", command = self.displayRect)
        btOval = Button(frame, text = "Oval", command = self.displayOval)
        …
        btString = Button(frame, text = "String", command = self.displayString)
        btClear = Button(frame, text = "Clear", command = self.clearCanvas)

        btRectangle.grid(row = 1, column = 1)
        btOval.grid(row = 1, column = 2)
        …
        btString.grid(row = 1, column = 6)
        btClear.grid(row = 1, column = 7)

        window.mainloop()
```

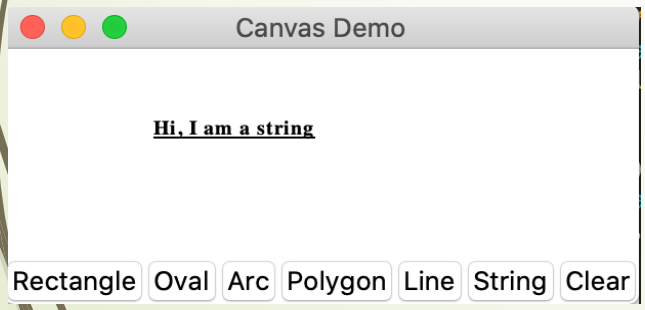


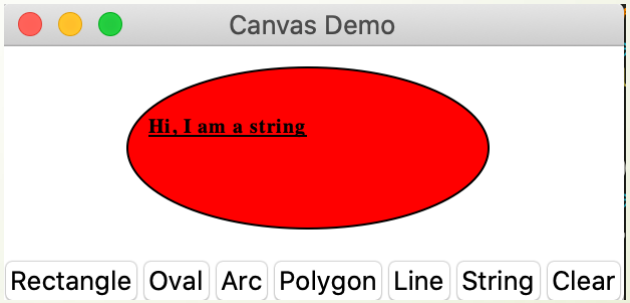
# Canvas

```
class CanvasDemo:
      def displayRect(self):
          self.canvas.create_rectangle(10, 10, 190, 90, tags = "rect")
      def displayOval(self):
          self.canvas.create_oval(10, 10, 190, 90, fill = "red", tags = "oval")
      …
      def displayString(self):
          self.canvas.create_text(60, 40, text = "Hi, I am a string",
               font = "Times 10 bold underline", tags = "string")

      def clearCanvas(self):
          self.canvas.delete("rect", "oval", "arc", "polygon", "line", "string")
```



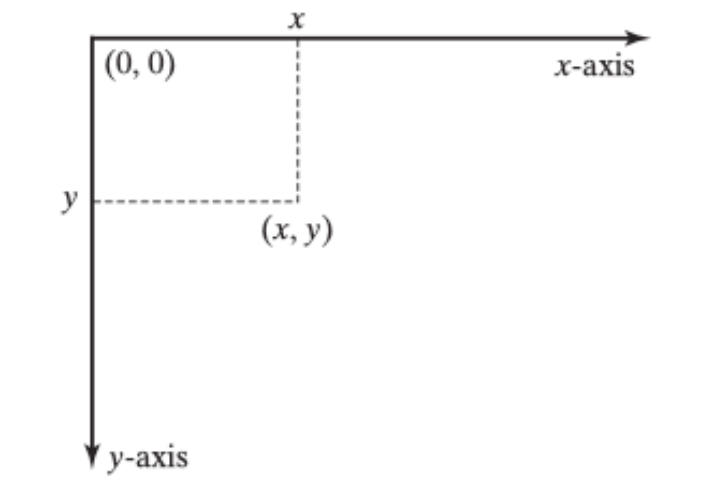



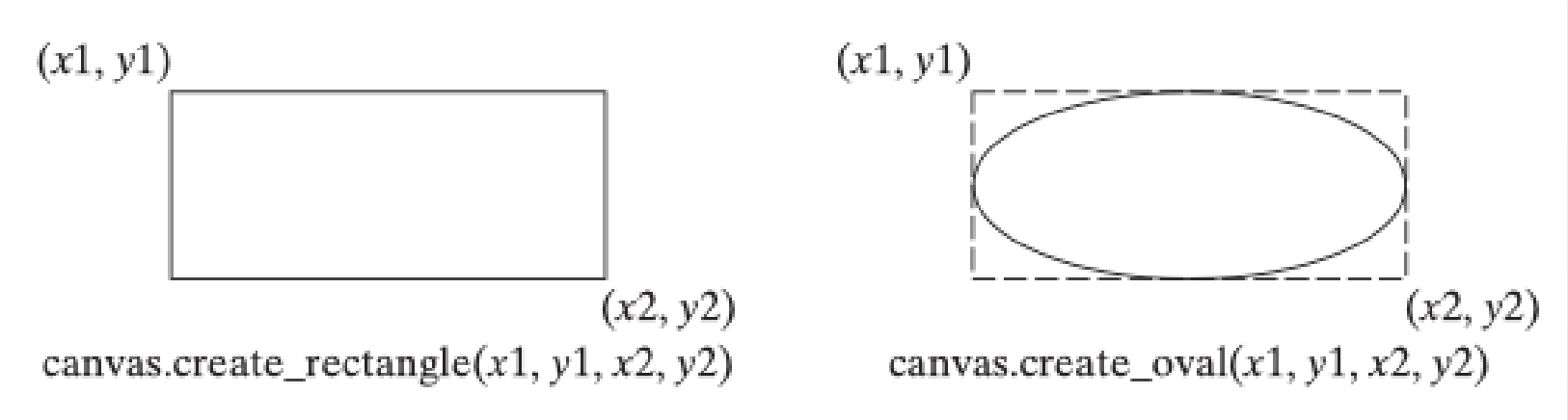



# Canvas

