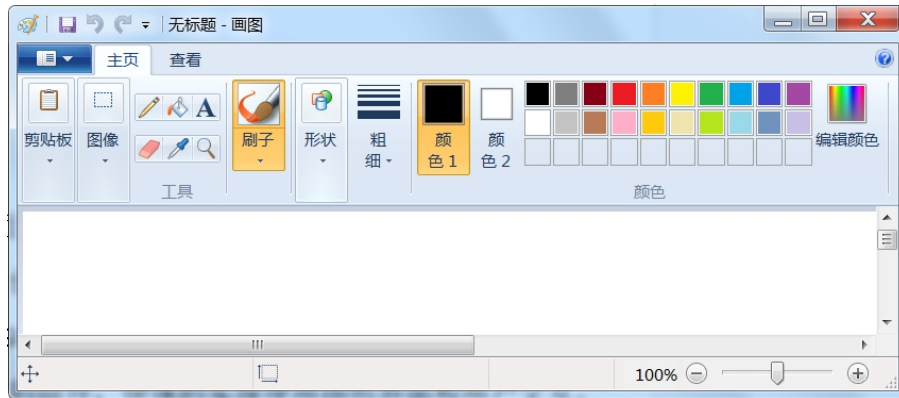


GUI 图形用户界面编程

我们前面实现的都是基于控制台的程序，程序和用户的交互通过控制台来完成。

本章，我们将学习 GUI (Graphics User Interface)，即图形用户界面编程，我们可以通过 python 提供的丰富的组件，快速的实现使用图形界面和用户交互。

GUI 编程类似于“搭积木”，将一个个组件(Widget)放到窗口中。如下是 windows 中的画图软件，就是一个典型的 GUI 程序：



上面的各种按钮、菜单、编辑区域等都是一个个组件，它们都放置到窗口中，并通过增加“对事件的处理”成为一个完整的程序。

常用的 GUI 库

1. Tkinter

tkinter(Tk interface)是 Python 的标准 GUI 库，支持跨平台的 GUI 程序开发。tkinter 适合小型的 GUI 程序编写，也特别适合初学者学习 GUI 编程。本书以 tkinter 为核心进行讲解。

2. wxPython

wxPython 是比较流行的 GUI 库，适合大型应用程序开发，功能强于 tkinter，整体设计框架类似于 MFC(Microsoft Foundation Classes 微软基础类库)。

3. PyQt

Qt 是一种开源的 GUI 库，适合大型 GUI 程序开发，PyQt 是 Qt 工具包标准的 Python 实现。我们也可以使用 Qt Designer 界面设计器快速开发 GUI 应用程序。

tkinter 模块

本章中，涉及大量的 API 讲解。学习 API 最好的来源就是官方提供的文档：tkinter 官

方网址:

<https://docs.python.org/3.7/library/tk.html>

或者: <http://effbot.org/tkinterbook/> (相对规整, 适合初学者查找)

由于官方都是英文, 我们在授课过程中尽量不涉及。英文好的同学可以自行查找相关说明。我们也希望英文较差的同学也能尽量多的学习英文, 对于后续技术的理解有较大的帮助。大家也可以去“北京尚学堂”官网下载我们提供的常见 1800 个开发词汇。

GUI 编程的核心步骤和第一个 GUI 程序

基于 tkinter 模块创建 GUI 程序包含如下 4 个核心步骤:

1. 创建应用程序主窗口对象 (也称: 根窗口)

(1) 通过类 Tk 的无参构造函数

```
from tkinter import *  
  
root = Tk()
```

2. 在主窗口中, 添加各种可视化组件, 比如: 按钮 (Button)、文本框 (Label) 等。

```
btn01 = Button(root)  
btn01["text"] = "点我就送花"
```

3. 通过几何布局管理器, 管理组件的大小和位置

```
btn01.pack()
```

4. 事件处理

(1) 通过绑定事件处理程序, 响应用户操作所触发的事件 (比如: 单击、双击等)

```
def songhua(e):  
    messagebox.showinfo("Message", "送你一朵玫瑰花, 请你爱上我")  
    print("送你 99 朵玫瑰花")  
  
btn01.bind("<Button-1>", songhua)
```

【示例】使用 tkinter 模块, 创建 GUI 应用程序, 并实现点击按钮的事件处理

```
from tkinter import *  
from tkinter import messagebox  
  
root = Tk()  
btn01 = Button(root)
```

```
btn01["text"] = "点我就送花"
btn01.pack()

def songhua(e):
    messagebox.showinfo("Message", "送你一朵玫瑰花，请你爱上我")
    print("送你 99 朵玫瑰花")

btn01.bind("<Button-1>", songhua)

root.mainloop()          #调用组件的 mainloop 方法，进入事件循环
```

tkinter 主窗口

主窗口位置和大小

通过 `geometry('wxh±x±y')` 进行设置。w 为宽度，h 为高度。+x 表示距屏幕左边的距离；-x 表示距屏幕右边的距离；+y 表示距屏幕上边的距离；-y 表示距屏幕下边的距离。

