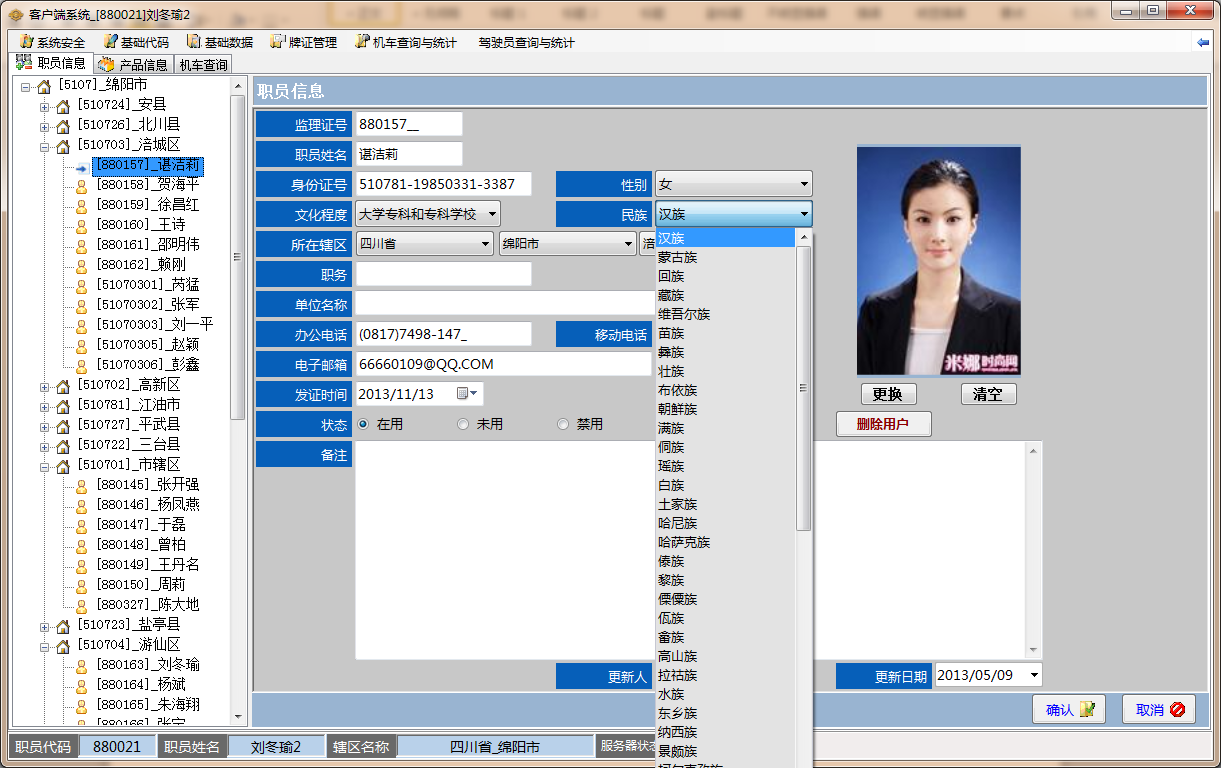
**[tkinter学习系列](https://www.cnblogs.com/zyg123/p/10351034.html)**

**

**

**python tkinter 全部组件（widget）及事件类型（event）一览**

对于一个简单的GUI程序设计来说，我觉得无非就是三个要素，**widget（部件），layout（布局），event（事件的响应）**

GUI就和搭积木差不多，

widget就好比是各种形状的积木，比如长方形，正方形，三角形等等。

layout就是如何将这些积木搭建成为一个小房子，小木屋。

widget（部件）+event（事件的响应）

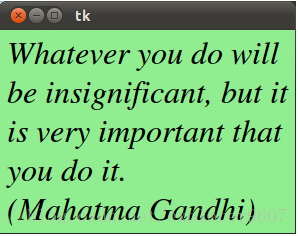
**tkinter中的widget主要有**

1. Button（按钮）,
2. Checkbutton（复选按钮）,
3. Canvas（画布），
4. Entry（条目）,
5. Frame（框架）,
6. Label（标签）,
7. LabelFrame（标签框架）,
8. Listbox（列表框），
9. menu（菜单），
10. Menubutton（菜单按钮）,
11. Message （消息），
12. OptionMenu（选项菜单），
13. PanedWindow（中分栏窗口）,
14. Radiobutton（单选按钮）,
15. Scale（刻度条）,
16. Scrollbar（滚动条），
17. Spinbox（整数调节框），
18. Text（文本框），
19. Combobox（下拉列表框）,
20. Notebook（笔记本）,
21. Progressbar（进度条）,
22. Separator（分离器）,
23. Sizegrip（尺寸调节器）,
24. Treeview（树视图）

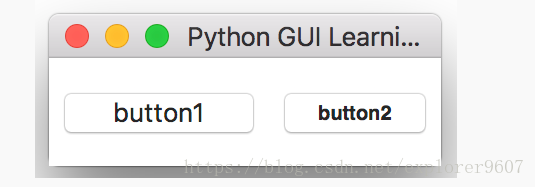
**部件样式**



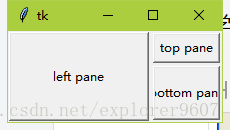
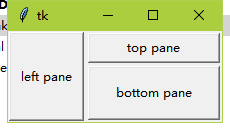
整体框架



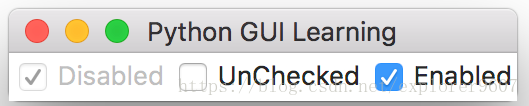
Message:消息控件；用来显示多行文本，与label比较类似



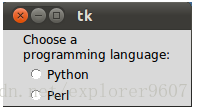
Button:按钮控件；在程序中显示按钮。



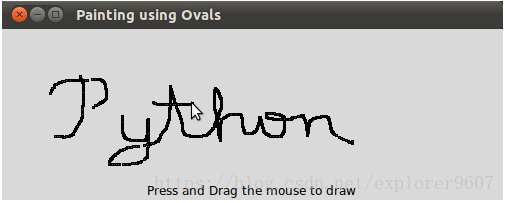
PanedWindow:窗口布局管理的插件，可以包含一个或者多个子控件



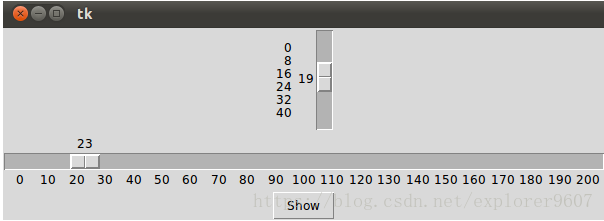
Checkbutton:多选框控件；用于在程序中提供多项选择框



Radiobutton:单选按钮控件；显示一个单选的按钮状态



Canvas:画布控件；显示图形元素如线条或文本

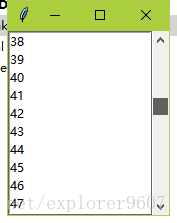


Scale:范围控件；显示一个数值刻度，为输出限定范围的数字区间

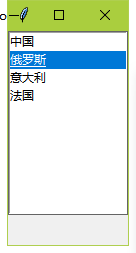


Label:标签控件；可以显示文本和位图

Entry:输入控件；用于显示简单的文本内容



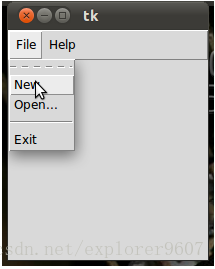
Scrollbar:滚动条控件，当内容超过可视化区域时使用，如列表框。



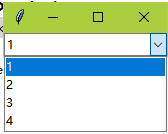
Listbox:列表框控件；在Listbox窗口小部件是用来显示一个字符串列表给用户



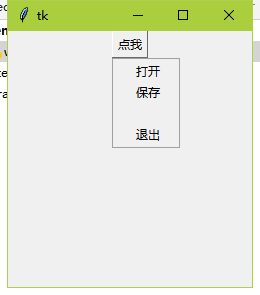
Text:文本控件；用于显示多行文本



menu:菜单控件；显示菜单栏,下拉菜单和弹出菜单



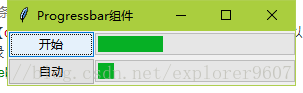
Combobox:下拉列表



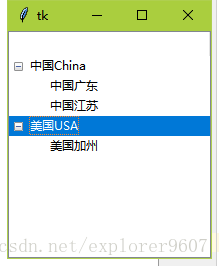
Menubutton:菜单按钮控件，由于显示菜单项。

https://img-blog.csdn.net/20180921200542880?watermark/2/text/aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L2V4cGxvcmVyOTYwNw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70

Notebook：提供一个区域，用户可以通过单击该区域顶部的选项卡来选择内容页面



Progressbar：进度条



Treeview：

**event的类型主要有**

对每一个组件来说，可以通过bind()的方法来将自己定义的函数或方法绑定到具体的事件上,event的类型主要有

|  |  |
| --- | --- |
| **事件** | **说明** |
| **Active** | 当组件的状态从“未激活”变为“激活”的时候触发该事件 |
| **Button** | 当用户点击鼠标按键的时候触发该事件 |
| detail部分指定是具体用哪个键：<Button-1>鼠标左键，<Button-2>鼠标中键（滚轮点击），<Button-3>鼠标右键，<Button-4>滚轮上滚（Linux），<Button-5>滚轮下滚（Linux） |
| **ButtonRelease** | 当用户释放鼠标按键的时候触发该事件 |
| 在大多数情况下，比Button要更好使用，因为如果当用户不小心按下鼠标键，用户可以将鼠标移出组件再释放鼠标，从而避免不小心触发事件 |
| **Configure** | 当组件的尺寸改变的时候触发该事件（窗口管理器触发的重绘事件，当你调整组件的尺寸或者移动应用程序，组件会和窗口一样被重绘） |
| **Deactivate** | 当组件的状态从“激活”变为“未激活”的时候触发该事件 |
| Destroy当组件被销毁时触发该事件 |
| **Enter** | 当鼠标指针进入组件的时候触发该事件 |
| 注意：不是用户按下回车键（回车键是Return<Key-Return>） |
| **Expose** | 当窗口或组件的某部分不再被覆盖的时候触发该事件 |
| **FocusIn** | 当组件获得焦点的时候触发该事件 |
| 用户可以用Tab键将焦点转移到该组件上（需要该组件的takefocus选项为True） |
| 你也可以调用focus\_set()方法使该组件获得焦点 |
| **FocusOut** | 当组件失去焦点的时候触发该事件 |
| **KeyPress** | 当用户按下键盘按键的时候触发该事件 |
| detail可以指定具体的按键，例如<KeyPress-H>表示当大写字母H被按下的时候触发该事件 |
| KeyPress可以缩写为Key |
| **KeyRelease** | 当用户释放键盘按键的时候触发该事件 |
| **Leave** | 当鼠标指针离开组件的时候触发该事件 |
| **Map** | 当组件被映射的时候触发该事件 |
| 意思是在应用程序中显示该组件的时候，例如调用get()方法 |
| **Motion** | 当鼠标在组件内移动的时候触发该事件 |
| **MouseWheel** | 当鼠标滚轮滚动的时候触发该事件 |
| 目前该事件仅支持Windows和Mac系统 |
| **Unmap** | 当组件被取消映射的时候触发该事件 |
| 意思是在应用程序中不再显示该组件的时候，例如调用grid\_remove()方法 |
| **Visibility** | 当应用程序至少有一部分在屏幕中是可见的时候触发该事件 |

**layout（布局）**

这部分将介绍tkinter中的几种布局方式:Grid,Place,pack

布局方式：

                       pack：方位布局

                       place：坐标布局

                       gird：网格布局

**部件分类**

  按钮：

                       Button

                       RadioButton

                       CheckButton

                       Listbox

  文本输入组件：

                       Entry：单行

                       Text：多行

  标签：

                       label：显示图片和文字

                       Message：根据内容文字换行

  菜单：

                       Menu

                       MenuButton

  滚动条：

                       scale：滑块组件

                       Scrollbar滚动条组件

  其他组件：

                       Canvas：画布组件

                       Frame：框架组件，将多个组件分组

                       Toplevel：创建子窗口容器组件

**02窗口的设置\_第三周**

**(一)窗体的最小框架**

**1.说明：**

*需要导入标准库：  
import tkinter*

**2.源代码：**

# 导入模块，并取别名

import tkinter as tk

# 实例化一个窗体对象

win = tk.Tk()

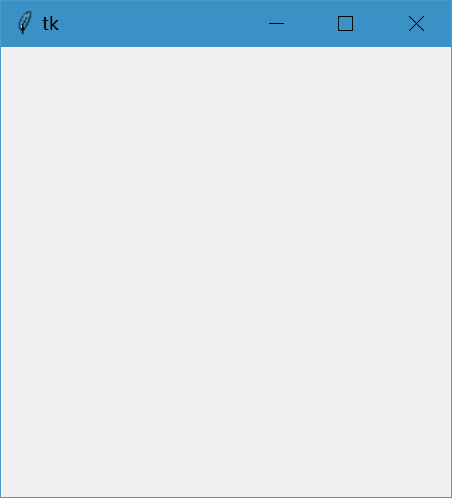
# 设置窗体的大小(300x300)，与出现的位置距离窗体左上角(+150+150)

win.geometry("300x300+150+150")

# 进入消息循环，显示窗体

win.mainloop()

**3.实现效果：**



**(二)窗体的基本设置**

**1.说明：**

**(1)添加标题：**

*# 设置标题*

*win.title("标题")*

**(2)添加图标：**

*图标：必须是以 .ico 为后缀的图标文件，放于同目录下。*

*# 设置图标，以QQ头为例*

*win.iconbitmap("app.ico")*

**(3)添加背景：**

*颜色：可以用英文名，也可以用RGU，十六进制表示的颜色。*

*# 设置背景色，以“蓝色”为例*

*win["background"] = "blue"*

**2.完整代码：**

# 导入模块，并去别名

import tkinter as tk

# 实例化一个窗体对象

win = tk.Tk()

# 设置窗体的大小(300x300)，与出现的位置距离窗体左上角(+150+150)

win.geometry("300x300+150+150")

# 设置标题

win.title("标题")

# 设置图标，以QQ头为例

win.iconbitmap("app.ico")

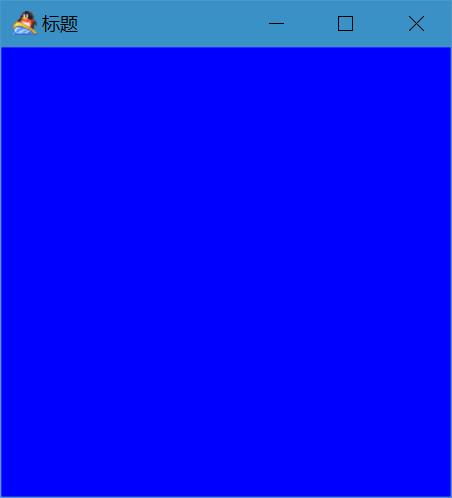
# 设置背景色，以“蓝色”为例

win["background"] = "blue"

# 进入消息循环，显示窗体

win.mainloop()