```
self.canvas.create_rectangle(10, 10, 190, 90, tags = "rect")
```



# 几何图形管理器

- Tkinter使用几何管理器将小部件放置在容器中。
- Tkinter支持三个几何图形管理器：网格（grid）管理器、包（pack）管理器和位置（place）管理器。

注意：由于每个管理器都有自己的小部件放置样式，因此将小部件的管理器混合在同一容器中不是一个好的做法。可以使用frame部件作为子容器来实现所需的布局。

网格（grid）管理器

•将小部件放入容器中不可见网格的单元格中。

•将小部件放在指定的行和列中。

•还可以使用rowspan和columnspan参数将小部件放置在多个行和列中。

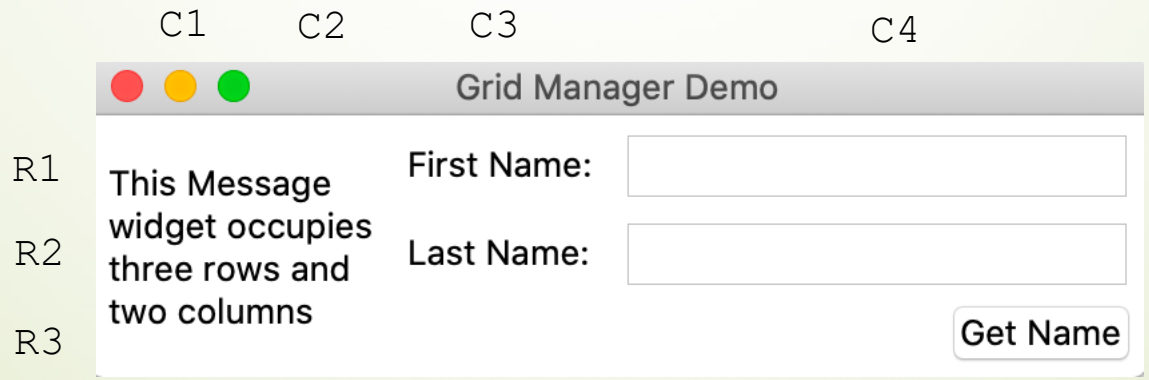
# 网格（grid）管理器

```
class GridManagerDemo:
    window = Tk()
    window.title("Grid Manager Demo")

    message = Message(window, text = "This Message widget occupies
three rows and two columns")
    message.grid(row = 1, column = 1, rowspan = 3, columnspan = 2)

    Label(window, text = "First Name:").grid(row = 1, column = 3)
    Entry(window).grid(row = 1, column = 4, padx = 5, pady = 5)

    Label(window, text = "Last Name:").grid(row = 2, column = 3)
    Entry(window).grid(row = 2, column = 4)

    Button(window, text = "Get Name").grid(row = 3,
        padx = 5, pady = 5, column = 4, sticky = E)

    window.mainloop()

GridManagerDemo()
```

# 网格（grid）管理器

```
class GridManagerDemo:
    …
    message = Message(window, text = "This Message widget occupies
three rows and two columns")
    message.grid(row = 1, column = 1, rowspan = 3, columnspan = 2)

    Label(window, text = "First Name:").grid(row = 1, column = 3)
    Entry(window).grid(row = 1, column = 4, padx = 5, pady = 5)

    Label(window, text = "Last Name:").grid(row = 2, column = 3)
    Entry(window).grid(row = 2, column = 4)

    Button(window, text = "Get Name").grid(row = 3,
        padx = 5, pady = 5, column = 4, sticky = E)
```

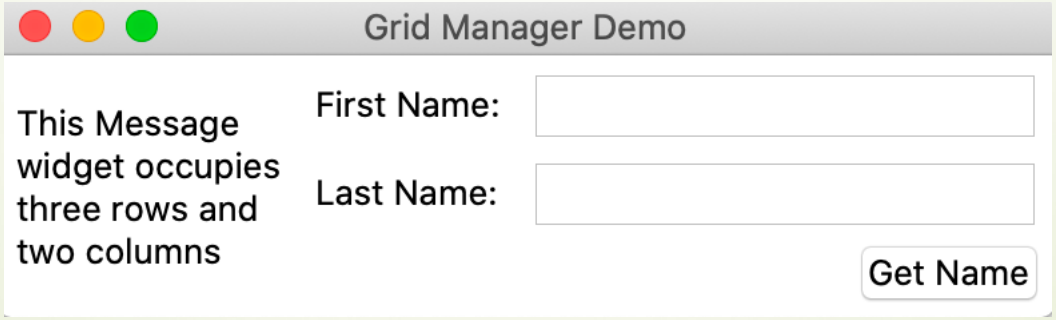
Message 小部件放在第1,2,3 行和第1, 2列合并的单元格



# 网格（grid）管理器