【示例】测试 tkinter 主窗口位置和大小的设置

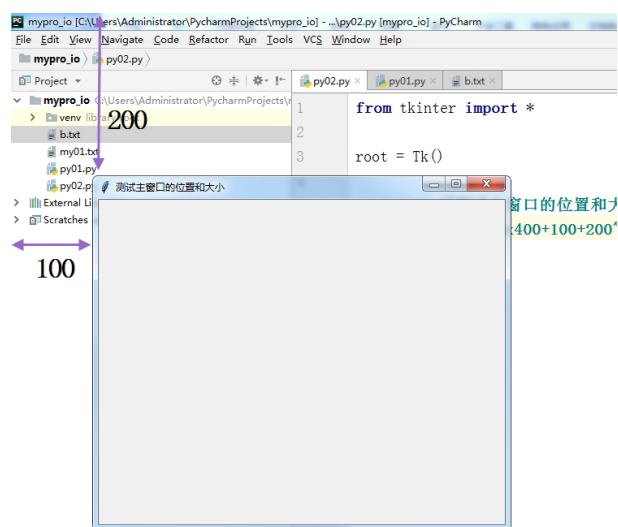
```
from tkinter import *

root = Tk()

root.title("测试主窗口的位置和大小")
root.geometry("500x400+100+200")  #宽度 500，高度 400；距屏幕左边 100，距屏幕上边 200

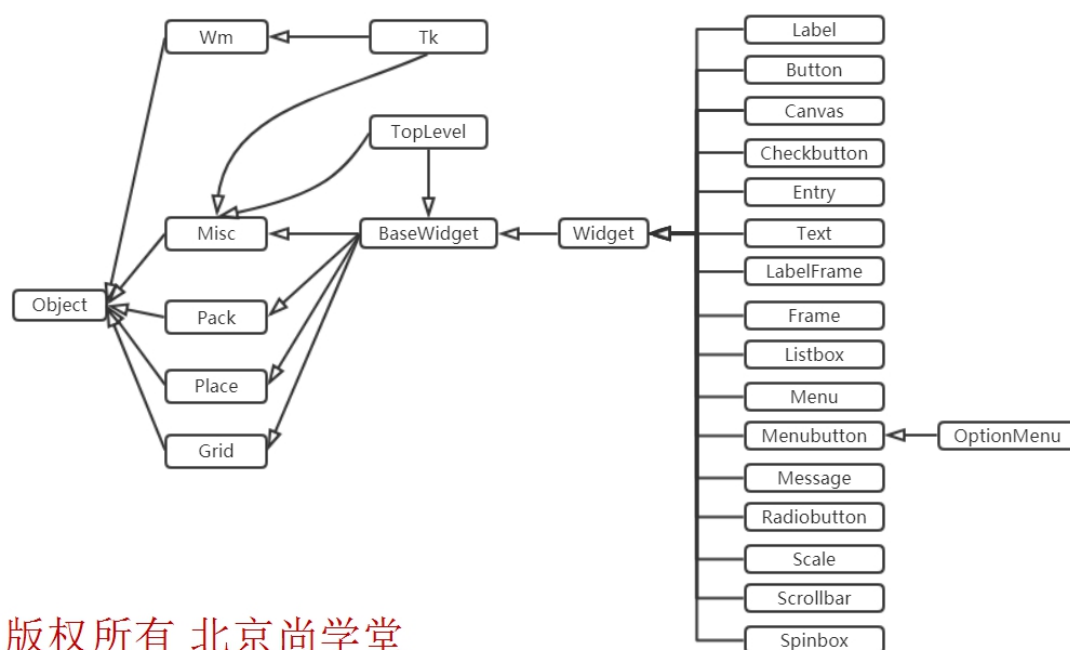
root.mainloop()
```

执行结果：



GUI 编程整体描述

图形用户界面是由一个个组件组成，就像小孩“搭积木”一样最终组成了整个界面。有的组件还能在里面再放置其他组件，我们称为“容器”。Tkinter 的 GUI 组件关系图如下：



版权所有 北京尚学堂

图 tkinter 中 GUI 组件的继承关系图

根据上图所示，我们依次讲解这些类的基本作用。

• Misc 和 Wm:

Tkinter 的 GUI 组件有两个根父类，它们都直接继承了 object 类：

- Misc: 它是所有组件的根父类。
- Wm: 它主要提供了一些与窗口管理器通信的功能函数。

• Tk

Misc 和 Wm 派生出子类 Tk，它代表应用程序的主窗口。一般应用程序都需要直接或间接使用 Tk。

• Pack、Place、Grid

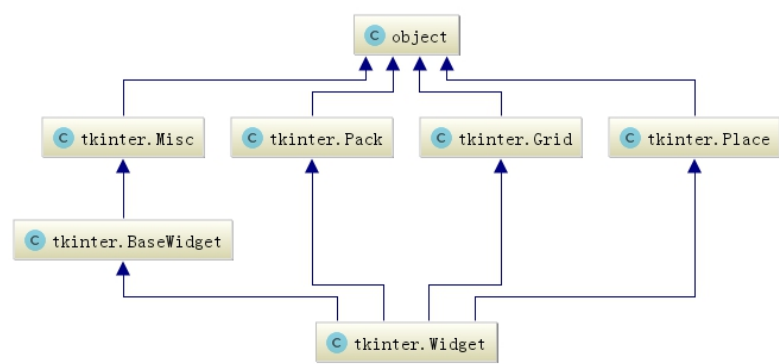
Pack、Place、Grid 是布局管理器。布局管理器管理组件的：大小、位置。通过布局管理器可以将容器中的组件实现合理的排布。

• BaseWidget

BaseWidget 是所有组件的父类

• Widget

Widget 是所有组件类的父类。Widget 一共有四个父类：BaseWidget、Pack、Grid、Place。意味着，所有 GUI 组件同时具备这四个父类的属性和方法。