**3.实现效果：**



**(三)窗体的外形设置**

**1.说明：**

**(1)设置透明度：**

*透明度的值是：0~1 可以是小数点，0：全透明；1：全不透明*

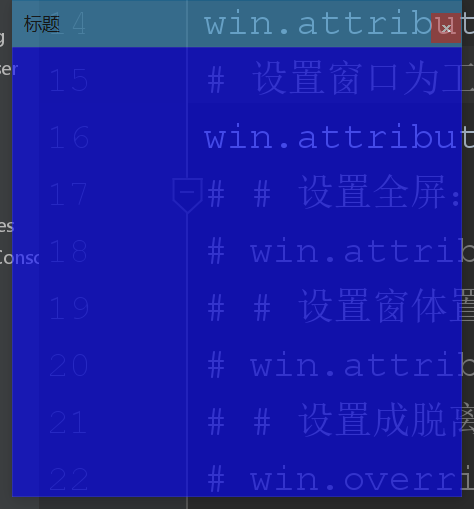
*win.attributes("-alpha", 0.6)*

**(2)设置工具栏样式：**

*工具样式：True 只有退出按钮，也没有图标； False 正常的窗体样式*

*win.attributes("-toolwindow", True)*

透明度与工具栏的效果图：



**(3)设置全屏：**

*全屏模式：True 全屏；False 正常显示*

*win.attributes("-fullscreen", True)*

**(4)设置窗体置顶**

*窗体置顶：True 所有窗口中处于最顶层；False 正常显示*

*win.attributes("-topmost", True)*

**(5)设置成脱离工具栏**

*脱离工具栏：True 没有工具栏按钮；False 正常显示*

*win.overrideredirect(True)*

脱离工具栏的效果图：



**2.完整代码：**

# 导入模块，并去别名

import tkinter as tk

# 实例化一个窗体对象

win = tk.Tk()

# 设置窗体的大小(300x300)，与出现的位置距离窗体左上角(+150+150)

win.geometry("300x300+150+150")

# 设置标题

win.title("标题")

# 设置图标，以QQ头为例

win.iconbitmap("app.ico")

# 设置背景色，以“蓝色”为例

win["background"] = "blue"

# 设置透明度

win.attributes("-alpha", 0.6)

# 设置窗口为工具样式：

win.attributes("-toolwindow", True)

# # 设置全屏：

win.attributes("-fullscreen", False)

# 设置窗体置顶

win.attributes("-topmost", True)

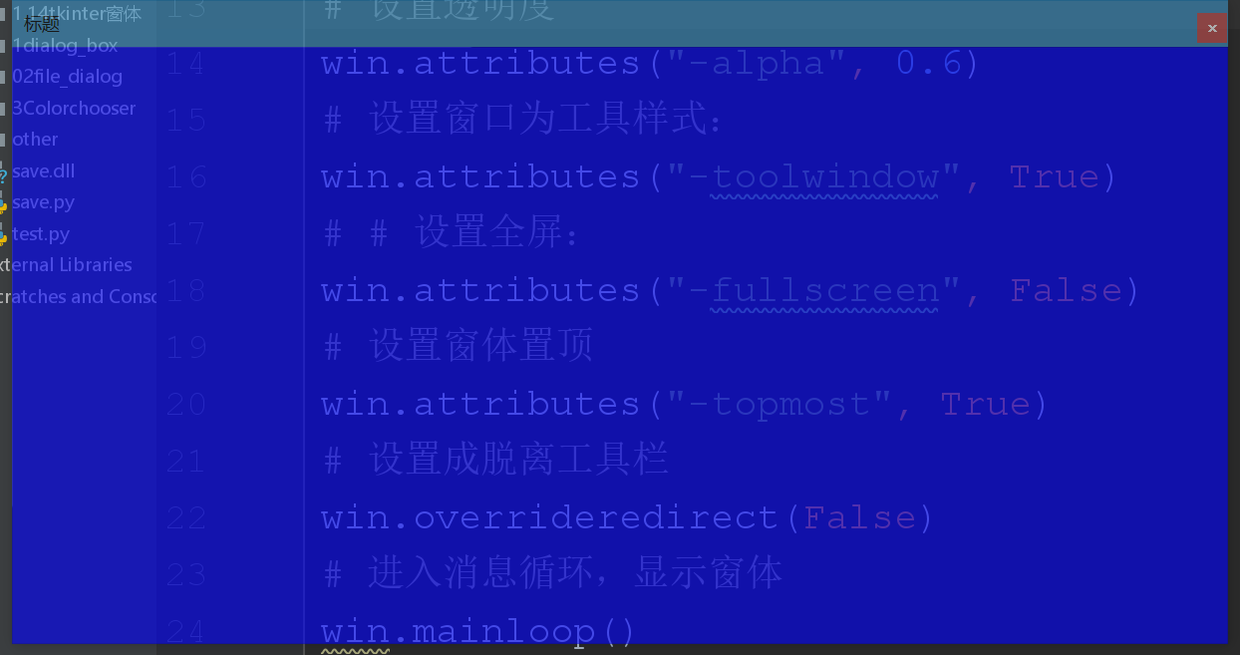
# 设置成脱离工具栏

win.overrideredirect(False)

# 进入消息循环，显示窗体

win.mainloop()

**3.实现效果：**



**(四)窗体的三个方法：**

**1.获取屏幕的大小**

**(1)语法说明：**

*# 获取屏幕的大小;*

*screen\_height = root.winfo\_screenheight()*

*screen\_width = root.winfo\_screenwidth()*

**(2)源代码：**

import tkinter as tk

root = tk.Tk()

# 获取屏幕的大小;

screen\_height = root.winfo\_screenheight()

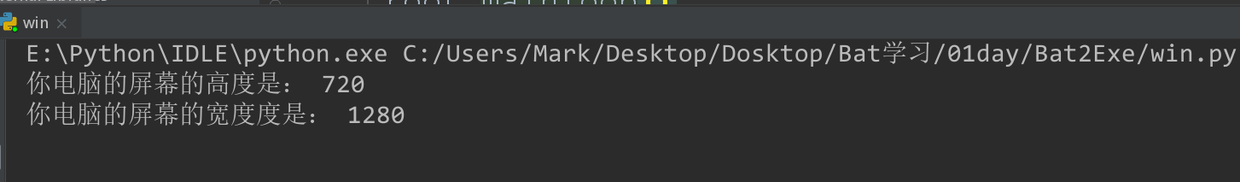
screen\_width = root.winfo\_screenwidth()

print("你电脑的屏幕的高度是：", screen\_height)

print("你电脑的屏幕的宽度度是：", screen\_width)

root.mainloop()

**(3)输出结果：**



**2.获取窗体的大小**

**(1)语法说明：**

*# 获取屏幕的大小;*

*win\_height = root.winfo\_height()*

*win\_width = root.winfo\_width()*

*需要注意的是：要用root.update()方法，才能看到更新后的数值*

**(2)源代码：**

import tkinter as tk

root = tk.Tk()

root.geometry("600x500")

# 更新窗体

root.update()

# 获取屏幕的大小;

win\_height = root.winfo\_height()

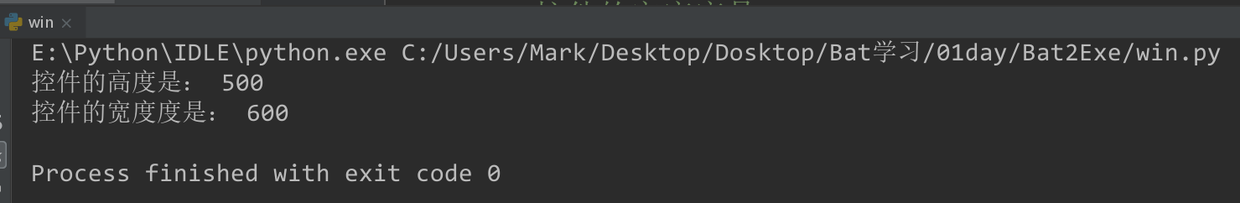
win\_width = root.winfo\_width()

print("控件的高度是：", win\_height)

print("控件的宽度度是：", win\_width)

root.mainloop()

**(3)输出结果：**



**3.获取窗体的位置**

**(1)语法说明：**

*# 获取窗体的位置*

*win\_x = win.winfo\_x()*

*win\_y = win.winfo\_y()*

**(2)源代码：**

import tkinter as tk

win = tk.Tk()

win.geometry("600x500")

def change(event):

win.update()

# 获取窗体的位置

win\_x = win.winfo\_x()

win\_y = win.winfo\_y()

print(win\_x)

print(win\_y)

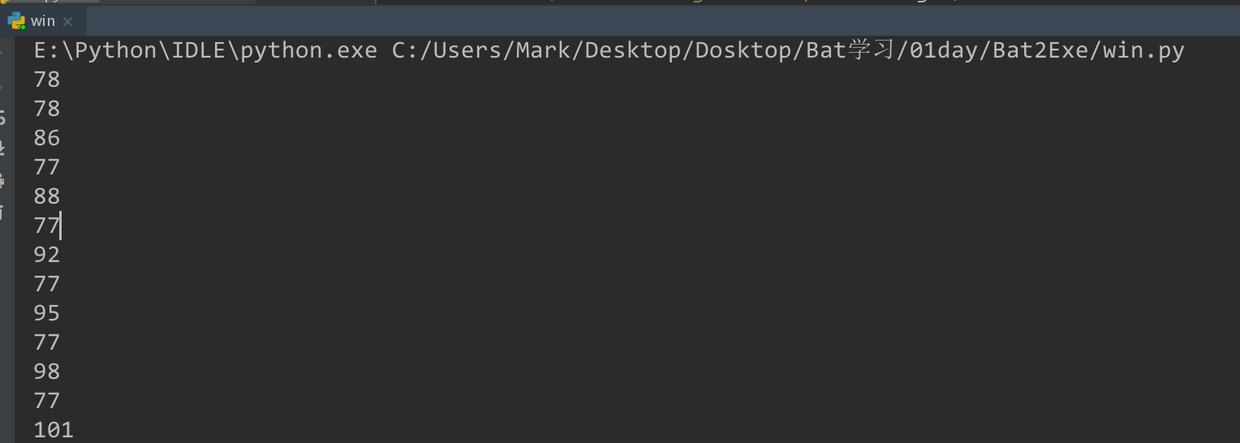
# 绑定事件，窗体的改变事件

win.bind("<Configure>", change)

win.mainloop()

**(3)输出结果：**

*当我用鼠标拖动窗体时，更新了一系列的位置数值*



分类: [Python-GUI篇之tkinter](https://www.cnblogs.com/zyg123/category/1394759.html)

**H\_01字体类型**

Tkinter字体类型根据Python使用的平台，可能会有三种方法来指定类型的风格。

1）作为一个元组的第一个元素是字体，紧随其后的是分大小，可选地紧随其后　　字符串包含一个或多个风格修饰符粗体、斜体、下划线、和超调。

例如：(“Helvetica”, “16”)和(“Times”, “24”, “bolditalic”)。

2）可以导入字体对象创建一个tkFont模块和使用它的字体类构造函数。

**import tkinter.font as tkFont #导入Tkinter字体模块**

创建字体对象的语法格式如下。

**font = tkFont.Font ( option, ... )**

创建方法中的 **option** 选项参数见下表。

|  |  |
| --- | --- |
| 参数 | 说明 |
| family | 字体名称为字符串。 |
| size | 字体高度为整数点。获得字体n像素高，使用 - n。 |
| **weight** | 字体宽度。“bold” 表示加粗， "normal"表示正常字体宽度。 |
| **slant** | 字体倾斜。“italic” 表示斜体, "roman"表示不斜的。 |
| **underline** | 下划线。1表示下划线文本，0表示正常。 |
| **overstrike** | 划线字体。1表示划线文本，0表示正常。 |

tkFont除了选项参数外，还有一些方法。

显示所有可用字体，可以使用如下方法。

tkFont.families()

对于创建的字体对象font，还有如下方法可以使用。

**1）返回字体对象全部或一个属性值。**

font.actual ( option=None )

如果你不传递任何参数，返回字体对象实际属性，字典数据。

**2）返回给定的字体对象属性值。**

font.cget ( option )

**3）改变字体对象一个或多个属性值。**

font.configure ( option, ... )

**使用这种方法来改变一个或多个选项字体。**使用这个字体对象的小部件字体也会随之改变。

**4）返回一个字体对象的副本。**

font.copy()

**5）返回使用这个字体对象的字符串像素宽度。**

font.measure ( text )

通过这个方法将返回字符串的像素宽度。注意：有些倾斜字符可能延长到这个像素宽度区域以外。

**6）回返字体度量。**

font.metrics ( option )

如果你不带参数调用这个方法，它返回一个字典的所有字体度量。可选option参数含义见下表。

|  |  |
| --- | --- |
| 参数 | 说明 |
| ascent | 高度的像素数量之间的基线和上升最高的顶部。 |
| descent | 高度的像素数量之间的基线和最低的人或物的底部。 |
| fixed | 此值对于变幅字体为0，对于单幅字体为1。 |
| linespace | 高度的像素数量。 |

linespace 高度的像素数量。

下面给出一个Tkinter字体对象示例程序。

import tkinter as tk  #装载tkinter模块,用于Python3

import  tkinter.font   as  tkFont  #导入Tkinter字体模块

root=tk.Tk()  #创建Tkinter主窗口

root.title("Tkinter字体演示")

helv36 = tkFont.Font ( family="黑体",size=36, weight="bold",slant= "italic"  ,underline=1) #创建字体对象

w = tk.Label(root, text="Tkinter字体演示", font=helv36) #创建Labkel部件

w.pack()  #放置部件

f = tkFont.Font(family="times", size=30, weight="normal",slant= "roman",overstrike=1 )

w = tk.Label(root, text="Hello, world", font=f)

w.pack()

w = tk.Button(root, text="Quit!", command=root.destroy)

w.pack()

fb = tkFont.Font(font=w["font"]).copy()

fb.config(weight="bold")

fb.config(size=20)

w.config(font=fb)

root.mainloop()



[**03 Label控件**](https://www.cnblogs.com/zyg123/p/10351055.html)**\_第三周**

**前言**

Label是用于显示文本或图像的控件。

**(一)基本用法和可选属性**

**1.基本用法**

基本用法：Label(*根*对象, [属性列表])

父对象：在那个窗体显示，例如主窗体。

属性列表：是可选的属性=属性值组成。

**2.可选属性**

|  |  |
| --- | --- |
| **属性** | **说明** |
| **text** | **标签显示的文本** |
| **font** | **设置文本的字体和大小** |
| **fg(foreground)** | **字体的颜色，** |
| **bg (background)** | **标签的背景色** |
| **width** | **标签的宽度(一个中文的字体宽为单位)** |
| **height** | **标签的高度(一个中文的字体高为单位)** |
| **padx** | **文字到边框的距离，水平方向** |
| **pady** | **文字到边框的距离，垂直方向** |
| **bd(borderwidth)** | **边框的宽度** |
| **relief** | **边框的样式** |
| **justify** | **文本对齐方式** |
| **image** | **图片** |
| **compound** | **图片与文字的混搭** |
| **anchor** | **方位** |

**justify:**      **指定多行的对齐方式**，必须是 left, right, or center

**anchor：**       指定文本(text)或图像(bitmap/image)在Label中的显示位置（方位）

可用的值：

e、w、n、s、ne、se、sw、sn、center

布局如下图：类似空间平面中的8个方位

                nw（左上）        n （中上）      ne（右上）

                w  （左中）   center（中间）   e（右中）

                sw（左下）        s （中下）      se（右下）

**(二)属性的具体使用**

**1.常用属性**

**(1)font**

font:设置字体与字体的大小

用法：font=("字体名"，大小) 例如：font=(“黑体”， 20)

|  |  |
| --- | --- |
| 参数 | 说明 |
| family | 字体名称为字符串。 |
| size | 字体高度为整数点。获得字体n像素高，使用 - n。 |
| weight | 字体宽度。“bold” 表示加粗， "normal"表示正常字体宽度。 |
| slant | 字体倾斜。“italic” 表示斜体, "roman"表示不斜的。 |
| underline | 下划线。1表示下划线文本，0表示正常。 |
| overstrike | 划线字体。1表示划线文本，0表示正常。 |