```
class GridManagerDemo:
    …
    message = Message(window, text = "This Message widget occupies
three rows and two columns")
    message.grid(row = 1, column = 1, rowspan = 3, columnspan = 2)

    Label(window, text = "First Name:").grid(row = 1, column = 3)
    Entry(window).grid(row = 1, column = 4, padx = 5, pady = 5)

    Label(window, text = "Last Name:").grid(row = 2, column = 3)
    Entry(window).grid(row = 2, column = 4)

    Button(window, text = "Get Name").grid(row = 3,
        padx = 5, pady = 5, column = 4, sticky = E)
```

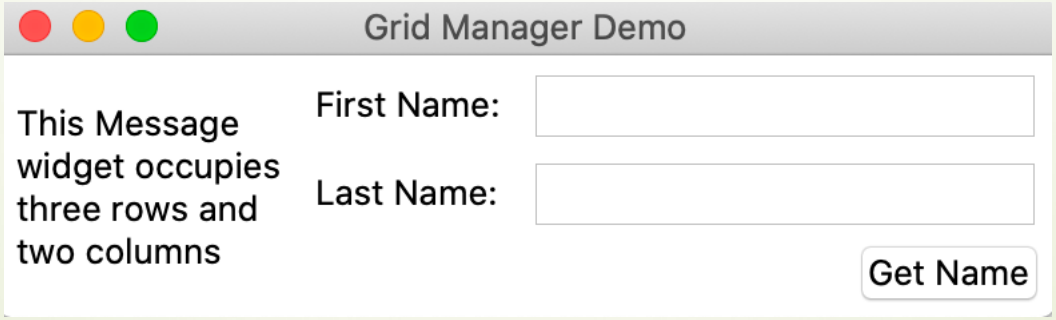
sticky=E选项在单元格中向东粘贴，使其与同一列中的条目小部件右对齐（S, N, E, W, NW, NE, SW, SE）。



# 网格（grid）管理器

```
class GridManagerDemo:
    …
    message = Message(window, text = "This Message widget occupies
three rows and two columns")
    message.grid(row = 1, column = 1, rowspan = 3, columnspan = 2)

    Label(window, text = "First Name:").grid(row = 1, column = 3)
    Entry(window).grid(row = 1, column = 4, padx = 5, pady = 5)

    Label(window, text = "Last Name:").grid(row = 2, column = 3)
    Entry(window).grid(row = 2, column = 4)

    Button(window, text = "Get Name").grid(row = 3,
        padx = 5, pady = 5, column = 4, sticky = E)
```

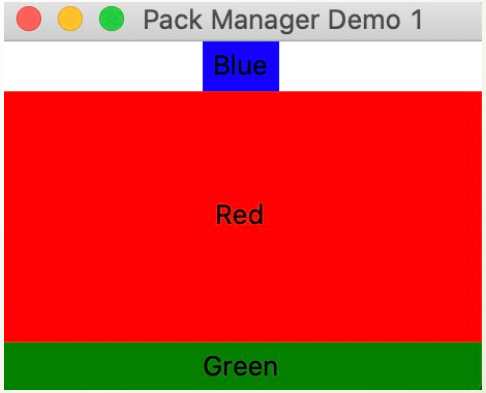
padx和pady选项填充单元格中的水平和垂直空间。还可以使用ipadx和ipady选项在小部件边框内填充的水平和垂直空间。



# 包（pack）管理器

包管理器可以将小部件放在彼此的顶部或并排放置。

```
class PackManagerDemo:
    def __init__(self):
        window = Tk()
        window.title("Pack Manager Demo 1")

        Label(window, text = "Blue", bg = "blue").pack()
        Label(window, text = "Red", bg = "red").pack(
            fill = BOTH, expand = 1)
        Label(window, text = "Green", bg = "green").pack(
            fill = BOTH)

        window.mainloop()