【注】想观察类的层次结构可以在类定义处的类名上单击右键，选择 Diagram-->show Diagram。

常用组件汇总列表

| Tkinter 类 | 名称 | 简介 |
|-------------|-------|---------------------------------------|
| Toplevel | 顶层 | 容器类，可用于为其他组件提供单独的容器；Toplevel 有点类类似于窗口 |
| Button | 按钮 | 代表按钮组件 |
| Canvas | 画布 | 提供绘图功能，包括直线、矩形、椭圆、多边形、位图等 |
| Checkbutton | 复选框 | 可供用户勾选的复选框 |
| Entry | 单行输入框 | 用户可输入内容 |
| Frame | 容器 | 用于装载其它 GUI 组件 |

| | | |
|-------------|-------|--|
| Label | 标签 | 用于显示不可编辑的文本或图标 |
| LabelFrame | 容器 | 也是容器组件，类似于 Frame，但它支持添加标题 |
| Listbox | 列表框 | 列出多个选项，供用户选择 |
| Menu | 菜单 | 菜单组件 |
| Menubutton | 菜单按钮 | 用来包含菜单的按钮（包括下拉式、层叠式等） |
| OptionMenu | 菜单按钮 | Menubutton 的子类，也代表菜单按钮，可通过按钮打开一个菜单 |
| Message | 消息框 | 类似于标签，但可以显示多行文本；后来当 Label 也能显示多行文本之后，该组件基本处于废弃状态 |
| PanedWindow | 分区窗口 | 该容器会被划分成多个区域，每添加一个组件占一个区域，用户可通过拖动分隔线来改变各区域的大小 |
| Radiobutton | 单选钮 | 可供用户点边的单选钮 |
| Scale | 滑动条 | 拖动滑块可设定起始值和结束值，可显示当前位置的精确值 |
| Spinbox | 微调选择器 | 用户可通过该组件的向上、向下箭头选择不同的值 |
| Scrollbar | 滚动条 | 用于为组件（文本域、画布、列表框、文本框）提供滚动功能 |
| Text | 多行文本框 | 显示多行文本 |

GUI 应用程序类的经典写法

本节程序也是 GUI 应用程序编写的一个主要结构，采用了面向对象的方式，更加合理的组织代码。

通过类 Application 组织整个 GUI 程序，类 Application 继承了 Frame 及通过继承拥有了父类的特性。通过构造函数__init__() 初始化窗口中的对象，通过 createWidgets() 方法创建窗口中的对象。

Frame 框架是一个 tkinter 组件，表示一个矩形的区域。Frame 一般作为容器使用，可以放置其他组件，从而实现复杂的布局。

【示例】标准的 GUI 程序类的写法

```
"""测试一个经典的 GUI 程序的写法，使用面向对象的方式"""

from tkinter import *
from tkinter import messagebox

class Application(Frame):
```

"""一个经典的 GUI 程序的类的写法"""

```
def __init__(self, master=None):
    super().__init__(master)          # super() 代表的是父类的定义，而不是父类
    对象
    self.master = master
    self.pack()

    self.createWidget()

def createWidget(self):
    """创建组件"""
    self.btn01 = Button(self)
    self.btn01["text"] = "点击送花"
    self.btn01.pack()
    self.btn01["command"] = self.songhua

    # 创建一个退出按钮
    self.btnQuit = Button(self, text="退出", command=root.destroy)
    self.btnQuit.pack()

def songhua(self):
    messagebox.showinfo("送花", "送你 99 朵玫瑰花")

if __name__ == '__main__':
    root = Tk()
    root.geometry("400x100+200+300")
    root.title("一个经典的 GUI 程序类的测试")
    app = Application(master=root)
    root.mainloop()
```

简单组件

Label 标签

Label（标签）主要用于显示文本信息，也可以显示图像。

Label（标签）有这样一些常见属性：

1. width,height:

用于指定区域大小，如果显示是文本，则以单个英文字符大小为单位（一个汉字宽度占2个字符位置，高度和英文字符一样）；如果显示是图像，则以像素为单位。默认值是根据具体显示的内容动态调整。

2. font

指定字体和字体大小，如：font = (font_name,size)

3. image:

显示在 Label 上的图像，目前 tkinter 只支持 gif 格式。

4. fg 和 bg

fg (foreground):前景色、bg (background):背景色

5. justify

针对多行文字的对齐，可设置 justify 属性，可选值“left”，“center” or “right”

【示例】Label（标签）的用法

```
"""测试 Label 组件的基本用法，使用面向对象的方式"""

from tkinter import *

class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)          # super() 代表的是父类的定义，而不是父类
        # 对象
        self.master = master
        self.pack()
        self.createWidget()

    def createWidget(self):
        """创建组件"""
        self.label01 = Label(self, text="百战程序员", width=10, height=2,
                               bg="black", fg="white")
        self.label01["text"] = "ccc"
        self.label01.config(fg="red", bg="green")
        self.label01.pack()

        self.label02 = Label(self, text="高淇老师", width=10, height=2,
                               bg="blue", fg="white", font=("黑体", 30))
        self.label02.pack()

        # 显示图像
        global photo          # 把 photo 声明成全局变量。如果是局部变量，本方法执行完毕
                               # 后，图像对象销毁，窗口显示不出图像。
```