**(2)fg 与 bg**

fg 前景色，也就是字体的颜色，bg 背景颜色

用法：fg="red", fg="#121234"

**(3)width 与 height**

width height 标签的宽度与高度，都是以系统默认的中文的一个**字体宽高为单位**

用法：width = 5, height=2

**2.边距与文本对齐方式**

**(1)边距：padx 与 pady**

*文字到边框的距离 padx 与 pady 水平与竖直方向*

*用法：padx=2, pady = 1 ,单位是像素，默认是1像素*

**(2)文本对齐方式：justify**

*Label 里的文本支持 转移字符 \n \t 等*

*用法：justify="left" ,可取 "left" ,"right", "center" ,默认是"center"居中对齐*

**案例一**

import tkinter as tk

# 背景色与前景色

win = tk.Tk()

label\_1 = tk.Label(win, text='fg="red" bg="blue"', fg="red", bg="blue")

label\_1.pack()

# 字体与字体的大小

label\_2 = tk.Label(win, text='font=("微软雅黑", 20)', font=("微软雅黑", 20))

label\_2.pack()

# 宽度与高度

label\_3 = tk.Label(win, text='width=10, height=3', bg="yellow", width=10, height=3)

label\_3.pack()

label\_4 = tk.Label(win, text='width=20, height=1', bg="yellow", width=20, height=1)

label\_4.pack()

# 边距

label\_5 = tk.Label(win, text='padx pady 默认', bg="blue",fg="white")

label\_5.pack()

label\_6 = tk.Label(win, text='padx=0, pady=0', bg="blue",fg="white", padx=0, pady=0)

label\_6.pack()

label\_7 = tk.Label(win, text='padx=10, pady=10', bg="blue",fg="white", padx=10, pady=10)

label\_7.pack()

# 对齐方式

label\_8 = tk.Label(win, text="我是张一根，一匹想飞的狼，\n999号哦")

label\_8.pack()

label\_9 = tk.Label(win, text="我是张一根，一匹想飞的狼，\n999号哦", justify="left")

label\_9.pack()

label\_9 = tk.Label(win, text="我是张一根，一匹想飞的狼，\n999号哦", justify="right")

label\_9.pack()

win.mainloop()

**案例一的效果**



**3.Label里放图片**

**(1)只放图片，没有文字**

*需要先导入图片的路径：img1 = tk.PhotoImage(file="image/01.png")*

*再使用：image=img1*

*注：目前支持 .png 与 .gif 格式， 还不支持 .jpg格式*

**案例二**

import tkinter as tk

win = tk.Tk()

# 导入图片

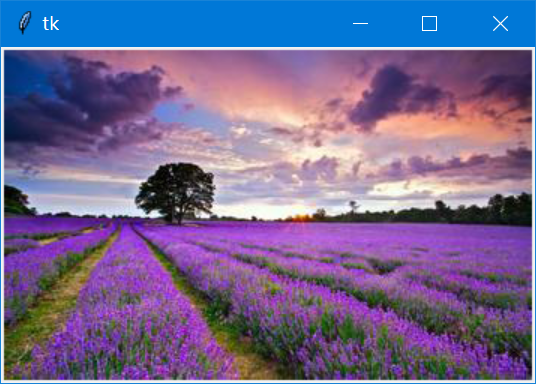
img1 = tk.PhotoImage(file="image/01.png")

# 在标签里放入图片

label\_image1 = tk.Label(win, image=img1)

label\_image1.pack()

win.mainloop()

**案例二的效果**

**(2)图片与文字混搭**

*需要使用：compound="对齐方式"，*

*对齐方式有：'left', "right", "center"*

**案例三**

import tkinter as tk

win = tk.Tk()

# 导入图片

img1 = tk.PhotoImage(file="image/01.png")

# 在标签里放入图片

label\_image1 = tk.Label(win, text="图文并茂", image=img1, compound="left")

label\_image1.pack()

label\_image2 = tk.Label(win, text="图文并茂", image=img1, compound="right")

label\_image2.pack()

label\_image3 = tk.Label(win, text="图文并茂", image=img1, compound="center", fg="red")

label\_image3.pack()

win.mainloop()

**案例三的效果**



**4.Label可变的文字内容**

***先定义一个：tk变量，v = tk.StringVar()***

***在使用：textvariable=v***

**案例四**

import tkinter as tk

import time

# 背景色与前景色

win = tk.Tk()

# 定义变量,必须是使用的是tk里的这个变量

v = tk.StringVar()

# 使用v.set("string") 来定义变量

v.set("这是刚开始的文字")

# 在标签里添加可变的文本

label\_1 = tk.Label(win, textvariable=v, fg="red", bg="green", font=("黑体", 40))

label\_1.pack()

while True:

v.set("%s" % time.ctime())

label\_1.update()

time.sleep(1)

win.mainloop()

**案例四的效果**

*可以制作一个时间表*



**4.文字所在标签框的方位**

*anchor 可选的参数有："e","w","s","n","ne","se","nw","sw",就是东西南北八个方向*

*anchor="n"*

**案例五**

import tkinter as tk

win = tk.Tk()

label\_1 = tk.Label(win, text="方位是 N", bg="green", width=10, height=3, anchor="n")

label\_1.pack()

label\_2 = tk.Label(win, text="方位是 NE", bg="blue", width=10, height=3, anchor="ne")

label\_2.pack()

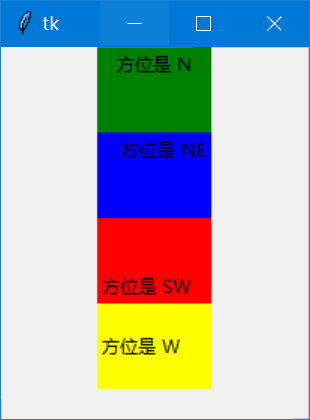
label\_3 = tk.Label(win, text="方位是 SW", bg="red", width=10, height=3, anchor=tk.SW)

label\_3.pack()

label\_4 = tk.Label(win, text="方位是 W", bg="yellow", width=10, height=3, anchor=tk.W)

label\_4.pack()

win.mainloop()

**案例五的效果**

**5.边框样式与宽度**

*relief 表示边框样式，可选的参数有：flat(平的)，sunken (沉没的, 凹下去的)，raised (提高，凸出来的)，ridge(脊，中键凸的)*

*relief="sunken", 默认是flat.*

**案例六**

import tkinter as tk

win = tk.Tk()

# 在标签里放入图片SUNKEN, RAISED, GROOVE, and RIDGE

# 默认 flat平的

label\_1 = tk.Label(win, text='relief=flat', bd=8)

label\_1.pack()

# sunken 沉没的

label\_2 = tk.Label(win, text='relief=sunken', relief="sunken", bd=2)

label\_2.pack()

# raised 提高

label\_3 = tk.Label(win, text='relief=raised', relief="raised", bd=2)

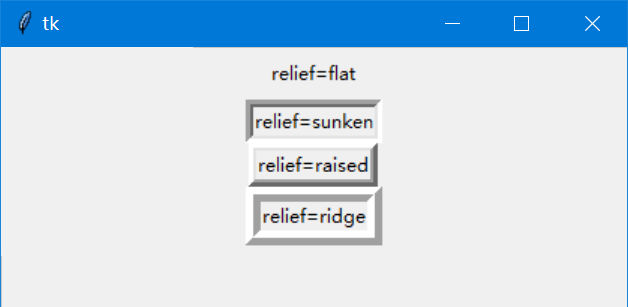
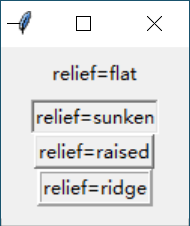
label\_3.pack()

# ridge脊

label\_4 = tk.Label(win, text='relief=ridge', relief="ridge", bd=3)

label\_4.pack()

win.mainloop()

**案例六的效果**

**1. 简介**

Label用于在指定的窗口中显示文本和图像。最终呈现出的Label是由背景和前景叠加构成的内容。

Label组件定义函数：Label（master=None, cnf={}, \*\*kw） 其中，kw参数是用来自定义lable组件的键值对。

**2. 背景自定义**

背景的话，有三部分构成：内容区+填充区+边框

<1>内容区参数有：width,length用于指定区域大小，如果显示前景内容是文本，则以单个字符大小为单位；如果显示的是图像，则以像素为单位。默认值是根据具体显示的内容动态调整。类型是int。

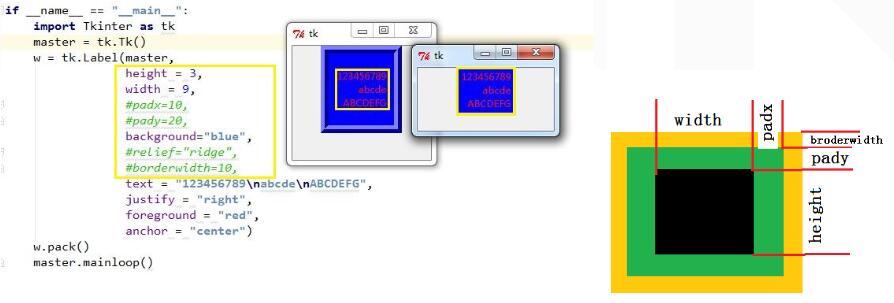
background用于指定背景的颜色，默认值根据系统而定。

<2>填充区参数:指的是内容区和边框之间的间隔大小，单位是像素。参数有：padx , pady，类型是int。

<3>边框参数：样式relief(可选值为：flat(默认),sunken,raised,groove,ridge)，borderwidth(边框的宽度，单位是像素，默认根据系统而定，一般是1或2像素)

highlightbackground,highlightcolor,highlightthickness 三个边框参数仅在Label允许接收焦点的情况下（tackfocus=True），用于设置焦点获取前后高亮边框颜色以及高亮边框宽度。

举个栗子（@-@）



上图右侧为，背景图构成：内容区（黑色），填充区（绿色），边框（黄色）

定义的背景内容区是可容纳3X9的字符区，如上图中的右侧小窗口中的Label。增加了填充区和边框后的效果如上图中左侧的Label。

**3. 前景自定义**

前景定义分为文本内容和图像两小块来说明。

3.1 文本

文本内容选项有：<1>指定字体和字体大小，如：font = (font\_name,size)，默认有系统指定。

<2>文本对齐方式，justify = "center(默认)left/right/"

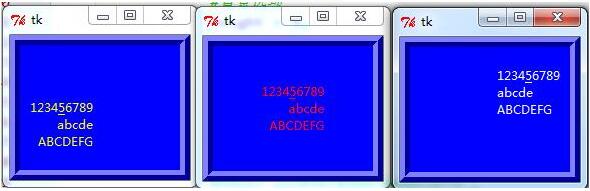
<3>指定文本（或图像）颜色，foreground = "指定的颜色"，可以是英文名字，也可以是RGB格式的

<4>指定文本内容：（静态的）text = "目标字符串....."； （动态更新的）textvariable = str\_obj，当str\_obg的内容改变时，会更新Label中对应内容。

这里需要注意的是str\_obj必须是TKinter所支持的字符串类型变量，如：str\_obj = Tkinter.StringVar() str\_obj.set("目标文本内容")

<5>单个字符添加下划线，underline = index， index是目标字符串中的字符索引值。

<6>文本或图像在背景内容区的位置：anchor 可选值为（n,s,w,e,ne,nw,sw,se,center）eswn是东南西北英文的首字母，表示：上北下南左西右东



图像内容选项有：<1>指定图片：bitmap = bitmap\_image，当指定image选项的时候，这个参数会被忽略掉 或者 image = normal\_image(仅支持GIF, PPM/PGM格式的图片)" 。

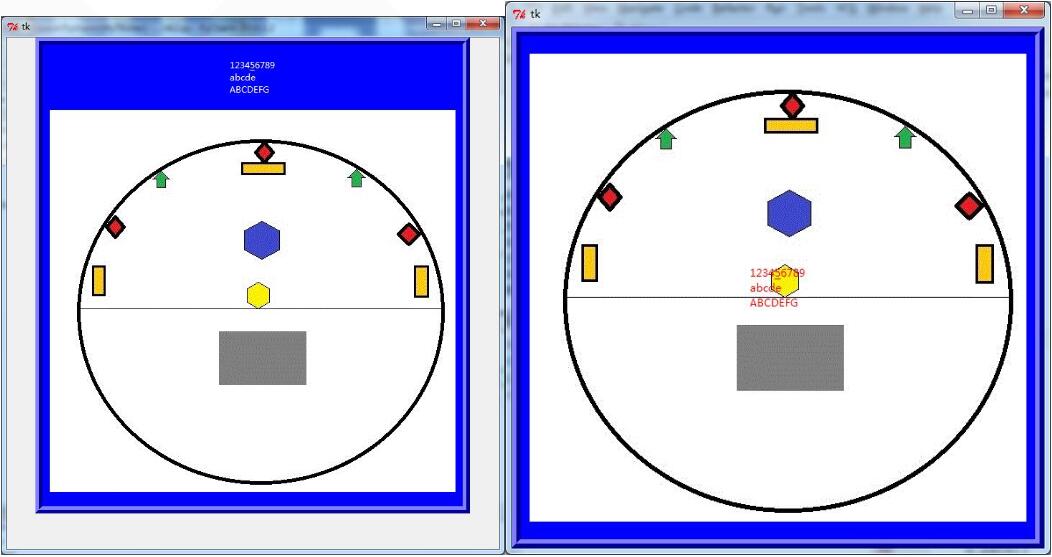
需要注意的是这里的所用到的图片对象bitmap\_image normal\_image都是需要经过TKinter转换后的图像格式。

如： bitmap\_image = TKinter.BitmapImage(file = "位图片路径")

normal\_image = TKinter.PhotoImage(file = "gif 、ppm/pgm图片路径")

图片和文本取舍：compound参数可以控制要显示的文本和图像。当同时指明了要显示的文本和图像时，可以通过该参数来进行不同设置。

可选值：None 默认值，表示只显示图像，不显示文本；bottom/top/left/right，表示图片显示在文本的下/上/左/右；center,表示文本显示在图片中心上方。



上图左中的compound="bottom"，表示图片显示在文字下方；上图左中的compound="center"，表示文字显示在图片中间上方

**4. Label的其他参数**

<1>activebacakground activeforground 用于设置Label处于活动（active）状态下的背景和前景颜色,默认由系统指定。

<2>diableforground 指定当Label不可用的状态（Disable）下的前景颜色，默认由系统指定。

<3>cursor 指定鼠标经过Label的时候，鼠标的样式，默认由系统指定。

<4>state 指定Label的状态，用于控制Label如何显示。可选值有：normal(默认)/active/disable。

程序源码

#coding=utf-8

import Tkinter as tk

if \_\_name\_\_ == "\_\_main\_\_":

  import Tkinter as tk

  master = tk.Tk()

  str\_obj = tk.StringVar()

  str\_obj.set("这是TKinter所支持的字符串类型")