PackManagerDemo()
```

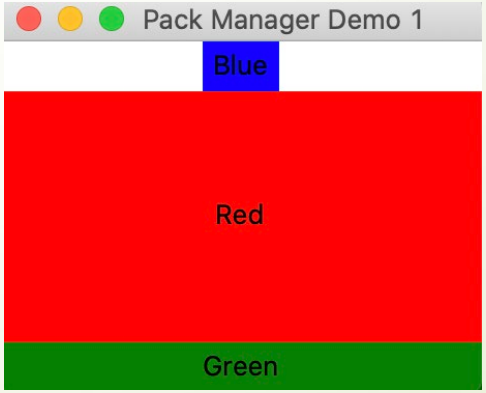


# 包（pack）管理器

```
class PackManagerDemo:
    def __init__(self):
        …
        Label(window, text = "Blue", bg = "blue").pack()
        Label(window, text = "Red", bg = "red").pack(
            fill = BOTH, expand = 1)
        Label(window, text = "Green", bg = "green").pack(
            fill = BOTH)
        …
```

fill选项使用X、Y或BOTH来水平、垂直或双向填充。

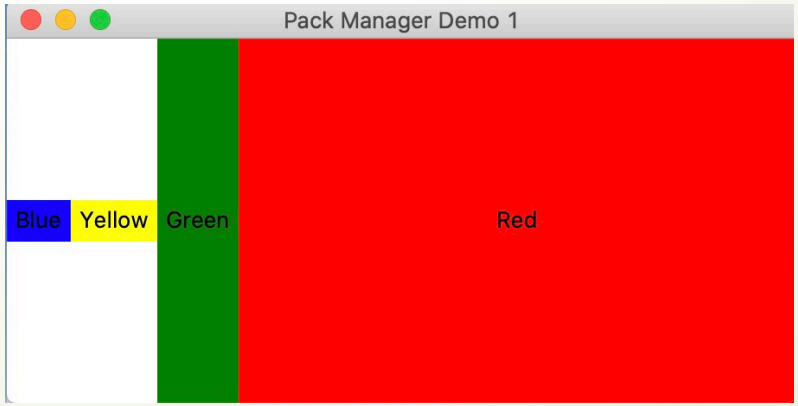
expand选项告诉pack管理器为小部件框分配额外的空间



# 包（pack）管理器

```
class PackManagerDemo:
    def __init__(self):
        …
        Label(window, text = "Blue", bg = "blue").pack(side = LEFT)
        Label(window, text = "Yellow", bg = "yellow").pack(side = LEFT)
        Label(window, text = "Red", bg = "red").pack(
            fill = BOTH, expand = 1,side = RIGHT)
        Label(window, text = "Green", bg = "green").pack(
            fill = BOTH, side = RIGHT)        …
```

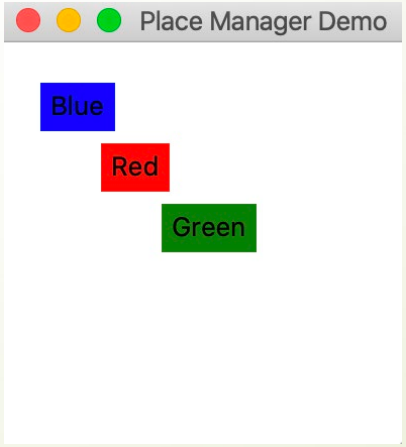
标签使用side选项并排打包。side选项可以是LEFT、RIGHT、TOP或BOTTOM。默认情况下，设置为“TOP”。



# 位置（place）管理器

位置（place）管理器将小部件放置在绝对位置。

```
class PlaceManagerDemo:
    def __init__(self):
        window = Tk()
        window.title("Place Manager Demo")

        Label(window, text = "Blue", bg = "blue").place(
            x = 20, y = 20)
        Label(window, text = "Red", bg = "red").place(
            x = 50, y = 50)
        Label(window, text = "Green", bg = "green").place(
            x = 80, y = 80)

        window.mainloop()

PlaceManagerDemo()
```



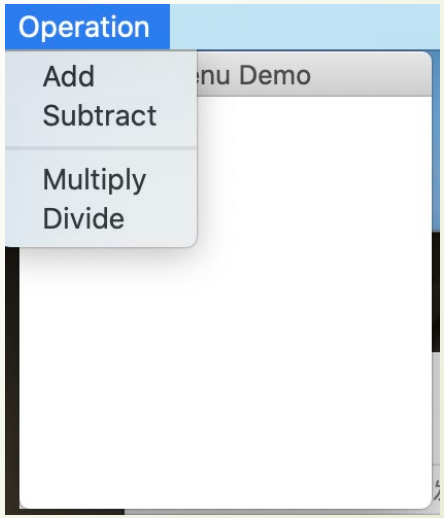
# 菜单

- 使用Tkinter创建菜单、弹出菜单和工具栏。
- 使用Menu类创建菜单栏和菜单，并使用add_command方法向菜单选项。

```
class MenuDemo:
    def __init__(self):
        window = Tk()
        window.title("Menu Demo")

        menubar = Menu(window)
        window.config(menu = menubar)

        mainloop()
```

# 菜单

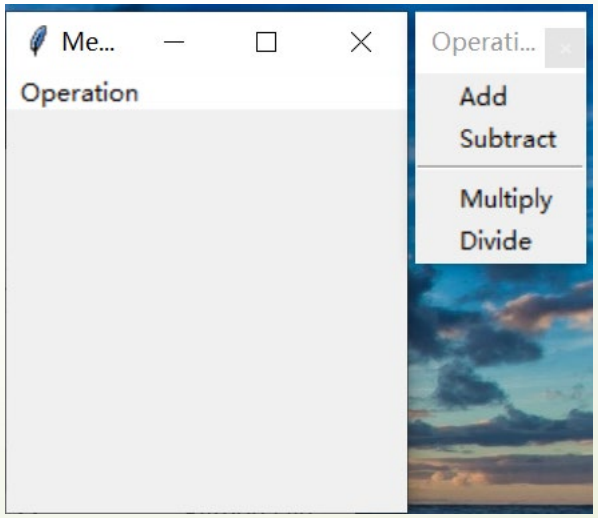
```
class MenuDemo:
    def __init__(self):
        …
        operationMenu = Menu(menubar, tearoff = 0)
        menubar.add_cascade(label = "Operation", menu =
operationMenu)
        #要在菜单栏中创建菜单，将menubar用作父容器，并调用menubar的
add_cascade方法来设置menu标签。