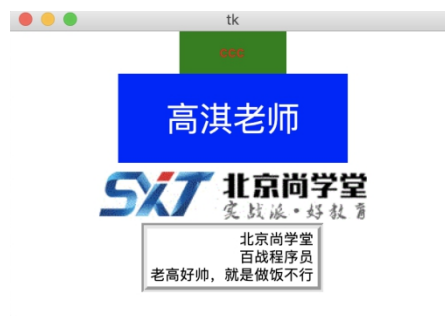


```
photo = PhotoImage(file="imgs/logo.gif")
self.label103 = Label(self, image=photo)
self.label103.pack()

self.label104 = Label(self, text="北京尚学堂\n 百战程序员\n 老高好帅，就
是做饭不行",
                      borderwidth=5, relief="groove", justify="right")
self.label104.pack()

if __name__ == '__main__':
    root = Tk()
    root.geometry("400x260+200+300")
    app = Application(master=root)
    root.mainloop()
```

运行结果如下：



Options 选项详解

通过学习 Label 组件，我们发现可以通过 Options 设置组件的属性，从而控制组件的各种状态。比如：宽度、高度、颜色、位置等等。

我们可以通过三种方式设置 Options 选项，这在各种 GUI 组件中用法都一致。

1. 创建对象时，使用可变参数

```
fred = Button(self, fg="red", bg="blue")
```

2. 创建对象后，使用字典索引方式

```
fred["fg"] = "red"
fred["bg"] = "blue"
```

3. 创建对象后，使用 config() 方法

```
fred.config(fg="red", bg="blue")
```

如何查看组件的 Options 选项：

- 1. 可以通过打印 config() 方法的返回值，查看 Options 选项
print(fred.config())
- 2. 通过在 IDE 中，点击组件对象的构造方法，进入到方法内观察：

```
#创建组件
btn01 = Button(root)
btn01["text"] = "送一朵花"
```

按ctrl+左键

我们可以看到如下的代码：

```
class Button(Widget):
    """Button widget."""
    def __init__(self, master=None, cnf={}, **kw):
        """Construct a button widget with the parent MASTER.

        STANDARD OPTIONS

        activebackground, activeforeground, anchor,
        background, bitmap, borderwidth, cursor,
        disabledforeground, font, foreground
        highlightbackground, highlightcolor,
        highlightthickness, image, justify,
        padx, pady, relief, repeatdelay,
        repeatinterval, takefocus, text,
        textvariable, underline, wraplength

        WIDGET-SPECIFIC OPTIONS

        command, compound, default, height,
        overrelief, state, width
        """
    Widget.__init__(self, master, 'button', cnf, kw)
```

上面代码中有：“standard options 标准选项”和“widget-specific options 组件特定选项”。我们将常见的选项汇总如下：

| 选项名（别名） | 含义 |
|------------------|---|
| activebackground | 指定组件处于激活状态时的背景色 |
| activeforeground | 指定组件处于激活状态时的前景色 |
| anchor | 指定组件内的信息（比如文本或图片）在组件中如何显示(当所在组件比信息大时，可以看出效果)。必须为下面的值之一：N、NE、E、SE、S、SW、W、NW 或 CENTER。比如 NW（NorthWest）指定将信息显示在组 |

| | |
|---------------------|---|
| | 件的左上角 |
| background(bg) | 指定组件正常显示时的背景色 |
| bitmap | 指定在组件上显示该选项指定的位图，该选项值可以是 Tk_GetBitmap 接收的任何形式的位图。位图的显示方式受 anchor、justify 选项的影响。如果同时指定了 bitmap 和 text，那么 bitmap 覆盖文本；如果同时指定了 bitmap 和 image，那么 image 覆盖 bitmap |
| borderwidth | 指定组件正常显示时的 3D 边框的宽度，该值可以是 Tk_GetPixels 接收的任何格式 |
| cursor | 指定光标在组件上的样式。该值可以是 Tk_GetCursors 接受的任何格式 |
| command | 指定按组件关联的命令方法，该方法通常在鼠标离开组件时被触发调用 |
| disabledforeground | 指定组件处于禁用状态时的前景色 |
| font | 指定组件上显示的文本字体 |
| foreground(fg) | 指定组件正常显示时的前景色 |
| highlightbackground | 指定组件在高亮状态下的背景色 |
| highlightcolor | 指定组件在高亮状态下的前景色 |
| highlightthickness | 指定组件在高亮状态下的周围方形区域的宽度，该值可以是 Tk_GetPixels 接收的任何格式 |
| height | 指定组件的高度，以 font 选项指定的字体的字符高度为单位，至少为 1 |
| image | 指定组件中显示的图像，如果设置了 image 选项，它将会覆盖 text、bitmap 选项 |
| justify | 指定组件内部内容的对齐方式，该选项支持 LEFT（左对齐）、CENTER（居中对齐）或 RIGHT（右对齐）这三个值 |
| padx | 指定组件内部在水平方向上两边的空白，该值可以是 Tk_GetPixels 接收的任何格式 |
| pady | 指定组件内部在垂直方向上两地的空白，该值可以是 Tk_GetPixels 接收的任何格式 |
| relief | 指定组件的 3D 效果，该选项支持的值包括 RAISED、SUNKEN、FLAT、RIDGE、SOLID、GROOVE。该值指出组件内部相对于外部的的外观样式，比如 RAISED 表示组件内部相对于外部凸起 |
| selectbackground | 指定组件在选中状态下的背景色 |
| selectborderwidth | 指定组件在选中状态下的 3D 边框的宽度，该值可以是 Tk_GetPixels 接收的任何格式 |
| selectforeground | 指定组在选中状态下的前景色 |
| state | 指定组件的当前状态。该选项支持 NORMAL（正常）、DISABLE（禁用）这两个值 |
| takefocus | 指定组件在键盘遍历（Tab 或 Shift+Tab）时是否接收焦点，将该选项设为 1 表示接收焦点；设为 0 表示不接收焦点 |
| text | 指定组件上显示的文本，文本显示格式由组件本身、anchor 及 |