  #bitmap\_image = tk.BitmapImage(file = "./tmp/11.bmp")

  normal\_image = tk.PhotoImage(file = "./tmp/11.gif")

  print type(normal\_image)

  print normal\_image

  w = tk.Label(master,

         #背景选项

         #height = 5,

         #width = 20,

         padx=10,

         pady=20,

         background="blue",

         relief="ridge",

         borderwidth=10,

         #文本

         text = "123456789\nabcde\nABCDEFG",

         #textvariable = str\_obj,

         justify = "left",

         foreground = "white",

         underline = 4,

         anchor = "ne",

         #图像

         image = normal\_image,

         compound = "bottom",

         #接受焦点

         #takefocus = True,

         #highlightbackground = "yellow",

         #highlightcolor = "white",

         #highlightthickness = 5

         )

  w.pack()

  master.mainloop()

**04 Entry \*\*\*\_第三周**

Entry（输入框）组件通常用于获取用户的输入文本。

**何时使用 Entry 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（你可以用鼠标或键盘的方向键调整文本的可见范围）。

如果你希望接收多行文本的输入，可以使用 Text 组件（后面介绍）。

**用法**

使用代码为 Entry 组件添加文本，可以使用 insert() 方法。如果要替换当前文本，可以先使用 delete() 方法，再使用 insert() 方法实现：

import tkinter as tk

master = tk.Tk()

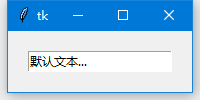
e = tk.Entry(master)

e.pack(padx=20, pady=20)

e.delete(0, "end")

e.insert(0, "默认文本...")

master.mainloop()



获取当前输入框的文本，可以使用 get() 方法：

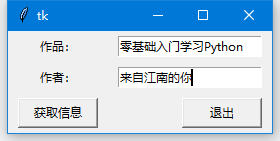
s = e.get()

也可以**绑定 Entry 组件到 Tkinter 变量（StringVar）**，并通过该变量设置和获取输入框的文本：

1. v = tk.StringVar()
2. e = tk.Entry(master, textvariable=v)
3. e.pack()
5. v.set("I love Python!")
6. s = v.get()

下边的例子演示将 Entry 组件和 Button 组件配合，点击 “获取信息” 按钮时自动清空输入框并将内容输出：

1. import tkinter as tk
3. master = tk.Tk()
5. tk.Label(master, text="作品：").grid(row=0)
6. tk.Label(master, text="作者：").grid(row=1)
8. e1 = tk.Entry(master)
9. e2 = tk.Entry(master)
10. e1.grid(row=0, column=1, padx=10, pady=5)
11. e2.grid(row=1, column=1, padx=10, pady=5)
13. def show():
14. print("作品：《%s》" % e1.get())
15. print("作者：%s" % e2.get())
16. e1.delete(0, "end")
17. e2.delete(0, "end")
19. tk.Button(master, text="获取信息", width=10, command=show).grid(row=3, column=0, sticky="w", padx=10, pady=5)
20. tk.Button(master, text="退出", width=10, command=master.quit).grid(row=3, column=1, sticky="e", padx=10, pady=5)
22. master.mainloop()



最后需要提到的是 Entry 组件允许通过以下几种方式指定字符的位置：

**数字索引号**

"anchoe"

"end"

"insert"

鼠标坐标（"@x"）

数字索引号：常规的 Python 索引号，从 0 开始

"anchor"：对应第一个被选中的字符（如果有的话）

"end"：对应已存在文本的后一个位置

"insert"：对应插入光标的当前位置

鼠标坐标（"@x"）：x 是鼠标位置与 Entry 左侧边缘的水平距离，这样就可以通过鼠标相对地定位字符的位置

**Entry参数**

**Entry(master=None, \*\*options) (class)**

master -- 父组件

\*\*options -- 组件选项，下方表格详细列举了各个选项的具体含义和用法：

|  |  |
| --- | --- |
| **选项** | 含义 |
| **background** | 1. 设置 Entry 的背景颜色 |
| 2. 默认值由系统指定 |
| **bg** | 跟 background 一样 |
| **borderwidth** | 1. 设置 Entry 的边框宽度 |
| 2. 默认值是 1 或 2 像素 |
| **bd** | 跟 borderwidth 一样 |
| **cursor** | 1. 指定当鼠标在 Entry 上飘过的时候的鼠标样式 |
| 2. 默认值由系统指定 |
| **exportselection** | 1. 指定选中的文本是否可以被复制到剪贴板 |
| 2. 默认值是 True |
| 3. 可以修改为 False 表示不允许复制文本 |
| **font** | 1. 指定 Entry 中文本的字体 |
| 2. 默认值由系统指定 |
| **foreground** | 1. 设置 Entry 的文本颜色 |
| 2. 默认值由系统指定 |
| **fg** | 跟 foreground 一样 |
| **highlightbackground** | 1. 指定当 Entry 没有获得焦点的时候高亮边框的颜色 |
| 2. 默认值由系统指定 |
| **highlightcolor** | 1. 指定当 Entry 获得焦点的时候高亮边框的颜色 |
| 2. 默认值由系统指定 |
| **highlightthickness** | 1. 指定高亮边框的宽度 |
| 2. 默认值是 1 或 2 像素 |
| **insertbackground** | 指定输入光标的颜色 |
| **insertborderwidth** | 1. 指定输入光标的边框宽度 |
| 2. 如果被设置为非 0 值，光标样式会被设置为 RAISED |
| 3. 小甲鱼温馨提示：将 insertwidth 设置大一点才能看到效果哦 |
| **insertofftime** | 1. 该选项控制光标的闪烁频率（灭） |
| 2. 单位是毫秒 |
| **insertontime** | 1. 该选项控制光标的闪烁频率（亮） |
| 2. 单位是毫秒 |
| **insertwidth** | 1. 指定光标的宽度 |
| 2. 默认值是 1 或 2 像素 |
| **invalidcommand** | 1. 指定当输入框输入的内容“非法”时调用的函数 |
| 2. 也就是指定当 validateCommand 选项指定的函数返回 False 时的函数 |
| 3. 详见本内容最下方小甲鱼关于验证详解 |
| **invcmd** | 跟 invalidcommand 一样 |
| **justify** | 1. 定义如何对齐输入框中的文本 |
| 2. 使用 "left"，"right" 或 "center" |
| 3. 默认值是 "left" |
| **relief** | 1. 指定边框样式 |
| 2. 默认值是 "sunken" |
| 3. 其他可以选择的值是 "flat"，"raised"，"groove" 和 "ridge" |
| **selectbackground** | 1. 指定输入框的文本被选中时的背景颜色 |
| 2. 默认值由系统指定 |
| **selectborderwidth** | 1. 指定输入框的文本被选中时的边框宽度（选中边框） |
| 2. 默认值由系统指定 |
| **selectforeground** | 1. 指定输入框的文本被选中时的字体颜色 |
| 2. 默认值由系统指定 |
| **show** | 1. 设置输入框如何显示文本的内容 |
| 2. 如果该值非空，则输入框会显示指定字符串代替真正的内容 |
| 3. 将该选项设置为 "\*"，则是密码输入框 |
| **state** | 1. Entry 组件可以设置的状态："normal"，"disabled" 或 "readonly"（注意，它跟 "disabled" 相似，但它支持选中和拷贝，只是不能修改，而 "disabled" 是完全禁止） |
| 2. 默认值是 "normal" |
| 3. 注意，如果此选项设置为 "disabled" 或 "readonly"，那么调用 insert() 和 delete() 方法都会被忽略 |
| **takefocus** | 1. 指定使用 Tab 键可以将焦点移动到输入框中 |
| 2. 默认是开启的，可以将该选项设置为 False 避免焦点在此输入框中 |
| **textvariable** | 1. 指定一个与输入框的内容相关联的 Tkinter 变量（通常是 StringVar） |
| 2. 当输入框的内容发生改变时，该变量的值也会相应发生改变 |
| **validate** | 1. 该选项设置是否启用内容验证 |
| 2. 详见本内容最下方小甲鱼关于验证详解 |
| **validatecommand** | 1. 该选项指定一个验证函数，用于验证输入框内容是否合法 |
| 2. 验证函数需要返回 True 或 False 表示验证结果 |
| 3. 注意，该选项只有当 validate 的值非 "none" 时才有效 |
| 3. 详见本内容最下方小甲鱼关于验证详解 |
| **vcmd** | 跟 validatecommand 一样 |
| **width** | 1. 设置输入框的宽度，以字符为单位 |
| 2. 默认值是 20 |
| 3. 对于变宽字体来说，组件的实际宽度等于字体的平均宽度乘以 width 选项的值 |
| **xscrollcommand** | 1. 与 scrollbar（滚动条）组件相关联 |
| 2. 如果你觉得用户输入的内容会超过该组件的输入框宽度，那么可以考虑设置该选项 |
| 3. 使用方法可以参考：Scrollbar 组件 |

**04** [**Button 控件**](https://www.cnblogs.com/zyg123/p/10351073.html)**\_第三周\_第四周**事件处理方法

**目录**

**前言**

*Button小部件是一个标准的Tkinter的控件，用于实现各种按钮。按钮可以包含文本或图像，您可以调用Python函数或方法用于每个按钮。Tkinter的按钮被按下时，会自动调用该函数或方法*

**(一)基本用法和可选属性**

1.基本用法

*基本用法：Button(根对象, [属性列表])*

*根对象：在那个窗体显示，例如主窗体。*

*属性列表：是可选的属性=属性值组成。*

2.可选属性

| **属性** | **说明** |
| --- | --- |
| text | 标签显示的文本 |
| font | 设置文本的字体和大小 |
| fg(foreground) | 字体的颜色， |
| bg (background) | 标签的背景色 |
| width | 标签的宽度(一个中文的字体宽为单位) |
| height | 标签的高度(一个中文的字体高为单位) |
| cursor | 鼠标的样式 |
| command | 绑定事件 |
| padx | 文字到边框的距离，水平方向 |
| pady | 文字到边框的距离，垂直方向 |
| bd(borderwidth) | 边框的宽度 |
| relief | 边框的样式 |
| justify | 文本对齐方式 |
| image | 图片 |
| compound | 图片与文字的混搭 |
| anchor | 方位 |

**(二)属性的具体实现和案例**

1.常用属性

(1)font

*font:设置字体与字体的大小*

*用法：font=("字体名"，大小) 例如：font=(“黑体”， 20)*

(2)fg 与 bg

*fg 前景色，也就是字体的颜色，bg 背景颜色*

*用法：fg="red", fg="#121234"*

(3)width 与 height

*width height 标签的宽度与高度，都是以系统默认的中文的一个字体宽高为单位*

*用法：width = 5, height=2*

案例一

(1)源代码

import tkinter as tk

win = tk.Tk()

# 普通的按钮

button1 = tk.Button(win, text="Button1")

button1.pack()

# 背景色与前景色

button2 = tk.Button(win, text="Button2", bg="green", fg="blue")

button2.pack()

# 宽度与高度

button3 = tk.Button(win, text="Button3", width=10, height=2)

button3.pack()

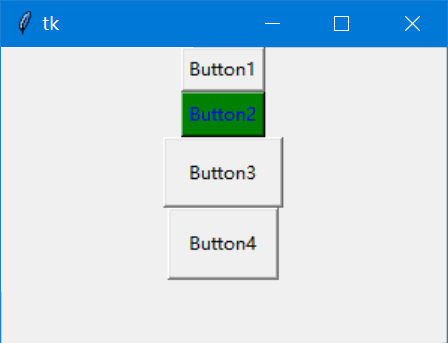
# 边距

button4 = tk.Button(win, text="Button4", padx=10, pady=10)

button4.pack()

win.mainloop()

(2)输出效果



2.按钮里的图片

(1)只放图片，没有文字

*需要先导入图片的路径：img1 = tk.PhotoImage(file="image/01.png")*

*再使用：image=img1*

*注：目前支持 .png 与 .gif 格式， 还不支持 .jpg格式，Button的大小是根据图片的大小来确定的。*

案例二

(1)源代码：

import tkinter as tk

win = tk.Tk()

img1 = tk.PhotoImage(file="image/01.png")

img2 = tk.PhotoImage(file="image/03.png")

img3 = tk.PhotoImage(file="image/04.png")

# 300像素大小的图片

button1 = tk.Button(win, text="Button1", image=img1)

button1.pack()

# 150像素大小的图片

button2 = tk.Button(win, image=img2)

button2.pack()

# 50像素大小的图片

button3 = tk.Button(win, image=img3)

button3.pack()

win.mainloop()

(2)输出效果：



(3)图片与文字混搭

*需要使用：compound="对齐方式"，*

*对齐方式有：'left', "right", "center"*

案例三

(1)源代码

import tkinter as tk

win = tk.Tk()

img1 = tk.PhotoImage(file="image/01.png")

img2 = tk.PhotoImage(file="image/03.png")

img3 = tk.PhotoImage(file="image/04.png")

button1 = tk.Button(win, text="Button1", image=img1, compound="left")

button1.pack()

button2 = tk.Button(win, text="Button2", image=img2, compound="center")

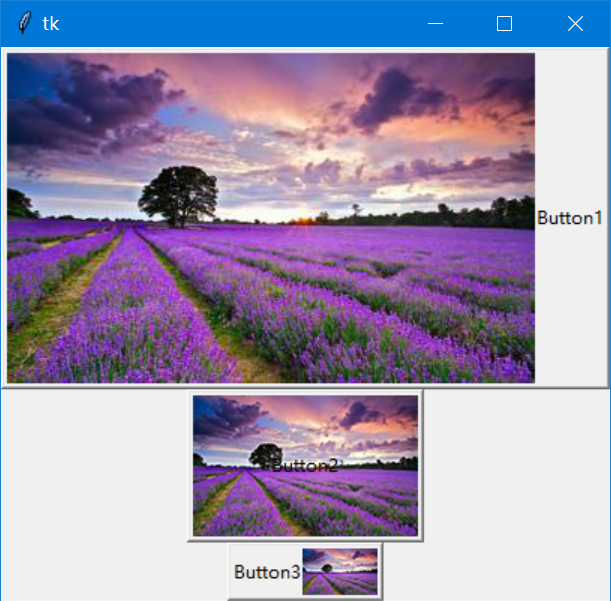
button2.pack()

button3 = tk.Button(win, text="Button3", image=img3, compound="right")

button3.pack()

win.mainloop()

(2)输出效果



3.鼠标的样式

*cursor="鼠标的属性值"*

*pencil：笔型*

*circle：圆形*

*hand1：手型1*

*hand2：手型2*

案例四

(1)源代码

import tkinter as tk

win = tk.Tk()

# 笔型

button1 = tk.Button(win, text="Button1", cursor="pencil")

button1.pack()

# 圆形

button2 = tk.Button(win, text="Button2", cursor="circle")

button2.pack()

# 手型1

button3 = tk.Button(win, text="Button3", cursor="hand1")

button3.pack()

# 手型2

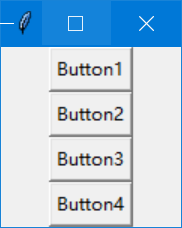
button4 = tk.Button(win, text="Button4", cursor="hand2")

button4.pack()

win.mainloop()

(2)输出效果

*当我们把鼠标放在按钮上时，鼠标的形状会显示不同的样式。*



4.边框样式

*relief= "边框样式值"*

*flat 无边框*

*groove 中间凹*

*ridge 中间凸*

*raised 往中间凸*

*solid 往中间凹*

*sunken 不可以*

案例五

(1)源代码

import tkinter as tk

win = tk.Tk()

# flat 无边框

button1 = tk.Button(win, text="flat", relief="flat", bd=10)

button1.pack()