        operationMenu.add_command(label = "Add",
            command = self.add)
        …

        mainloop()
```




```
def add(self):
        self.v3.set(eval(self.v1.get()) +
eval(self.v2.get()))
```

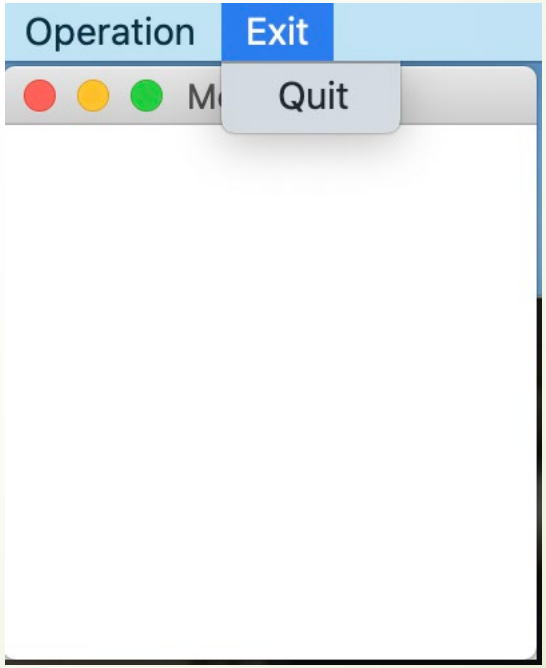
# 菜单

```
class MenuDemo:
    def __init__(self):
        …
        operationMenu = Menu(menubar, tearoff = 1)
        menubar.add_cascade(label = "Operation", menu =
operationMenu)

        operationMenu.add_command(label = "Add",
            command = self.add)
        …

        mainloop()
```



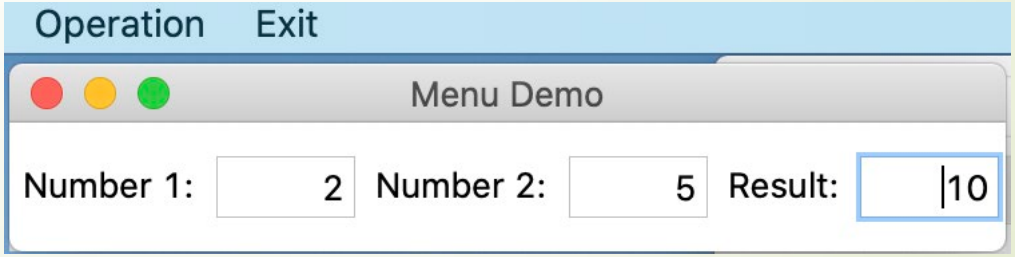
# 菜单

```
class MenuDemo:
    def __init__(self):
        …
        exitmenu = Menu(menubar, tearoff = 0)
        menubar.add_cascade(label = "Exit", menu = exitmenu)
        exitmenu.add_command(label = "Quit", command =
window.quit)
        …

        mainloop()
```



# 菜单

```
class MenuDemo:
    def __init__(self):
        …
        frame1 = Frame(window)
        frame1.grid(row = 2, column = 1, pady = 10)

        Label(frame1, text = "Number 1:").pack(side = LEFT)
        self.v1 = StringVar()
        Entry(frame1, width = 5, textvariable = self.v1,
              justify = RIGHT).pack(side = LEFT)
        Label(frame1, text = "Number 2:").pack(side = LEFT)
        self.v2 = StringVar()
        Entry(frame1, width = 5, textvariable = self.v2,
              justify = RIGHT).pack(side = LEFT)
        Label(frame1, text = "Result:").pack(side = LEFT)
        self.v3 = StringVar()
        Entry(frame1, width = 5, textvariable = self.v3,
              justify = RIGHT).pack(side = LEFT)  …

        mainloop()
```



# 鼠标、键盘键事件和绑定

可以使用bind方法将鼠标和键盘键事件绑定到小部件。

以下语法将鼠标事件与回调处理程序绑定：

```
widget.bind(event, handler)
```

如果发生匹配事件event，则调用处理程序handler。 event是一个标准的Tkinter对象，在事件发生时自动创建。

每个handler都有一个event作为其参数