| | |
|----------------|--|
| | justify 选项决定 |
| textvariable | 指定一个变量名，GUI 组件负责显示该变量值转换得到的字符串，文本显示格式由组件本身、anchor 及 justify 选项决定 |
| underline | 指定为组件文本的第几个字符添加下画线，该选项就相当于为组件绑定了快捷键 |
| width | 指定组件的宽度，以 font 选项指定的字体的字符高度为单位，至少为 1 |
| wraplength | 对于能支持字符换行的组件，该选项指定每行显示的最大字符数，超过该数量的字符将会转到下行显示 |
| xscrollcommand | 通常用于将组件的水平滚动改变（包括内容滚动或宽度发生改变）与水平滚动条的 set 方法关联，从而让组件的水平滚动改变传递到水平滚动条 |
| yscrollcommand | 通常用于将组件的垂直滚动改变（包括内容滚动或高度发生改变）与垂直滚动条的 set 方法关联，从而让组件的垂直滚动改变传递到垂直滚动条 |

Button

Button（按钮）用来执行用户的单击操作。Button 可以包含文本，也可以包含图像。按钮被单击后会自动调用对应事件绑定的方法。

【示例】Button 按钮用法(文字、图片、事件)

```
"""测试 Button 组件的基本用法，使用面向对象的方式"""

from tkinter import *
from tkinter import messagebox

class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)          # super() 代表的是父类的定义，而不是父类对象
        self.master = master
        self.pack()
        self.createWidget()

    def createWidget(self):
        """创建组件"""
        self.btn01 = Button(root, text="登录",
                               width=6,height=3,anchor=NE,command=self.login)
        self.btn01.pack()
```

```
global photo
photo = PhotoImage(file="imgs/start.gif")
self.btn02 = Button(root, image=photo, command=self.login)
self.btn02.pack()
self.btn02.config(state="disabled") #设置按钮为禁用

def login(self):
    messagebox.showinfo("尚学堂学习系统", "登录成功! 欢迎开始学习!")

if __name__ == '__main__':
    root = Tk()
    root.geometry("400x130+200+300")
    app = Application(master=root)
    root.mainloop()
```

运行结果:



Entry 单行文本框

Entry 用来接收一行字符串的控件。如果用户输入的文字长度长于 Entry 控件的宽度时，文字会自动向后滚动。如果想输入多行文本，需要使用 Text 控件。



【示例】Entry 单行文本框实现简单登录界面

"""测试 Entry 组件的基本用法，使用面向对象的方式"""

```
from tkinter import *
from tkinter import messagebox
```

```
class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)          # super() 代表的是父类的定义，而不是父类
        对象
        self.master = master
        self.pack()
        self.createWidget()

    def createWidget(self):
        """创建登录界面的组件"""
        self.label01 = Label(self, text="用户名")
        self.label01.pack()

        # StringVar 变量绑定到指定的组件。
        # StringVar 变量的值发生变化，组件内容也变化；
        # 组件内容发生变化，StringVar 变量的值也发生变化。
        v1 = StringVar()
        self.entry01 = Entry(self, textvariable=v1)
        self.entry01.pack()
        v1.set("admin")
        print(v1.get());print(self.entry01.get())

        # 创建密码框
        self.label02 = Label(self, text="密码")
        self.label02.pack()

        v2 = StringVar()
        self.entry02 = Entry(self, textvariable=v2, show="*")
        self.entry02.pack()

        Button(self, text="登陆", command=self.login).pack()

    def login(self):
        username = self.entry01.get()
        pwd = self.entry02.get()

        print("去数据库比对用户名和密码！")
        print("用户名：" + username)
        print("密码：" + pwd)
```

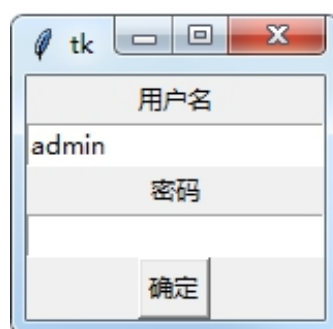
```

        if username=="gaoqi" and pwd=="123456":
            messagebox.showinfo("尚学堂学习系统", "登录成功! 欢迎开始学习!")
        else:
            messagebox.showinfo("尚学堂学习系统", "登录失败! 用户名或密码错误!")
    ")

if __name__ == '__main__':
    root = Tk()
    root.geometry("400x130+200+300")
    app = Application(master=root)
    root.mainloop()

```

界面效果:



Text 多行文本框

Text(多行文本框)的主要用于显示多行文本, 还可以显示网页链接, 图片, HTML 页面, 甚至 CSS 样式表, 添加组件等。因此, 也常被当做简单的文本处理器、文本编辑器或者网页浏览器来使用。比如 IDLE 就是 Text 组件构成的。

【示例】Text 多行文本框基本用法(文本输入、组件和图像显示)

```

"""测试 Text 多行文本框组件的基本用法, 使用面向对象的方式"""

from tkinter import *
import webbrowser

class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)          # super()代表的是父类的定义, 而不是父类对象
        self.master = master
        self.pack()
        self.createWidget()