# groove 中间凹

button2 = tk.Button(win, text="groove", relief="groove", bd=10)

button2.pack()

# ridge 中间凸

button3 = tk.Button(win, text="raised", relief="ridge", bd=10)

button3.pack()

# raised 往中间凸

button4 = tk.Button(win, text="ridge", relief="raised", bd=10)

button4.pack()

# solid 往中间凹

button5 = tk.Button(win, text="solid", relief="solid", bd=10)

button5.pack()

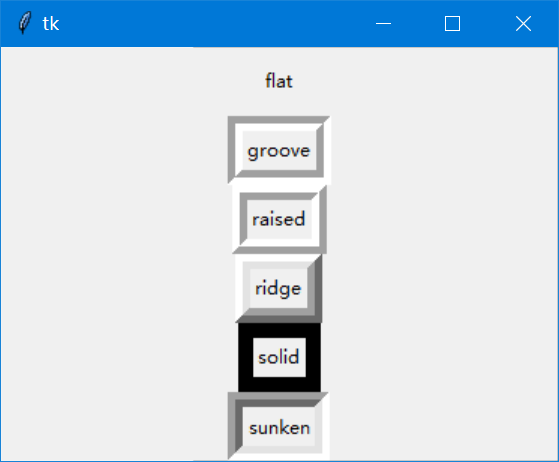
# sunken 不可以

button6 = tk.Button(win, text="sunken", relief="sunken", bd=10)

button6.pack()

win.mainloop()

(2)输出效果



**(三)按钮的事件绑定**

1.普通的Button绑定事件

(1)说明：

*Button 使用 command=功能函数 来绑定*

*Button(win, text="确定", command=功能函数)*

案例六

(1)源代码：

*我们创建一个简单的窗体，只有一个按钮控件，*

*我们绑定的事件是，当我们点击"确定"按钮时，会输出“你点击了按钮”*

import tkinter as tk

win = tk.Tk()

# 定义功能函数, event是必须添加的参数，不知道来自哪里

def button\_command():

print("你点击了按钮")

# 绑定事件

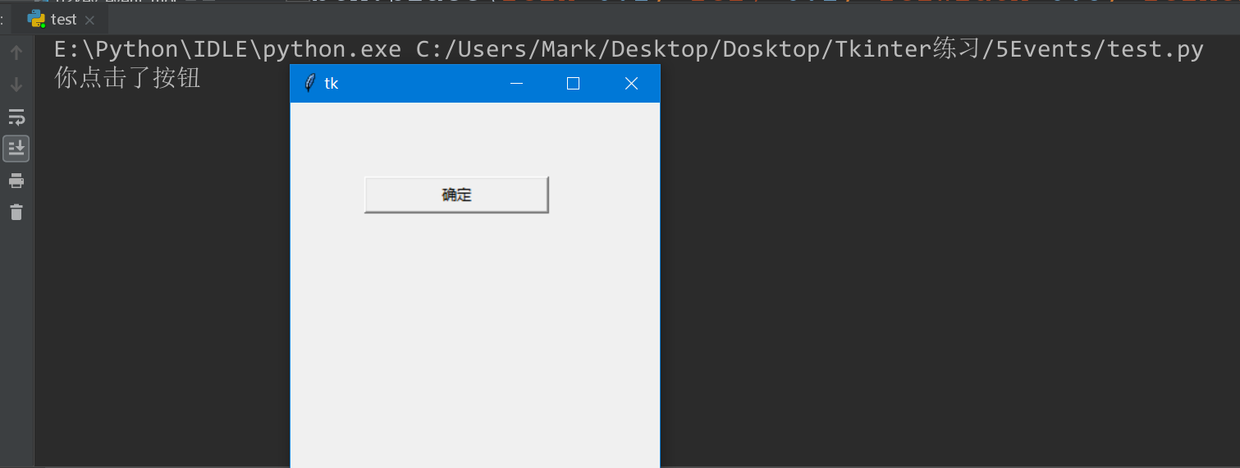
btn = tk.Button(win, text="确定", command=button\_command)

btn.place(relx=0.2, rely=0.2, relwidth=0.5, relheight=0.1)

win.geometry("300x300+200+200")

win.mainloop()

(2)输出效果：



2.传参数Button绑定事件

(1)说明：

*我们使用Button传递数值时，需要用：*

*lambda: 功能函数(var1, var2, ……)*

案例七

(1)源代码：

*我们同样创建一个简单的窗体，只有一个控件按钮*

*我们绑定的事件是，当我们点击按钮时，会传入两个参数，并在功能函数进行计算。*

import tkinter as tk

"""

Button command 传值事件

command= lambda: function(var1, var2, ...)

"""

def sum\_fun(a, b):

result = a + b

print("%d + %d = %d" % (a, b, result))

win = tk.Tk()

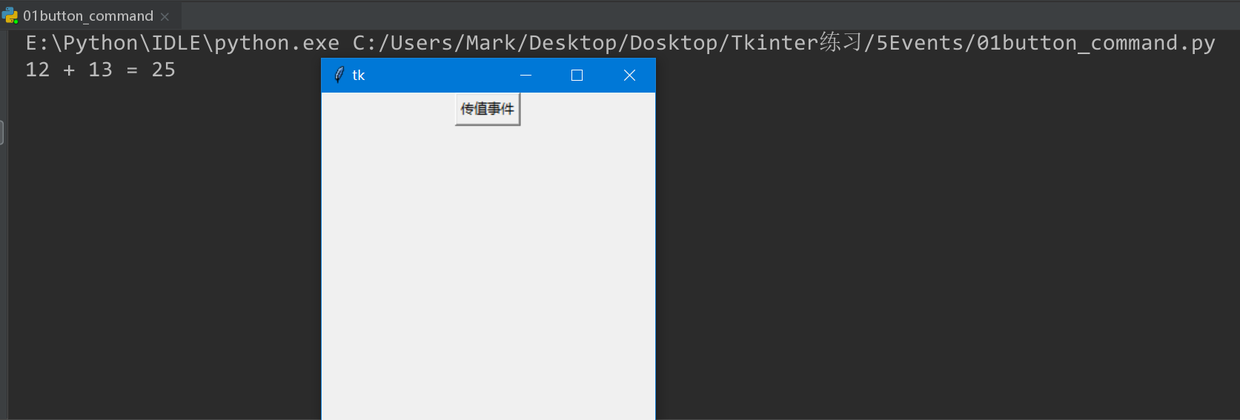
button = tk.Button(win, text="传值事件", command=lambda: sum\_fun(12, 13))

button.pack()

win.geometry("300x300+200+200")

win.mainloop()

(2)输出效果：



## 两种事件处理方法

### command 绑定事件处理方法

简单的事件处理可通过 command 选项来绑定，该选项绑定为一个函数或方法，当用户单击指定按钮时，通过该 command 选项绑定的函数或方法就会被触发。

下面程序示范了为按钮的 command 绑定事件处理方法：

上面程序中代码为 Button 的 command 选项指定为 start，这意味着当该按钮被单击时，将会触发当前对象的 start() 方法。该 start() 方法会改变界面上 Label 的文本和背景色。

运行该程序，单击界面上的“单击我”按钮，将看到如图 1 所示的界面。

from tkinter import \*

root = Tk()

root.geometry('300x200+150+150')

lab\_text = Label(root, text = '点击次数= 0')

lab\_text.pack()    #独立.pack出来！<=====================

count = 0

def start():

    #要改的label、替换的图片，缺一不可都要global引用！<======================

    global lab\_text,count

    count += 1

    lab\_text.configure(text='点击次数= %d'%count)

but\_OK = Button(root,text = '开始', command=start)

but\_OK.pack(side= 'right')

but\_Cancel = Button(root,text = '取消', command=start)

but\_Cancel.pack(side = 'left')

root.mainloop()



图 1 使用 command 绑定事件处理

### bind绑定事件处理方法

上面这种**简单的事件绑定方式**虽然简单，但它存在较大的局限性：

* 程序无法为具体事件（比如鼠标移动、按键事件）绑定事件处理方法。
* 程序无法获取事件相关信息。

**以上面例子无法获取是哪个按钮点击**

为了弥补这种不足，Python 提供了更灵活的事件绑定方式，所有 Widget 组件都提供了一个**bind()**方法，该方法可以为“任意” 事件绑定事件处理方法

我们先看一下 **单击鼠标左键为 <Button-1> ，以上面例子**

from tkinter import \*

root = Tk()

root.geometry('300x200+150+150')

lab\_text = Label(root, text = '点击次数= 0')

lab\_text.pack()    #独立.pack出来！<=====================

count = 0

def start(event):

    #要改的label、替换的图片，缺一不可都要global引用！<======================

    global lab\_text,count

    count += 1

    #获取事件源的方式

    ss = '事件源:%s \n 事件源的文本%s:' %(event.widget,event.widget['text'])

    lab\_text.configure(text= ss +'点击次数= %d'%count)

but\_OK = Button(root,text = '开始')

but\_OK.bind('<Button-1>',start)

but\_OK.pack(side= 'right')

but\_Cancel = Button(root,text = '取消')

but\_Cancel.bind('<Button-1>',start)

but\_Cancel.pack(side = 'left')

root.mainloop()





Tkinter 直接使用字符串来代表事件类型，比如使用 <Button-1> 代表鼠标左键单击事件，使用 <Double-1> 代表鼠标左键双击事件。那问题来了，其他事件应该怎么写呢？

代表 Tkinter 事件的字符串大致遵循如下格式：

<modifier-type-detail>

其中 type 是事件字符串的关键部分，用于描述事件的种类，比如鼠标事件、键盘事件等；modifer 则代表事件的修饰部分，比如单击、双击等；detail 用于指定事件的详情，比如指定鼠标左键、右键、滚轮等。

Tkinter 支持的各种鼠标、键盘事件如表 3 所示。

|  |  |
| --- | --- |
| **事件** | **简介** |
| <Button-detail> | 鼠标按键的单击事件，detail 指定哪一个鼠标键被单击。比如单击鼠标左键为 <Button-1>，单击鼠标中键为 <Button-2>，单击鼠标右键为 <Button-3>，单击向上滚动的滚轮为 <Button-4>，单击向下滚动的滚轮为 <Button-5> |
| <modifier Motion> | 鼠标在组件上的移动事件，modifier 指定要求按住哪个鼠标键。比如按住鼠标左键移动为 <B1-Motion>，锁住鼠标中键移动为 <B2-Motion>，按住鼠标右键移动为 <B3-Motion> |
| <ButtonRelease-detail> | 鼠标按键的释放事件，detail 指定哪一个鼠标键被释放。比如鼠标左键被释放为 <ButtonRelease-1>，鼠标中键被释放为 <ButtonRelease-2>，鼠标右键被释放为 <ButtonRelease-3> |
| <Double-Button-detail>或<Double-detail> | 用户双击某个鼠标键的事件，detail 指定哪一个鼠标键被双击。比如双击鼠标左键为 <Double-1>，双击鼠标中键为 <Double-2>，双击鼠标右键为 <Double-3>，双击向上滚动的滚轮为 <Double-4>，双击向下滚动的滚轮为 <Double-5> |
| <Enter> | 鼠标进入组件的事件。注意，<Enter> 事件不是按下回车键事件，按下回车键的事件是 <Return> |
| <Leave> | 鼠标移出组件事件 |
| <Focusln> | 组件及其包含的子组件获得焦点 |
| <FocusOut> | 组件及其包含的子组件失去焦点 |
| <Return> | 按下回车键的事件。实际上可以为所有按键绑定事件处理方法。特殊键位名称包括 Cancel、BackSpace、Tab、Return（回车）、Shift\_L（左Shift，如果只写 Shift 则代表任意 Shift）、Control\_L（左 Ctrl，如果只写 Control 则代表任意 Ctrl）、Alt\_L（左 Alt，如果只写 Alt 则代表任意 Alt）、Pause、Caps\_Lock、Escape、Prior（Page Up）、Next（Page Down）、End、Home、Left、Up、Right、Down、Print、Insert、Delete、F1、F2、F3、F4、F5、F6、F7、F8、F9、F10、F11、F12、Num\_Lock 和 Scroll\_Lock |
| <Key> | 键盘上任意键的单击事件，程序可通过 event 获取用户单击了哪个键 |
| a | 键盘上指定键被单击的事件。比如‘a’代表 a 键被单击，‘b’代表 b 键被单击（不要尖括号）…… |
| <Shift-Up> | 在 Shift 键被按下时按 Up 键。类似的还有 <Shift-Left>、<Shift-Down>、<Alt-Up>、<Control-Up> 等 |
| <Configure> | 组件大小、位置改变的事件。组件改变之后的大小、位置可通过 event 的 width、height、x、y 获取 |

## 计算器实例\_之事件

from tkinter import \*

import  tkinter.font   as  tkFont  #导入Tkinter字体模块

reset = True

def buttonCallBack(event):

    global label,reset

    showText = "1234567890.+-\*/"

    num = event.widget['text']

    if label['text'] == '0' or reset == True:

        label['text'] = ""

        reset = False

    if num in showText:

        label['text'] = label['text'] + num

        return

    if num in "=":

        label['text'] = str(eval(label['text']))

        reset = True

        return

    if num == 'C':

        label['text'] = "0"

        return

# 主窗口

root = Tk()

root.wm\_title("计算器")

# 显示栏1

f = tkFont.Font(family="黑休", size=16, weight="bold",slant= "roman")

label = Label(root, text="0", background="darkgreen", fg = 'white',anchor="e",font =f )

label['width'] = 35

label['height'] = 2

label.grid(row=1, columnspan=4, sticky=W,padx = 20,pady = 5)

# 按钮

showText = "789/456\*123-0.C+"

for i in range(4):

    for j in range(4):

        b = Button(root, text=showText[i\*4+j], width=5,font =f )

        b.grid(row=i+2, column=j,padx = 0,pady = 5)

        b.bind("<Button-1>", buttonCallBack)

showText = "()"

for i in range(2):

    b = Button(root, text=showText[i], width=5,font =f )

    b.grid(row=6, column=2+i,padx = 0,pady = 5)

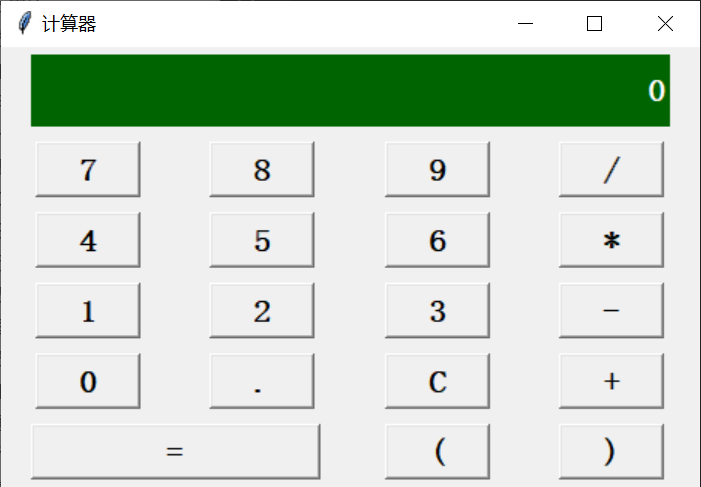
    #b.bind("<button-1>", buttonCallBack)

b = Button(root, text="=",font =f )

b.grid(row=6, columnspan=2, sticky="we",padx = 20,pady = 5)

b.bind("<Button-1>", buttonCallBack)

root.mainloop()