```
handler(event):
```

event对象有许多属性，描述与事件相关的事件。例如，对于鼠标事件，对象使用x，y属性以像素为单位捕获当前鼠标位置。

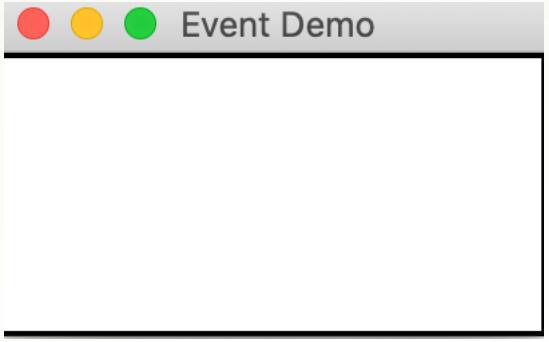
# 事件

| 事件 | 描述 |
| --- | --- |
| <Bi-Motion> | 在按住小部件时移动鼠标按钮 |
| <Button-i> | Button-1、Button-2和Button-3标识左、中、右按钮。当鼠标按钮按在小部件上时，Tkinter会自动获取鼠标指针的位置 |
| <ButtonReleased-i> | 释放鼠标按钮时 |
| <Double-Button-i> | 双击鼠标按钮时 |
| <Enter> | 鼠标指针进入小部件时 |
| <Key> | 按键按下时 |
| <Leave> | 鼠标指针离开小部件时 |
| <Return> | 按下回车键时 |
| <Shift+A> | 按下Shift+A键时 |

# 事件的属性

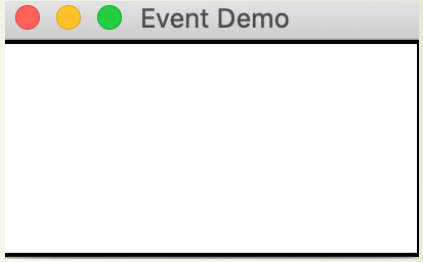
| 事件 | 描述 |
| --- | --- |
| char | 从键盘输入的用于键事件的字符 |
| keycode | 从键盘输入的用于键事件的键代码（即Unicode）。 |
| keysym | 从键盘输入的用于键事件的键符号（即字符）。 |
| num | 按钮编号（1、2、3）指示单击了哪个鼠标按钮 |
| widget | 触发此事件的小部件对象 |
| x, y | 小部件中的当前鼠标位置（像素） |
| | |

# 鼠标、键盘键事件和绑定




```
class MouseKeyEventDemo:
    def __init__(self):
        window = Tk()
        window.title("Event Demo")
        canvas = Canvas(window, bg = "white", width = 200, height = 100)
        canvas.pack()

        # 绑定<Button-1>事件
        canvas.bind("<Button-1>", self.processMouseEvent)

        # 绑定<Key>事件
        canvas.bind("<Key>", self.processKeyEvent)
        canvas.focus_set()#获得键盘焦点：设置在画布上，以便画布接收来自键盘的输入

        window.mainloop()
```

# 鼠标、键盘键事件和绑定



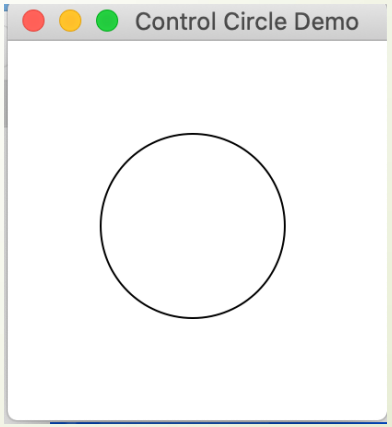

```
class MouseKeyEventDemo:
    …

    def processMouseEvent(self, event):
        print("clicked at", event.x, event.y)
        print("Position in the screen", event.x_root, event.y_root)
        print("Which button is clicked? ", event.num)

    def processKeyEvent(self, event):
        print("keysym? ", event.keysym)
        print("char? ", event.char)
        print("keycode? ", event.keycode)
```

```
clicked at 29 28
Position in the screen 723 500
Which button is clicked?  1

clicked at 77 81
Position in the screen 771 553
Which button is clicked?  1

clicked at 165 41
Position in the screen 859 513
Which button is clicked?  1
```

```
keysym?  space
char?
keycode?  32

keysym?  a
char?  a
keycode?  97

keysym?  Return
char?
keycode?  2359309
```

# 鼠标、键盘键事件和绑定

```
class EnlargeShrinkCircle:
    def __init__(self):
        self.radius = 50

        window = Tk()
        window.title("Control Circle Demo")
        self.canvas = Canvas(window, bg = "white",
            width = 200, height = 200)
        self.canvas.pack()
        self.canvas.create_oval(
            100 - self.radius, 100 - self.radius,
            100 + self.radius, 100 + self.radius, tags = "oval")

        self.canvas.bind("<Button-1>", self.increaseCircle)
        self.canvas.bind("<Button-3>", self.decreaseCircle)

        window.mainloop()