```

```
def createWidget(self):
    self.w1 = Text(root, width=40, height=12, bg="gray")
    # 宽度 20 个字母(10 个汉字), 高度一个行高
    self.w1.pack()

    self.w1.insert(1.0, "0123456789\nabcdefg")
    self.w1.insert(2.3, "锄禾日当午, 汗滴禾下土。谁知盘中餐, 粒粒皆辛苦\n")

    Button(self, text="重复插入文本", command=self.insertText).pack(side="left")
    Button(self, text="返回文本", command=self.returnText).pack(side="left")
    Button(self, text="添加图片", command=self.addImage).pack(side="left")
    Button(self, text="添加组件", command=self.addWidget).pack(side="left")
    Button(self, text="通过 tag 精确控制文本", command=self.testTag).pack(side="left")

def insertText(self):
    # INSERT 索引表示在光标处插入
    self.w1.insert(INSERT, ' Gaoqi ')
    # END 索引号表示在最后插入
    self.w1.insert(END, '[sxt]')
    self.w1.insert(1.8, "gaoqi")

def returnText(self):
    # Indexes(索引)是用来指向 Text 组件中文本的位置, Text 的组件索引也是对应
    # 实际字符之间的位置。
    # 核心: 行号以 1 开始 列号以 0 开始
    print(self.w1.get(1.2, 1.6))
    print("所有文本内容: \n"+self.w1.get(1.0, END))

def addImage(self):
    # global photo
    self.photo = PhotoImage(file="imgs/logo.gif")
    self.w1.image_create(END, image=self.photo)

def addWidget(self):
    b1 = Button(self.w1, text='爱尚学堂')
    # 在 text 创建组件的命令
    self.w1.window_create(INSERT, window=b1)
```



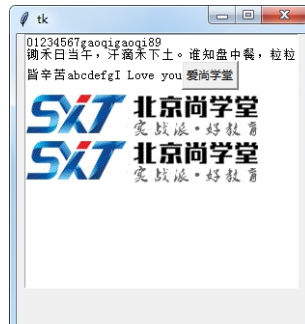
```
def testTag(self):
    self.w1.delete(1.0, END)
    self.w1.insert(INSERT, "good good study, day day up!\n 北京尚学堂\n 百战
程序员\n 百度，搜一下就知道")
    self.w1.tag_add("good", 1.0, 1.9)
    self.w1.tag_config("good", background="yellow", foreground="red")

    self.w1.tag_add("baidu", 4.0, 4.2)
    self.w1.tag_config("baidu", underline=True)
    self.w1.tag_bind("baidu", "<Button-1>", self.webshow)

def webshow(self, event):
    webbrowser.open("http://www.baidu.com")

if __name__ == '__main__':
    root = Tk()
    root.geometry("450x300+200+300")
    app = Application(master=root)
    root.mainloop()
```

运行结果:



- 利用 Tags 实现更加强大的文本显示和控制

Tags 通常用于改变 Text 组件中内容的样式和功能。你可以修改文本的字体、尺寸和颜色。另外，Tags 还允许你将文本、嵌入的组件和图片与鼠标和键盘等事件相关联。

【示例】利用 Tag 属性实现更复杂文本控制

```
#coding=utf-8
from tkinter import *
import webbrowser

root = Tk();root.geometry("300x300+400+400")

w1 = Text(root,width=40,height=20) #宽度 20 个字母(10 个汉字)，高度一个行高
w1.pack()

w1.insert(INSERT,"good good study,day day up!\n北京尚学堂\n百战程序员\n百度,
```

搜一下就知道”)

```
w1.tag_add("good", 1.0, 1.9)
w1.tag_config("good", background="yellow", foreground="red")

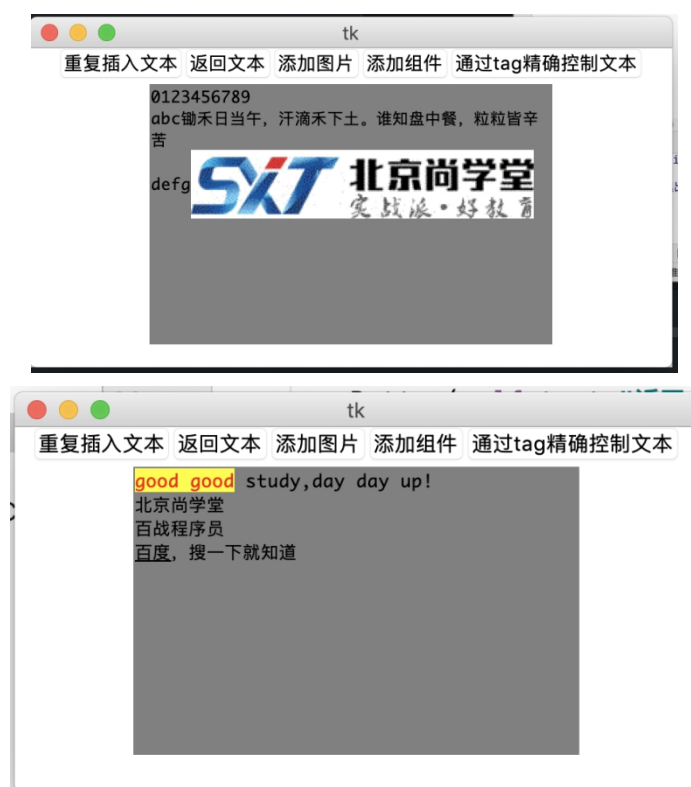
w1.tag_add("baidu", 4.0, 4.2)
w1.tag_config("baidu", underline=True)

def webshow(event):
    webbrowser.open("http://www.baidu.com")

w1.tag_bind("baidu", "<Button-1>", webshow)

root.mainloop()
```