[**07 Frame与Labelframe 控件**](https://www.cnblogs.com/zyg123/p/10382609.html)**\_第四周**

**前言**

*Frame与Labelframe都是容器，用来存放其他控件，也是用来更好的管理布局。*

*我一般是用来存放一组相关的控件，让Frame与Labelframe成为其他控件的父控件。*

**(一)Frame**

(1)说明：

*tk.Frame(master, option\*)*

*一般用颜色来区分Frame的位置*

(2)源代码：

import tkinter as tk

win = tk.Tk()

# 定义第一个容器

frame\_left = tk.Frame(win, bg="green")

frame\_left.place(relx=0.2, rely=0.2, relwidth=0.3, relheight=0.6)

label\_1 = tk.Label(frame\_left, text="富强")

label\_1.place(relx=0.2, rely=0.2)

label\_2 = tk.Label(frame\_left, text="民主")

label\_2.place(relx=0.6, rely=0.2)

label\_3 = tk.Label(frame\_left, text="文明")

label\_3.place(relx=0.2, rely=0.6)

label\_4 = tk.Label(frame\_left, text="和谐")

label\_4.place(relx=0.6, rely=0.6)

# 定义第二个容器

frame\_right = tk.Frame(win, bg="yellow")

frame\_right.place(relx=0.5, rely=0.2, relwidth=0.3, relheight=0.6)

label\_1 = tk.Label(frame\_right, text="自由")

label\_1.place(relx=0.2, rely=0.2)

label\_2 = tk.Label(frame\_right, text="平等")

label\_2.place(relx=0.6, rely=0.2)

label\_3 = tk.Label(frame\_right, text="公正")

label\_3.place(relx=0.2, rely=0.6)

label\_4 = tk.Label(frame\_right, text="法治")

label\_4.place(relx=0.6, rely=0.6)

win.mainloop()

(3)输出效果：



**(二)Labelframe**

(1)说明：

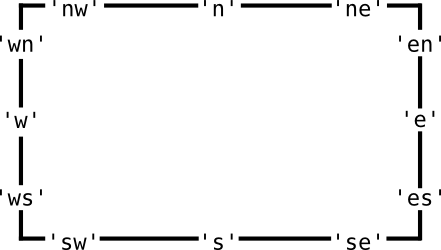
*大体上，Labelframe与Frame属性选项一样。*

*Labelframe有两点不同于Frame:*

1. *Labelframe默认的边框使tk.GROOVE，relief=tk.GROOVE*
2. *Labelframe还可以再边框写文字，即：text="文字"*

边框文字的位置：

*默认是：labelanchor="wn"*



(2)源代码：

import tkinter as tk

win = tk.Tk()

# 定义第一个容器

frame\_left = tk.LabelFrame(win, text="国家层面", labelanchor="n")

frame\_left.place(relx=0.2, rely=0.2, relwidth=0.3, relheight=0.6)

label\_1 = tk.Label(frame\_left, text="富强")

label\_1.place(relx=0.2, rely=0.2)

label\_2 = tk.Label(frame\_left, text="民主")

label\_2.place(relx=0.6, rely=0.2)

label\_3 = tk.Label(frame\_left, text="文明")

label\_3.place(relx=0.2, rely=0.6)

label\_4 = tk.Label(frame\_left, text="和谐")

label\_4.place(relx=0.6, rely=0.6)

# 定义第二个容器

frame\_right = tk.LabelFrame(win, text="社会层面", labelanchor="n")

frame\_right.place(relx=0.5, rely=0.2, relwidth=0.3, relheight=0.6)

label\_1 = tk.Label(frame\_right, text="自由")

label\_1.place(relx=0.2, rely=0.2)

label\_2 = tk.Label(frame\_right, text="平等")

label\_2.place(relx=0.6, rely=0.2)

label\_3 = tk.Label(frame\_right, text="公正")

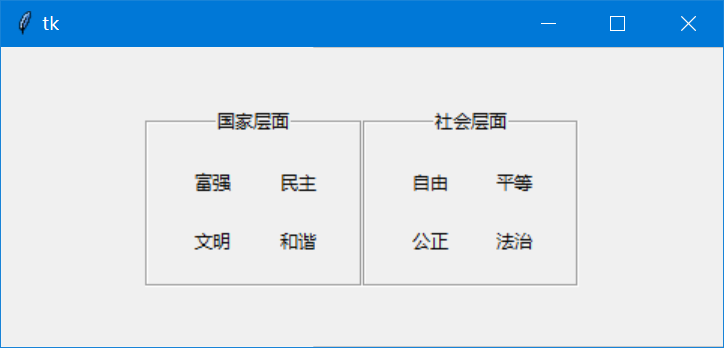
label\_3.place(relx=0.2, rely=0.6)

label\_4 = tk.Label(frame\_right, text="法治")

label\_4.place(relx=0.6, rely=0.6)

win.mainloop()

(3)输出效果：



作者：Mark

Python tkinter(三) 单选框(Radiobutton)组件的属性说明及示例

# 单选框(Radiobutton)组件的属性说明及示例

置顶 沉默的鹏先生 2019-10-30 11:08:07 4809 收藏 22

分类专栏： # tkinter控件

版权

Python tkinter 单选框组件用于tkinter GUI里添加单选按钮，单选框可以添加文本和图像。当单选框勾选时，可以执行指定的函数，或者获取勾选的值。

使用语法

widget = Radiobutton( master, parameter=value, ... )

master：按钮控件的父容器

parameter：按钮的参数

value：参数对应的值

各参数之间以逗号分隔。

参数说明

text

单选框文本显示

variable

关联单选框执行的函数

value

用于多个单选框值的区别

.set(value)

默认选中指定的单选框

relief

单选框的边框样式显示，可选项包括FLAT/SUNKEN/RAISED/GROOVE/RIDGE

height

单选框的高度，需要结合单选框的边框样式才能展示出效果

width

单选框的宽度，需要结合单选框的边框样式才能展示出效果

bd

单选框边框样式的宽度，需要结合单选框的边框样式才能展示出效果

activebackground

鼠标点击单选框时显示的前景色

activeforeground

鼠标点击单选框时显示的背景色

bg

单选框显示的前景色

fg

单选框显示的背景色

font

单选框的文字字体、字号、字形，字形可选项包括bold/italic/underline/overstrike

image

单选框显示图片，图片必须是gif格式，并且图片需要用PhotoImage赋值给变量，然后变量赋值给image

justify

单选框文字对齐方式，可选项包括LEFT, RIGHT, CENTER

wraplength

限制每行的文字，单选框文字达到限制的字符后，自动换行

underline

下划线。取值就是带下划线的字符串索引，为 0 时，第一个字符带下划线，为 1 时，第两个字符带下划线，以此类推

.config(state=)

单选框的状态，状态可选项有DISABLED/NORMAL/ACTIVE

源码示例

# -\*- coding:utf-8 -\*-

from tkinter import \*

class radiobutton:

def \_\_init\_\_(self):

root = Tk()

root.title("单选框") # 设置窗口标题

root.geometry("650x600") # 设置窗口大小 注意：是x 不是\*

'''单选框样式'''

# 指定Radiobutton的事件处理函数

iv\_command = IntVar()

self.rb\_function\_Label = Label(root, text='关联函数：')

self.rb\_function = Radiobutton(root, text='执行函数', command=self.function1, variable=iv\_command)

# 单选框默认选择

iv\_default = IntVar()

self.rb\_default\_Label = Label(root, text='默认选择：')

self.rb\_default1 = Radiobutton(root, text='单选框1', value=1, variable=iv\_default)

self.rb\_default2 = Radiobutton(root, text='单选框2', value=2, variable=iv\_default)

iv\_default.set(1)

# 单选框样式

iv\_style = IntVar()

self.rb\_style\_Label = Label(root, text='显示边框样式：')

self.rb\_style1 = Radiobutton(root, text='边框平坦', value=1, variable=iv\_style, relief=FLAT)

self.rb\_style2 = Radiobutton(root, text='边框凹陷', value=2, variable=iv\_style, relief=SUNKEN)

self.rb\_style3 = Radiobutton(root, text='边框凸起', value=3, variable=iv\_style, relief=RAISED)

self.rb\_style4 = Radiobutton(root, text='边框压线', value=4, variable=iv\_style, relief=GROOVE)

self.rb\_style5 = Radiobutton(root, text='边框脊线', value=5, variable=iv\_style, relief=RIDGE)

# 单选框高度，height='高度'，wideth='宽度'，单选框的高度和宽度必须和边框样式一起使用才有效果

iv\_xy = IntVar()

self.rb\_xy\_Label = Label(root, text='单选框边框xy：')

self.rb\_height = Radiobutton(root, text='边框高度', value=1, variable=iv\_xy, relief=RAISED, height=2)

self.rb\_wideth = Radiobutton(root, text='边框高度', value=2, variable=iv\_xy, relief=RAISED, width=16)

# 单选框边框大小，bd='边框大小'，单选框的边框效果必须和边框样式一起使用才有效果

iv\_border = IntVar()

self.rb\_border\_Label = Label(root, text='单选框边框大小：')

self.rb\_border = Radiobutton(root, text='边框大小', variable=iv\_border, relief=RAISED, bd=5)

# 鼠标点击到单选框后改变颜色，activebackground='背景色'，activeforeground='前景色'

iv\_click\_colour = IntVar()

self.rb\_click\_colour\_Label = Label(root, text='鼠标点击颜色：')

self.rb\_click\_colour1 = Radiobutton(root, text='前景色', value=1, variable=iv\_click\_colour,

activebackground='blue')

self.rb\_click\_colour2 = Radiobutton(root, text='背景色', value=2, variable=iv\_click\_colour,

activeforeground='blue')

# 单选框颜色，bg='背景色', fg='前景色'

iv\_colour = IntVar()

self.rb\_colour\_Label = Label(root, text='单选框颜色：')

self.rb\_colour1 = Radiobutton(root, text='前景色', value=1, variable=iv\_colour, bg='blue')

self.rb\_colour2 = Radiobutton(root, text='背景色', value=2, variable=iv\_colour, fg='blue')

# 单选框文字字体格式， font=('字体', 字号, 'bold/italic/underline/overstrike')

iv\_font = IntVar()

self.rb\_font\_Label = Label(root, text='显示边框样式：')

self.rb\_font1 = Radiobutton(root, text='软体雅黑/12/重打印', value=1, variable=iv\_font,

font=('软体雅黑', 10, 'overstrike'))

self.rb\_font2 = Radiobutton(root, text='宋体/12/斜体', value=2, variable=iv\_font, font=('宋体', 10, 'italic'))

self.rb\_font3 = Radiobutton(root, text='黑体/12/加粗', value=3, variable=iv\_font, font=('黑体', 10, 'bold'))

self.rb\_font4 = Radiobutton(root, text='楷体/12/下划线', value=4, variable=iv\_font,

font=('楷体', 10, 'underline'))

# 单选框图片设置，image=图片变量。图片必须以变量的形式赋值给image，图片必须是gif格式。

iv\_image = IntVar()

self.rb\_image\_Label = Label(root, text='单选框图片：')

gif = PhotoImage(file="1.gif")

self.rb\_image = Radiobutton(root, variable=iv\_image, image=gif)

# 单选框文字对齐方式，可选项包括LEFT, RIGHT, CENTER

iv\_aligning = IntVar()

self.rb\_aligning\_Label = Label(root, text='文字对齐方式：')

self.rb\_aligning1 = Radiobutton(root, text='左对齐\n文字左侧对齐', value=1, variable=iv\_aligning, justify=LEFT)

self.rb\_aligning2 = Radiobutton(root, text='居中对齐\n文字居中对齐', value=2, variable=iv\_aligning, justify=CENTER)

self.rb\_aligning3 = Radiobutton(root, text='右对齐\n文字右侧对齐', value=3, variable=iv\_aligning, justify=RIGHT)

# 单选框达到限制字符后换行显示

iv\_linefeed = IntVar()

self.rb\_linefeed\_Label = Label(root, text='文字换行显示：')

self.rb\_linefeed = Radiobutton(root, text='1234567890', variable=iv\_linefeed, wraplength=30)

# 下划线。取值就是带下划线的字符串索引，为 0 时，第一个字符带下划线，为 1 时，第两个字符带下划线，以此类推

iv\_underline = IntVar()

self.rb\_underline\_Label = Label(root, text='文字标下划线：')

self.rb\_underline = Radiobutton(root, text='12345', variable=iv\_underline, underline=2)

# 单选框状态

iv\_status = IntVar()

self.rb\_status\_Label = Label(root, text='按钮状态：')

self.rb\_disabled = Radiobutton(root, text='禁用状态', value=1, variable=iv\_status)

self.rb\_disabled.config(state=DISABLED)

self.rb\_usual = Radiobutton(root, text='普通状态', value=2, variable=iv\_status)

self.rb\_usual.config(state=NORMAL)

self.rb\_active = Radiobutton(root, text='活跃状态', value=3, variable=iv\_status)

self.rb\_active.config(state=ACTIVE)

'''grid布局'''

self.rb\_function\_Label.grid(row=1, column=0, sticky='E')

self.rb\_function.grid(row=1, column=1, sticky='W')

self.rb\_default\_Label.grid(row=2, column=0, sticky='E')

self.rb\_default1.grid(row=2, column=1, sticky='W')

self.rb\_default2.grid(row=2, column=2, sticky='W')

self.rb\_style1.grid(row=2, column=3, sticky='W')

self.rb\_style\_Label.grid(row=3, column=0, sticky='E')

self.rb\_style1.grid(row=3, column=1, sticky='W')

self.rb\_style2.grid(row=3, column=2, sticky='W')

self.rb\_style3.grid(row=3, column=3, sticky='W')

self.rb\_style4.grid(row=3, column=4, sticky='W')

self.rb\_style5.grid(row=3, column=5, sticky='W')

self.rb\_xy\_Label.grid(row=4, column=0, sticky='E')

self.rb\_height.grid(row=4, column=1, sticky='W')

self.rb\_wideth.grid(row=4, column=2, columnspan=2, sticky='W')

self.rb\_border\_Label.grid(row=5, column=0, sticky='E')

self.rb\_border.grid(row=5, column=1, sticky='W')

self.rb\_click\_colour\_Label.grid(row=6, column=0, sticky='E')

self.rb\_click\_colour1.grid(row=6, column=1, sticky='W')

self.rb\_click\_colour2.grid(row=6, column=2, sticky='W')

self.rb\_colour\_Label.grid(row=7, column=0, sticky='E')

self.rb\_colour1.grid(row=7, column=1, sticky='W')

self.rb\_colour2.grid(row=7, column=2, sticky='W')

self.rb\_font\_Label.grid(row=8, column=0, rowspan=2, sticky='E')

self.rb\_font1.grid(row=8, column=1, columnspan=2, sticky='W')

self.rb\_font2.grid(row=8, column=3, columnspan=2, sticky='W')

self.rb\_font3.grid(row=9, column=1, columnspan=2, sticky='W')

self.rb\_font4.grid(row=9, column=3, columnspan=2, sticky='W')

self.rb\_image\_Label.grid(row=10, column=0, sticky='E')

self.rb\_image.grid(row=10, column=1, columnspan=3, sticky='W')

self.rb\_aligning\_Label.grid(row=11, column=0, sticky='E')

self.rb\_aligning1.grid(row=11, column=1, sticky='W')

self.rb\_aligning2.grid(row=11, column=2, sticky='W')

self.rb\_aligning3.grid(row=11, column=3, sticky='W')

self.rb\_linefeed\_Label.grid(row=12, column=0, sticky='E')

self.rb\_linefeed.grid(row=12, column=1, sticky='W')

self.rb\_underline\_Label.grid(row=13, column=0, sticky='E')

self.rb\_underline.grid(row=13, column=1, sticky='W')

self.rb\_status\_Label.grid(row=14, column=0, sticky='E')

self.rb\_disabled.grid(row=14, column=1, sticky='W')

self.rb\_usual.grid(row=14, column=2, sticky='W')

self.rb\_active.grid(row=14, column=3, sticky='W')

root.mainloop()