    def increaseCircle(self, event):
        pass
    def decreaseCircle(self, event):
        pass
```

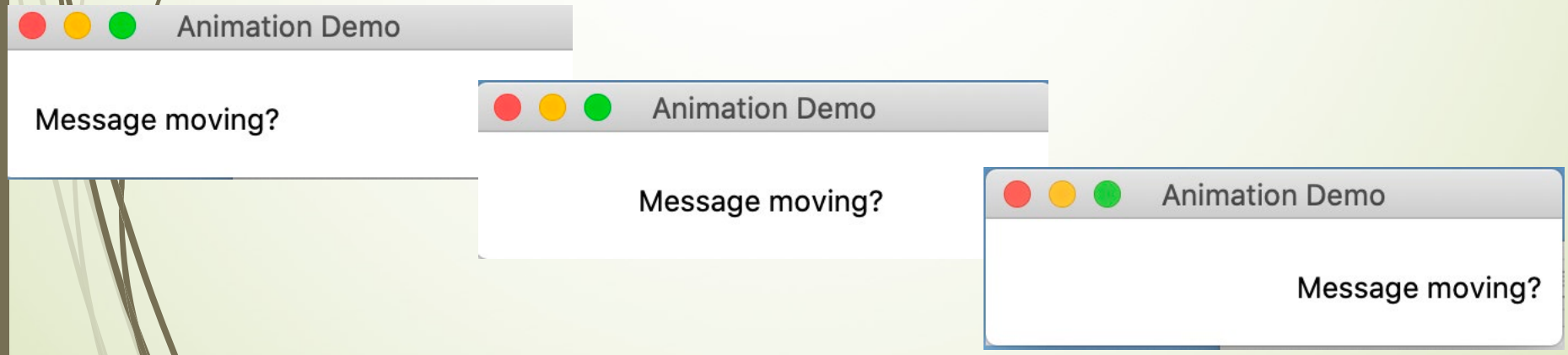


# 动画

```
class AnimationDemo:
    def __init__(self):
        window = Tk()
        window.title("Animation Demo")

        width = 250
        canvas = Canvas(window, bg = "white", width = 250, height = 50)
        canvas.pack()

        x = 0
        canvas.create_text(x, 30, text = "Message moving?", tags = "text")

        dx = 3
        while True:
            canvas.move(“text”, dx, 0)
            canvas.after(100)
            canvas.update()
            if x < width:
                x += dx
            else:
                x = 0
                canvas.delete("text")

                canvas.create_text(x, 30, text = "Message moving?",
                    tags = "text")

        window.mainloop()
```

# 动画

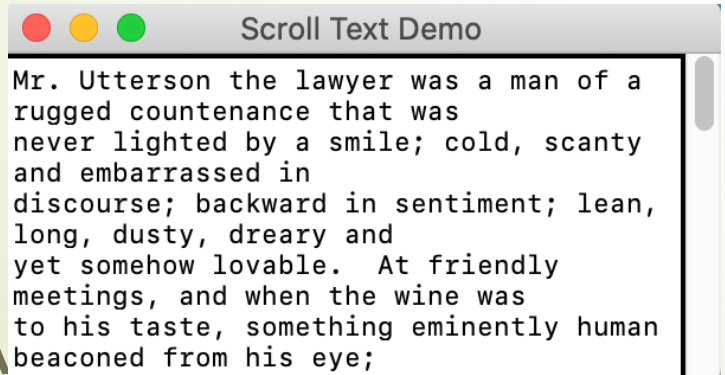
```
class AnimationDemo:
    def __init__(self):
        …
        dx = 3
        while True:
            canvas.move("text", dx, 0) # 移动文本dx单位
            canvas.after(100) # 睡眠100毫秒
            canvas.update() # 更新canvas
            if x < width:
                x += dx      # 获取字符串的当前位置
            else:
                x = 0        # 将字符串位置重置为开头
                canvas.delete("text")
                             # 在开始处重画文本
                canvas.create_text(x, 30, text = "Message moving?", tags = "text")

        window.mainloop()
```

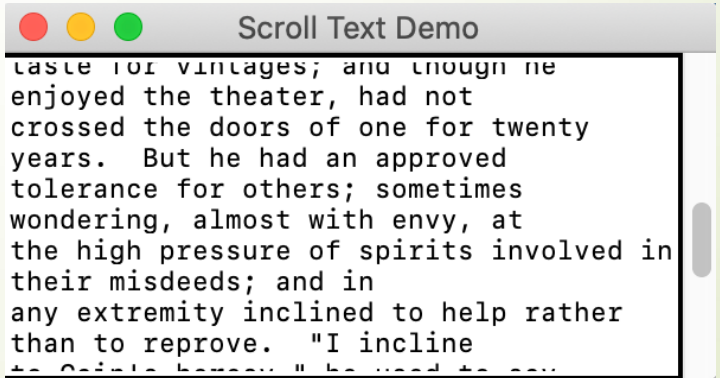


# 滚动条

滚动条小部件可用于垂直或水平滚动文本、画布或列表框小部件中的内容。

```
class ScrollText:
    def __init__(self):
        window = Tk()
        window.title("Scroll Text Demo")

        frame1 = Frame(window)
        frame1.pack()
        scrollbar = Scrollbar(frame1)
        scrollbar.pack(side = RIGHT, fill = Y)
        text = Text(frame1, width = 40, height = 10, wrap = WORD,
                    yscrollcommand = scrollbar.set)
        text.pack()
        scrollbar.config(command = text.yview)