运行结果：



点击“百度”后，系统默认浏览器打开百度页面。

Radiobutton 单选按钮

Radiobutton 控件用于选择同一组单选按钮中的一个。Radiobutton 可以显示文本，也可以显示图像。

【示例】Radiobutton 基础用法

```
"""测试 Radiobutton 组件的基本用法，使用面向对象的方式"""
```

```
from tkinter import *
from tkinter import messagebox

class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)      # super()代表的是父类的定义，而不是父类
        # 对象
        self.master = master
        self.pack()
        self.createWidget()

    def createWidget(self):
        self.v = StringVar()
        self.v.set("F")

        self.r1 = Radiobutton(self, text="男性", value="M", variable=self.v)
        self.r2 = Radiobutton(self, text="女性", value="F", variable=self.v)

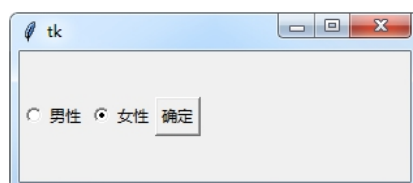
        self.r1.pack(side="left"); self.r2.pack(side="left")

        Button(self, text="确定", command=self.confirm).pack(side="left")

    def confirm(self):
        messagebox.showinfo("测试", "选择的性别:" + self.v.get())

if __name__ == '__main__':
    root = Tk()
    root.geometry("400x50+200+300")
    app = Application(master=root)
    root.mainloop()
```

运行结果：



Checkbox 复选按钮

Checkbox 控件用于选择多个按钮的情况。Checkbox 可以显示文本，也可以显示图像。

【示例】 Checkbutton 复选按钮用法

```
"""测试 Checkbutton 组件的基本用法，使用面向对象的方式"""

from tkinter import *
from tkinter import messagebox

class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)          # super()代表的是父类的定义，而不是父类
        对象
        self.master = master
        self.pack()
        self.createWidget()

    def createWidget(self):
        self.codeHobby = IntVar();
        self.videoHobby = IntVar()

        print(self.codeHobby.get())  # 默认值是 0
        self.c1 = Checkbutton(self, text="敲代码",
                               variable=self.codeHobby, onvalue=1, offvalue=0)
        self.c2 = Checkbutton(self, text="看视频",
                               variable=self.videoHobby, onvalue=1, offvalue=0)

        self.c1.pack(side="left");self.c2.pack(side="left")

        Button(self, text="确定", command=self.confirm).pack(side="left")

    def confirm(self):
        if self.videoHobby.get() == 1:
            messagebox.showinfo("测试", "看视频，都是正常人有的爱好！你喜欢看什
            么类型？")
        if self.codeHobby.get() == 1:
            messagebox.showinfo("测试", "抓获野生程序猿一只，赶紧送给他尚学堂的
            视频充饥")

if __name__ == '__main__':
    root = Tk()
```

```
root.geometry("400x50+200+300")
app = Application(master=root)
root.mainloop()
```

运行结果：



canvas 画布

canvas（画布）是一个矩形区域，可以放置图形、图像、组件等。本节我们简单介绍 canvas 的使用，更加详细和深入的内容将在后面的“图形绘制”章节讲解。

【示例】canvas 画布使用基础示例

"""测试 Canvas 组件的基本用法，使用面向对象的方式"""

```
from tkinter import *
from tkinter import messagebox
import random
```

```
class Application(Frame):
```

```
    def __init__(self, master=None):
        super().__init__(master)          # super() 代表的是父类的定义，而不是父类
        # 对象
        self.master = master
        self.pack()
        self.createWidget()
```

```
    def createWidget(self):
        self.canvas = Canvas(self, width=300, height=200, bg="green")
        self.canvas.pack()
        # 画一条直线
        line = self.canvas.create_line(10, 10, 30, 20, 40, 50)
        # 画一个矩形.
        rect = self.canvas.create_rectangle(50, 50, 100, 100)
        # 画一个椭圆. 坐标两双. 为椭圆的边界矩形左上角和底部右下角
        oval = self.canvas.create_oval(50, 50, 100, 100)
```

```

global photo
photo = PhotoImage(file="imgs/logo.gif")
self.canvas.create_image(150, 170, image=photo)

Button(self, text="画 10 个矩形",
command=self.draw50Recg).pack(side="left")

def draw50Recg(self):
    for i in range(0, 10):
        x1 = random.randrange(int(self.canvas["width"])/2)
        y1 = random.randrange(int(self.canvas["height"])/2)
        x2 = x1 + random.randrange(int(self.canvas["width"])/2)
        y2 = y1 + random.randrange(int(self.canvas["height"])/2)
        self.canvas.create_rectangle(x1, y1, x2, y2)

if __name__ == '__main__':
    root = Tk()
    root.geometry("400x300+200+300")
    app = Application(master=root)
    root.mainloop()

```

运行结果：



布局管理器

一个 GUI 应用程序必然有大量的组件,这些组件如何排布? 这时候,就需要使用 tkinter 提供的布局管理器帮助我们组织、管理在父组件中子组件的布局方式。tkinter 提供了三种管理器: pack、grid、place。

grid 布局管理器

grid 表格布局,采用表格结构组织组件。子组件的位置由行和列的单元格来确定,并

且可以跨行和跨列，从而实现复杂的布局。

grid() 方法提供的选项

| 选项 | 说明 | 取值范围 |
|--------------|----------------------------------|--|
| column | 单元格的列号 | 从 0 开始的正整数 |
| columnspan | 跨列，跨越的列数 | 正整数 |
| row | 单元格的行号 | 从 0 开始的正整数 |
| rowspan | 跨行，跨越的行数 | 正整数 |
| ipadx, ipady | 设置子组件之间的间隔，x 方向或者 y 方向，默认单位为像素 | 非负浮点数，默认 0.0 |
| padx, pady | 与之并列的组件之间的间隔，x 方向或者 y 方向，默认单位是像素 | 非负浮点数，默认 0.0 |
| sticky | 组件紧贴所在单元格的某一角，对应于东南西北中以及 4 个角 | “n”，“s”，“w”，“e”， “nw”，“sw”，“se”， “ne”，“center”（默认） |