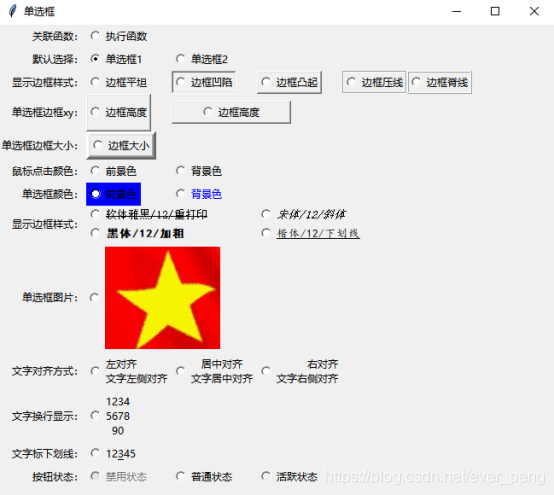
def function1(self):

print('选中function1')

if \_\_name\_\_ == '\_\_main\_\_':

radiobutton()

效果展示



[**08 Canvas控件**](https://www.cnblogs.com/zyg123/p/10382630.html)**\_第五、六周**

**前言**

*carves就是画布的意思，用于画图。*

**(一)carves创建方法**

**(1)说明：**

*与其他的控件一样，carves的创建方也是在tk里。*

**(2)源代码：**

import tkinter as tk

win = tk.Tk()

win.geometry("600x400+200+100")

# 创建画布

canvas = tk.Canvas(win, bg="orange")

canvas.place(relx=0.05, rely=0.05, relwidth=0.9, relheight=0.9)

win.mainloop()

**(3)输出效果：**



**(二)carves的画图方法**

**(1)说明：**

| **方法** | **说明** | **参数** |
| --- | --- | --- |
| .create\_line() | 画线 | (起始坐标)，(终点坐标)，width=线宽，fill=颜色 |
| .create\_arc() | 画圆弧 | (起始坐标)，(终点坐标)，width=线宽，fill=颜色 |
| .create\_rectangle() | 画矩形 | (起始坐标)，(终点坐标)，fill='填充的颜色, outline=边框的颜色 |
| .create\_oval() | 画椭圆 | (外接矩阵的四个坐标)，fill='填充的颜色, outline=边框的颜色 |
| .create\_polygon() | 画多边形 | d多个点的坐标，fill='填充的颜色, outline=边框的颜色 |
| .create\_text() | 显示文字 | text="文字" |
| .delete() | 删除对象 | 创建线返回的对象 |

**(2)源代码：**

import tkinter as tk

win = tk.Tk()

win.geometry("600x400+200+100")

canvas = tk.Canvas(win, bg="orange")

canvas.place(relx=0.05, rely=0.05, relwidth=0.9, relheight=0.9)

# 画一条实线， fill:填充的颜色

line1 = canvas.create\_line((0, 0), (100, 100), width=5, fill="red")

# 画一条虚线 dash=(1, 1)

canvas.create\_line((200, 200), (200, 300), width=5, fill="green", dash=(1, 1))

# 画一个圆弧

canvas.create\_arc((100, 100), (200, 200), width=5)

# 显示文字

canvas.create\_text((300, 100), text="文字", font=("微软雅黑", 18))

# 绘制矩形, outline：线条颜色

canvas.create\_rectangle(50, 25, 150, 75, fill='blue', outline='green', width=5)

# 绘制椭圆

canvas.create\_oval(200, 25, 350, 75, fill='pink', outline='green', width=5)

# 绘制多边形

point = [(100, 100), (100, 200), (200, 300), (300, 400), (400, 500)]

canvas.create\_polygon(point, outline='green', fill='yellow')

def dele\_line():

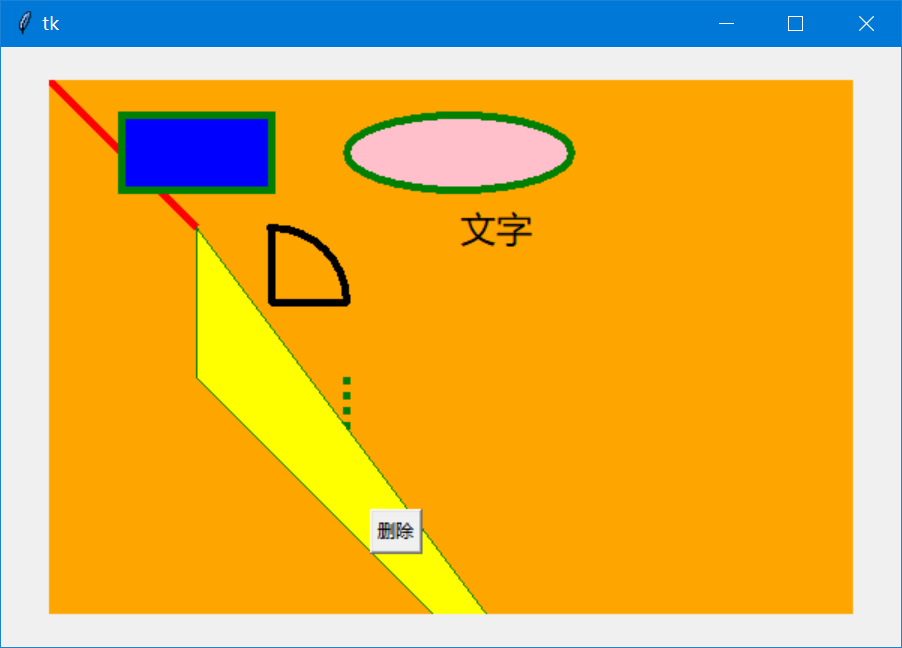
canvas.delete(line1)

btn = tk.Button(canvas, text="删除", command=dele\_line)

btn.place(relx=0.4, rely=0.8)

win.mainloop()

**(3)输出效果：**

****

[**05 Checkbutton控件**](https://www.cnblogs.com/zyg123/p/10372238.html)

**前言**

*复选框：可以同时多选的一组框，其只有两种状态，选中与未选中。*

**(一)基本属性**

(1)说明：

*tkinter里的变量：tk.Intvar()*

*用途：记录数值*

*其有两个方法：.get()获取值， .set()设置值*

1)特有的属性：

| **属性** | **说明** |
| --- | --- |
| **variable** | **控制变量** |
| **state** | **组件的状态，正常：normal 禁用：disabled** |
| **selectcolor** | **选中框内的颜色** |

2)专有方法：

| **.select()** | **设置为选中** |
| --- | --- |
| .delect() | 设置为未选中 |
| .toggle() | 切换选中状态 |

**(二)案例**

1.简单的复选框

(1)源代码：

import tkinter as tk

win = tk.Tk()

# 定义变量

v = tk.IntVar()

# 添加复选框

c = tk.Checkbutton(win, text="复选框", variable=v)

c.place(relx=0.3, rely=0.3)

# 获取状态

v.get()

win.mainloop()

(2)输出效果：



2.组合复选框

(1)源代码：

import tkinter as tk

win = tk.Tk()

# 定义变量

v1 = tk.IntVar()

v2 = tk.IntVar()

v3 = tk.IntVar()

frame = tk.Frame(win, bg="blue")

frame.place(relx=0.3, rely=0.3, relwidth=0.4, relheight=0.4)

# 添加复选框

c1 = tk.Checkbutton(frame, text="北京", variable=v1, bg="yellow")

c1.grid(row=0, column=0)

c2 = tk.Checkbutton(frame, text="上海", variable=v2, bg="yellow")

c2.grid(row=0, column=1)

c3 = tk.Checkbutton(frame, text="深圳", variable=v3, bg="yellow")

c3.grid(row=0, column=2)

# 获取状态

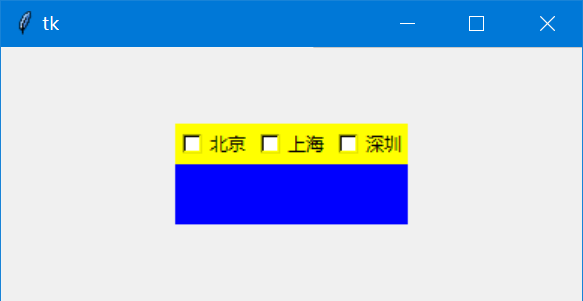
v1.get()

v2.get()

v3.get()

win.mainloop()

(2)输出效果：



作者：Mark

[**06 Radiobutton控件**](https://www.cnblogs.com/zyg123/p/10372243.html)

**目录**

目录

* [目录](https://www.cnblogs.com/zyg123/p/10372243.html#目录)
* [前言](https://www.cnblogs.com/zyg123/p/10372243.html#前言)
* [(一)基本属性](https://www.cnblogs.com/zyg123/p/10372243.html#（一）基本属性)
  + [(1)特有属性：](https://www.cnblogs.com/zyg123/p/10372243.html#（1）特有属性：)
  + [(2)使用思路：](https://www.cnblogs.com/zyg123/p/10372243.html#（2）使用思路：)
  + [(3)源代码：](https://www.cnblogs.com/zyg123/p/10372243.html#（3）源代码：)
  + [(4)输出效果：](https://www.cnblogs.com/zyg123/p/10372243.html#（4）输出效果：)
* [(二)在Frame里布局：](https://www.cnblogs.com/zyg123/p/10372243.html#（二）在frame里布局：)
  + [(1)源代码：](https://www.cnblogs.com/zyg123/p/10372243.html#（1）源代码：)
  + [(2)输出效果：](https://www.cnblogs.com/zyg123/p/10372243.html#（2）输出效果：)
  + [作者：Mark](https://www.cnblogs.com/zyg123/p/10372243.html#作者：mark)
  + [日期：2019/02/13 周三](https://www.cnblogs.com/zyg123/p/10372243.html#日期：20190213-周三)

**前言**

*Radiobutton单选框，在一组选框中，只能选中一个。*

**(一)基本属性**

(1)特有属性：

| **value** | **按钮的值** |
| --- | --- |
| variable | 控制变量 |
| selectcolor | 选中区域的颜色 |

(2)使用思路：

*需要先定义tkinter里的变量：v = Intvar()*

*v.set(),设置默认值*

*v.get()，获取选中的值*

(3)源代码：

import tkinter as tk

win = tk.Tk()

# 定义变量

v = tk.IntVar()

# 设置第二个未默认

v.set(2)

# 单选框

r1 = tk.Radiobutton(win, text="one", value=1, variable=v)

r1.pack()

r2 = tk.Radiobutton(win, text="two", value=2, variable=v)

r2.pack()

r3 = tk.Radiobutton(win, text="three", value=3, variable=v)

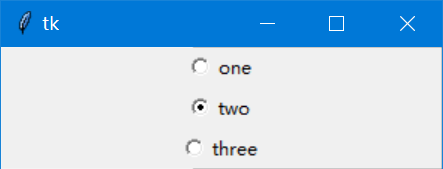
r3.pack()

# 获取状态

print(v.get())

win.mainloop()

(4)输出效果：



**(二)在Frame里布局：**

(1)源代码：

import tkinter as tk

win = tk.Tk()

# 定义变量

v = tk.IntVar()

frame = tk.Frame(win, bg="blue")

frame.place(relx=0.3, rely=0.3, relwidth=0.4, relheight=0.4)

# 单选框

r1 = tk.Radiobutton(frame, text="one", value=1, variable=v, bg="yellow")

r1.grid(row=0, column=0)

r2 = tk.Radiobutton(frame, text="two", value=2, variable=v, bg="yellow")

r2.grid(row=0, column=1)

r3 = tk.Radiobutton(frame, text="three", value=3, variable=v, bg="yellow")

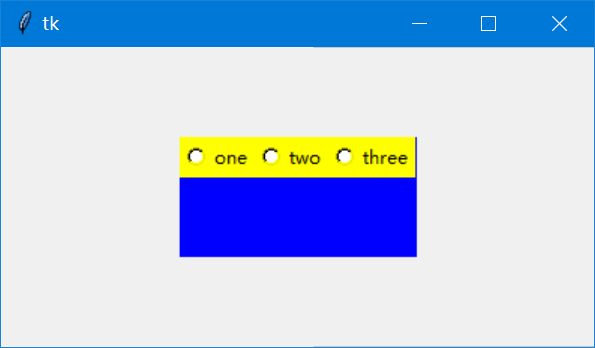
r3.grid(row=0, column=2)

# 获取状态

print(v.get())

win.mainloop()

(2)输出效果：

****

作者：Mark

Tkinter组件详解（五）：Listbox和Scrollbar

!一直往南方开. 2019-12-27 13:33:36 879 收藏 6

分类专栏： Tkinter 文章标签： Scrollbar

版权

# Tkinter组件详解之Listbox

Listbox（列表框）组件用于显示一个选择列表。Listbox 只能包含文本项目，并且所有的项目都需要使用相同的字体和颜色。根据组件的配置，用户可以从列表中选择一个或多个选项。

Listbox 组件通常被用于显示一组文本选项，Listbox 组件跟 Checkbutton 和 Radiobutton 组件类似，不过 Listbox 是以列表的形式来提供选项的（后两个是通过按钮的形式）。

## 用法

当你创建一个 Listbox 组件的时候，它是空的，所以第一件要做的事就是添加一行或多行文本进去。我们使用 insert() 方法添加文本，该方法有两个参数：第一个参数是插入的索引号，第二个参数是插入的字符串。索引号通常是项目的序号（0 是列表中第一项的序号）。

不过你也可以使用一些特殊的索引号：比如 active表示选中的项目（如果 Listbox 允许多选，那么它表示最后一个被选中的项目）；又如 end表示 Listbox 的最后一行，所以当要插入一个项目到列表时可以使用 end：

import tkinter as tk

master = tk.Tk()

# 创建一个空列表

theLB = tk.Listbox(master)

theLB.pack()

# 往列表里添加数据

for item in ["你好", "哈哈", "嘻嘻", "嘿嘿"]:

theLB.insert("end", item)

master.mainloop()



使用一个独立按钮来删除active状态的项目:

import tkinter as tk

master = tk.Tk()

# 创建一个空列表

theLB = tk.Listbox(master)

theLB.pack()

# 往列表里添加数据

for item in ["你好", "哈哈", "嘻嘻", "嘿嘿"]:

theLB.insert("end", item)

theButton = tk.Button(master, text="删除", command=lambda x=theLB: x.delete("active"))

theButton.pack()

master.mainloop()



Listbox 组件根据 selectmode 选项提供了四种不同的选择模式：“single”（单选）、“browse”（也是单选，但拖动鼠标或通过方向键可以直接改变选项）、“multiple”（多选）和 “extended”（也是多选，但需要同时按住 Shift 键或 Ctrl 键或拖拽鼠标实现）。默认是 “browse”。

## 参数



## 方法

activate(index)

– 将给定索引号对应的选项激活（在其文本下方画一条下划线）

bbox(index)

– 返回给定索引号对应的选项的边框

– 返回值是一个以像素为单位的 4 元祖表示边框：(xoffset, yoffset, width, height)