        window.mainloop()
```



# 标准对话框

使用标准对话框显示消息框或提示用户输入数字和字符串。

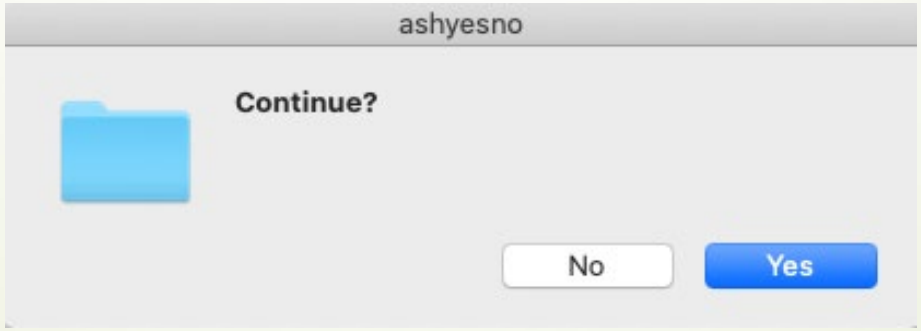
```
import tkinter.messagebox

tkinter.messagebox.showinfo("showinfo", "This is an info msg")
```


```
tkinter.messagebox.showwarning("showwarning", "This is a warning")
```

# 标准对话框

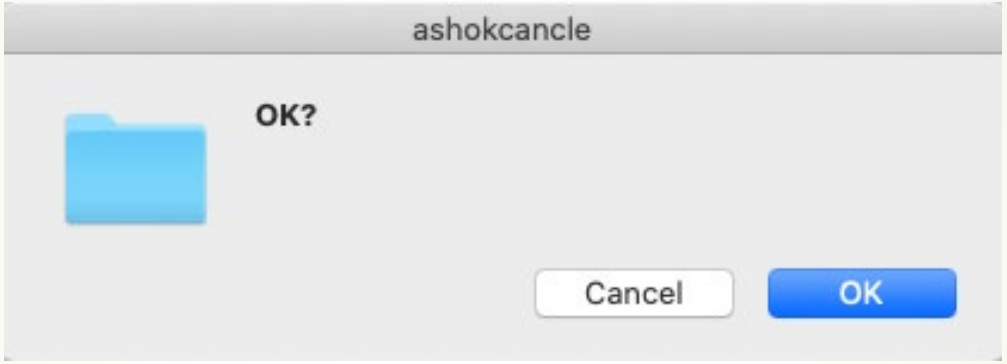
```
tkinter.messagebox.showerror("showerror", "This is an error")
```


```
isYes = tkinter.messagebox.askyesno("askyesno", "Continue?")
```

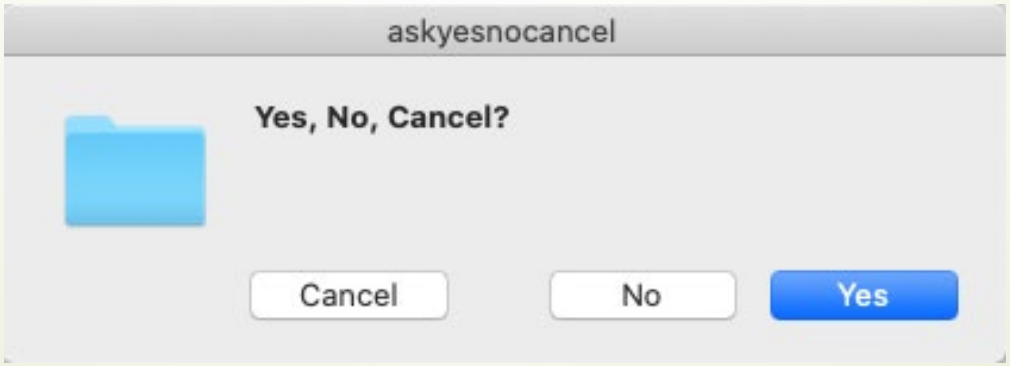


# 标准对话框

```
isOK = tkinter.messagebox.askokcancel("ashokcancel", "OK?")
```

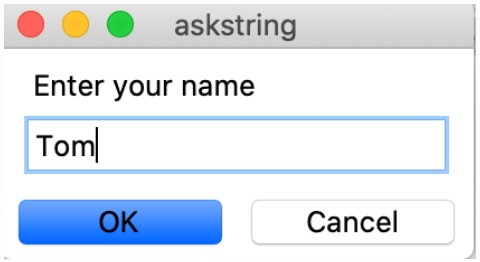



```
isYesNoCancel = tkinter.messagebox.askyesnocancel(
    "askyesnocancel", "Yes, No, Cancel?")
```

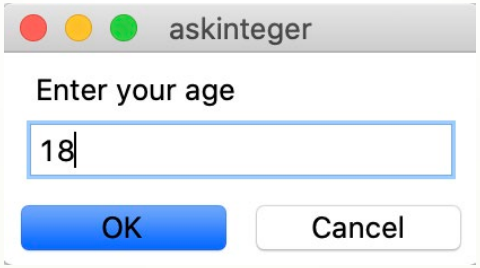


# 标准对话框

```
import tkinter.simpledialog
name = tkinter.simpledialog.askstring("askstring", "Enter your name")
```




```
age = tkinter.simpledialog.askinteger("askinteger", "Enter your age")
```




```
weight = tkinter.simpledialog.askfloat("askfloat", "Enter your weight")
```