– xoffset 和 yoffset 表示距离左上角的偏移位置

– 返回的 width 是文本的实际宽度（像素为单位）

– 如果指向的选项是不可见的，那么返回值是 None

curselection()

– 返回一个元组，包含被选中的选项的序号（从 0 开始）

– 如果没有选中任何选项，返回一个空元组

delete(first, last=None)

– 删除参数 first 到 last 范围内（包含 first 和 last）的所有选项

– 如果忽略 last 参数，表示删除 first 参数指定的选项

get(first, last=None)

– 返回一个元组，包含参数 first 到 last 范围内（包含 first 和 last）的所有选项的文本

– 如果忽略 last 参数，表示返回 first 参数指定的选项的文本

index(index)

– 返回与 index 参数相应的选项的序号（例如 lb.index(“end”)）

insert(index, \*elements)

– 添加一个或多个项目到 Listbox 中

– 使用 lb.insert(“end”) 添加新选项到末尾

itemcget(index, option)

– 获得 index 参数指定的项目对应的选项（由 option 参数指定）

itemconfig(index, \*\*options)

– 设置 index 参数指定的项目对应的选项（由可变参数 \*\*option 指定）

nearest(y)

– 返回与给定参数 y 在垂直坐标上最接近的项目的序号

scan\_dragto(x, y)

– 见下方 scan\_mark(x, y)

scan\_mark(x, y)

– 使用这种方式来实现 Listbox 内容的滚动

– 需要将鼠标按钮事件及当前鼠标位置绑定到 scan\_mark(x, y) 方法，然后再将 事件及当前鼠标位置绑定到 scan\_dragto(x, y) 方法，就可以实现 Listbox 在当前位置和 sacn\_mack(x, y) 指定的位置 (x, y) 之间滚动

see(index)

– 调整列表框的位置，使得 index 参数指定的选项是可见的

select\_anchor(index)

– 与 selection\_anchor(index) 相同，见下方解释

select\_clear(first, last=None)

– 与 selection\_clear(first, last=None) 相同，见下方解释

select\_includes(index)

– 与 selection\_includes(index) 相同，见下方解释

select\_set(first, last=None)

– 与 selection\_set(first, last=None) 相同，见下方解释

selection\_anchor(index)

– 在 index 参数的位置下一个锚点，此后你就可以通过特殊索引 “anchor” 访问

selection\_clear(first, last=None)

– 取消参数 first 到 last 范围内（包含 first 和 last）选项的选中状态

– 如果忽略 last 参数，则只取消 first 参数指定选项的选中状态

selection\_includes(index)

– 返回 index 参数指定的选项的选中状态

– 返回 1 表示选中，返回 0 表示未选中

selection\_set(first, last=None)

– 设置参数 first 到 last 范围内（包含 first 和 last）选项为选中状态

– 如果忽略 last 参数，则只设置 first 参数指定选项为选中状态

size()

– 返回 Listbox 组件中选项的数量

xview(\*args)

– 该方法用于在水平方向上滚动 Listbox 组件的内容，一般通过绑定 Scollbar 组件的 command 选项来实现

– 如果第一个参数是 “moveto”，则第二个参数表示滚动到指定的位置：0.0 表示最左端，1.0 表示最右端

–如果第一个参数是 “scroll”，则第二个参数表示滚动的数量，第三个参数表示滚动的单位（可以是 “units” 或 “pages”），例如：xview(“scroll”, 3, “pages”)表示向右滚动三行

xview\_moveto(fraction)

– 跟 xview(“moveto”, fraction) 一样

xview\_scroll(number, what)

– 跟 xview(“scroll”, number, what) 一样

yview(\*args)

– 该方法用于在垂直方向上滚动 Listbox 组件的内容，一般通过绑定 Scollbar 组件的 command 选项来实现（具体操作参考：Scrollbar）

– 如果第一个参数是 “moveto”，则第二个参数表示滚动到指定的位置：0.0 表示最顶端，1.0 表示最底端

– 如果第一个参数是 “scroll”，则第二个参数表示滚动的数量，第三个参数表示滚动的单位（可以是 “units” 或 “pages”），例如：yview(“scroll”, 3, “pages”) 表示向下滚动三页

yview\_moveto(fraction)

– 跟 yview(“moveto”, fraction) 一样

yview\_scroll(number, what)

– 跟 yview(“scroll”, number, what) 一样

# Tkinter组件详解之Scrollbar

Scrollbar（滚动条）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 组件通常几乎与 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。

用法

使用垂直滚动条。为了在某个组件上安装垂直滚动条，你需要做两件事：

设置该组件的 yscrollbarcommand 选项为 Scrollbar 组件的 set() 方法；

设置 Scrollbar 组件的 command 选项为该组件的 yview() 方法。

import tkinter as tk

root = tk.Tk()

sb = tk.Scrollbar(root)

sb.pack(side="right", fill="y")

lb = tk.Listbox(root, yscrollcommand=sb.set)

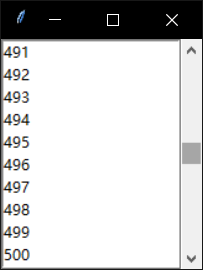
for i in range(1000):

lb.insert("end", str(i))

lb.pack(side="left", fill="both")

sb.config(command=lb.yview)

root.mainloop()



分析：当 Listbox 组件的可视范围发生改变的时候，Listbox 组件通过调用 set() 方法通知 Scrollbar 组件。而当用户操纵滚动条的时候，将自动调用 Listbox 组件的 yview() 方法。

添加水平滚动条方法跟上边一样，只是将 yscrollcommand 改为 xscrollcommand，yview 改为 xview 即可。

## 参数





## 方法

activate(element)

– 显示 element 参数指定的元素的背景颜色和样式

– element 参数可以设置为：“arrow1”（箭头1），“arrow2”（箭头2）或 “slider”（滑块）

delta(deltax, deltay)

– 给定一个鼠标移动的范围 deltax 和 deltay（像素为单位，deltax 表示水平移动量，deltay 表示垂直移动量），然后该方法返回一个浮点类型的值（范围 -1.0 ~ 1.0）

– 这通常在鼠标绑定上使用，用于确定当用户拖拽鼠标时滑块的如何移动

fraction(x, y)

– 给定一个像素坐标 (x, y)，该方法返回最接近给定坐标的滚动条位置（范围 0.0 ~ 1.0）

get()

– 返回当前滑块的位置 (a, b)

– a 值表示当前滑块的顶端或左端的位置，b 值表示当前滑块的底端或右端的位置（范围 0.0 ~ 1.0）

identify(x, y)

– 返回一个字符串表示指定位置下（如果有的话）的滚动条部件

– 返回值可以是：“arrow1”（箭头1），“arrow2”（箭头2）、“slider”（滑块）或 “”（啥都没有）

set(\*args)

– 设置当前滚动条的位置

– 如果设置则需要两个参数 (first, last)，first 表示当前滑块的顶端或左端的位置，last 表示当前滑块的底端或右端的位置（范围 0.0 ~ 1.0）

**H\_事件**

python学习笔记：tkinter之事件（event）、事件处理（event handler）、事件绑定（event binding）、基于事件的动画（animation）

时间:2019-02-25

本文章向大家介绍python学习笔记：tkinter之事件（event）、事件处理（event handler）、事件绑定（event binding）、基于事件的动画（animation），主要包括python学习笔记：tkinter之事件（event）、事件处理（event handler）、事件绑定（event binding）、基于事件的动画（animation）使用实例、应用技巧、基本知识点总结和需要注意事项，具有一定的参考价值，需要的朋友可以参考一下。

1. 事件（event）：是指点击、按键等操作，在tkinter中，event是一个类，当某个事件发生时，生成一个event对象，不同类型的事件生成具有不同属性的event对象。
2. 事件处理（event handler）：是指在捕获到事件后，程序自动执行的操作，是回调函数（recall function）。
3. 事件绑定（event binding）：是当一个事件发生时程序能够做出响应。tkinter提供三种绑定方式：实例绑定bind（将某个事件处理绑定到某个组件上）、类绑定bind\_class（将某个事件处理绑定到某类组件上）、应用绑定bind\_all（将某个事件处理绑定到所有组件上）。

from Tkinter import \*

root = Tk()

def eventHandler(event):

print((event.x, event.y))

canvas=Canvas(root, width=100, height=100)

root.bind("<Button-1>", eventHandler)

root.pack()

root.mainloop()

一、事件（event）

（1）事件格式：

在Tkinter中，事件的描述格式为：**<[modifier-]-type[-detail]>**，其中：

* modifier：事件修饰符。如：Alt、Shit组合键和Double事件。
* type：事件类型。如：按键（Key）、鼠标（Button/Motion/Enter/Leave/Relase）、Configure等。
* detail：事件细节。如：鼠标左键（1）、鼠标中键（2）、鼠标右键（3）。

**☆☆☆注意大小写！！！**

|  |  |  |
| --- | --- | --- |
| 事件类型 | **事件格式** | **事件解释** |
| 鼠标事件 | <Button-1> | 鼠标点击（1-左键，2-中键，3-右键） |
| <Double-Button-1> | 鼠标双击（1-左键，2-中键，3-右键） |
| <B1-Motion> | 鼠标拖动（1-左键，2-中键，3-右键） |
| <ButtonRelease-1> | 鼠标按下之后释放（1-左键，2-中键，3-右键） |
| <Enter> | 鼠标进入控件范围（widget），不是键盘按键 |
| <Leave> | 鼠标离开控件范围（widget） |
| 键盘事件 | <Key>/<KeyPress> | 任意键盘按键（键值会以char的格式放入event对象，event.keysym ） |
| <Return>  <Cancel>  <BackSpace>  <Tab>  <Shift\_L>  <Control\_L>  <Alt\_L>  <Home>  <Left>  <Up>  <Down>  <Right>  <Delete>  <F1>  <F2> | 对应键盘按键 |
| 组件事件 | <Configure> | 如果widget的大小发生改变，新的大小（width和height）会打包到event发往handler。 |
| <Activate> | 当组件从不可用变为可用 |
| <Deactivate> | 当组件从可用变为不可用 |
| <Destroy> | 当组件被销毁时 |
| <Expose> | 当组件从被遮挡状态变为暴露状态 |
| <Map> | 当组件由隐藏状态变为显示状态 |
| <Unmap> | 当组件由显示状态变为隐藏状态 |
| <FocusIn> | 当组件获得焦点时 |
| <FocusOut> | 当组件失去焦点时 |
| <Property> | 当组件属性发生改变时 |
| <Visibility> | 当组件变为可视状态时 |

（2）事件对象：

一个具体事件如<Button-1>是事件类（event class）的一个实例，事件类中设定了众多属性，其中部分属性是通用的，另一部分属性属于特定事件类型的，常用属性如下：

|  |  |  |
| --- | --- | --- |
| **属性** | **属性说明** | **适用事件类型** |
| **.char** | **如果按键事件产生通用ASCII字符，这个字符将赋值给event.char。（特殊ASCII字符，如delete等不属于该属性）** | <KeyPress>  <KeyRelease>等按键事件 |
| **.keysym** | **如果按键事件产生特殊ASCII字符，这个字符将赋值给event.keysym。** | <KeyPress>  <KeyRelease>等按键事件 |
| **.x** | **鼠标当前位置横坐标，相对于组件左上角** |  |
| **.y** | **鼠标当前位置纵坐标，相对于组件左上角** |  |
| **.x\_root** | **鼠标当前位置横坐标，相对于屏幕左上角** |  |
| **.y\_root** | **鼠标当前位置纵坐标，相对于屏幕左上角** |  |
| **.width** | **组件大小发生改变后的宽度** | <Configure> |
| **.height** | **组件大小发生改变后的高度** | <Configure> |
| **.type** | **事件类型** | ALL |

二、基于事件的动画（animation）

（1）模型-视图-控制器（MVC，Model、View and Controller）

        MVC是事件循环运行动画的三个核心内容：

* **Controller:** 当某个事件发生时，tkinter调用适当的（自定义的）事件处理函数（event handler function）或控制器（controller）。
* **Model:** 当某个事件发生后，事件处理函数/控制器会修改模型，这个模型就是存储了底层动画数据的地方，即：这个模型代表动画的状态，初始定义由初始化函数给出。
* **View:** 当事件处理结束后, Tkinterhui 会调用自定义的函数（redrawAll）更新视图，即利用模型内相关数据重新设定画布/视图。

（2）MVC必须遵循如下因果关系**：（C→M→V）**

* 控制器更新模型（C→M）
* 视图需使用来自模型的数据（M→V）
* 控制器不能直接影响视图（C→V（×））
* 视图不能修改模型（V→M（×））

**（3）一个例子：**

* 圆圈随按键位置改变
* 输出键盘键值

# Basic Animation Framework

from tkinter import \*

####################################

# customize these functions

####################################

# Set up the model data with init

# init is called once, at the beginning of the program

# data is a Struct, which can be given new data values using data.name = value

# data will be shared across all animation functions- it's aliased!

def init(data):

# data comes preset with width and height, from the run function

data.circleSize = min(data.width, data.height) / 10

data.circleX = data.width/2

data.circleY = data.height/2

data.charText = ""

data.keysymText = ""

# Track and respond to mouse clicks

# The event variable holds all of the data captured by the event loop

# For mousePressed, this is event.x and event.y, the position where

# the mouse was clicked

def mousePressed(event, data):

data.circleX = event.x

data.circleY = event.y

# Track and respond to key presses

# The event variable holds all of the data captured by the event loop

# For keyPressed, this is event.char and event.keysym

# event.char holds the direct key that was pressed, "a", "3", "@", etc.

# event.keysym holds special names for certain keys non-alphanumeric keys

# for example, "space", "BackSpace", "parenleft", "exclam"

def keyPressed(event, data):

data.charText = event.char

data.keysymText = event.keysym

# Draw graphics normally with redrawAll

# Main difference: the data struct contains helpful information to assist drawing

# Also, the canvas will get cleared and this will be called again

# constantly by the event loop.

def redrawAll(canvas, data):

canvas.create\_oval(data.circleX - data.circleSize,

data.circleY - data.circleSize,

data.circleX + data.circleSize,

data.circleY + data.circleSize)

if data.charText != "":

canvas.create\_text(data.width/10, data.height/3,

text="char: " + data.charText)

if data.keysymText != "":

canvas.create\_text(data.width/10, data.height\*2/3,

text="keysym: " + data.keysymText)

####################################

# use the run function as-is

####################################

def run(width=300, height=300):

def redrawAllWrapper(canvas, data):

canvas.delete(ALL)

canvas.create\_rectangle(0, 0, data.width, data.height,

fill='white', width=0)

redrawAll(canvas, data)

canvas.update()

def mousePressedWrapper(event, canvas, data):

mousePressed(event, data)

redrawAllWrapper(canvas, data)

def keyPressedWrapper(event, canvas, data):

keyPressed(event, data)

redrawAllWrapper(canvas, data)

# Set up data and call init

class Struct(object): pass

data = Struct()

data.width = width

data.height = height

root = Tk()

init(data)

# create the root and the canvas

canvas = Canvas(root, width=data.width, height=data.height)

canvas.pack()

# set up events

root.bind("<Button-1>", lambda event:

mousePressedWrapper(event, canvas, data))

root.bind("<Key>", lambda event:

keyPressedWrapper(event, canvas, data))

redrawAll(canvas, data)

# and launch the app

root.mainloop() # blocks until window is closed

print("bye!")

run(400, 200)