【示例】grid 布局用法-登录界面设计

```
"""测试 Grid 布局管理器的基本用法，使用面向对象的方式"""

from tkinter import *
from tkinter import messagebox
import random

class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)          # super() 代表的是父类的定义，而不是父类
        对象
        self.master = master
        self.pack()
        self.createWidget()

    def createWidget(self):
        """通过 grid 布局实现登录界面"""
        self.label01 = Label(self, text="用户名")
        self.label01.grid(row=0, column=0)
        self.entry01 = Entry(self)
        self.entry01.grid(row=0, column=1)
        Label(self, text="用户名为手机号").grid(row=0, column=2)
```

```

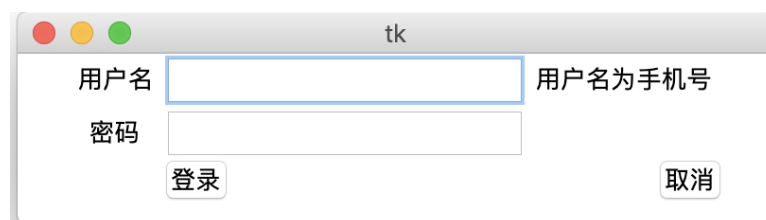
Label(self, text="密码").grid(row=1, column=0)
Entry(self, show="*").grid(row=1, column=1)

Button(self, text="登录").grid(row=2, column=1, sticky=EW)
Button(self, text="取消").grid(row=2, column=2, sticky=E)

if __name__ == '__main__':
    root = Tk()
    root.geometry("400x90+200+300")
    app = Application(master=root)
    root.mainloop()

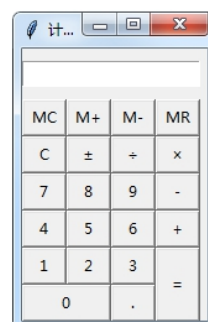
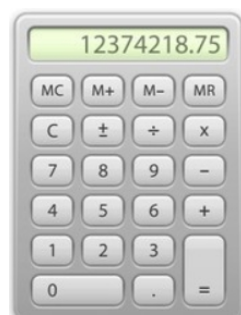
```

运行结果：



【示例】通过 grid 布局-实现计算器软件界面。

根据实际简易计算器的按键分布，设计一个相仿的计算器界面，相应的功能暂不需要实现。



如上界面，实际可以设计成一个 7 行 4 列的表格布局，然后将相应的按钮放置进去即可。

"""计算器软件界面的设计"""

```

from tkinter import *
from tkinter import messagebox
import random

```

```

class Application(Frame):

```



```
def __init__(self, master=None):
    super().__init__(master)    # super() 代表的是父类的定义，而不是父类
    对象
    self.master = master
    self.pack()
    self.createWidget()

def createWidget(self):
    """通过 grid 布局实现计算器的界面"""
    btnText = (("MC", "M+", "M-", "MR"),
                ("C", "±", "/", " "),
                (7, 8, 9, "-"),
                (4, 5, 6, "+"),
                (1, 2, 3, "="),
                (0, "."))

    Entry(self).grid(row=0, column=0, columnspan=4, pady=10)

    for rindex, r in enumerate(btnText):
        for cindex, c in enumerate(r):
            if c == "=":
                Button(self, text=c, width=2) \
                    .grid(row=rindex+1, column=cindex, rowspan=2, sticky=NSEW)
            elif c == 0:
                Button(self, text=c, width=2) \
                    .grid(row=rindex+1, column=cindex, columnspan=2, sticky=NSEW)

            elif c == ".":
                Button(self, text=c, width=2) \
                    .grid(row=rindex+1, column=cindex+1, sticky=NSEW)
            else:
                Button(self, text=c, width=2) \
                    .grid(row=rindex+1, column=cindex, sticky=NSEW)

if __name__ == '__main__':
    root = Tk()
    root.geometry("200x200+200+300")
    app = Application(master=root)
    root.mainloop()
```

pack 布局管理器

pack 按照组件的创建顺序将子组件添加到父组件中，按照垂直或者水平的方向自然排布。如果不指定任何选项，默认在父组件中自顶向下垂直添加组件。

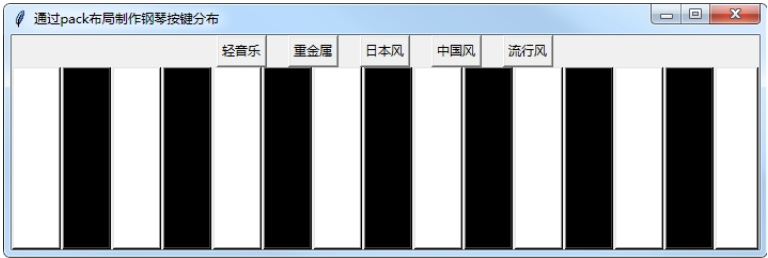
pack 是代码量最少，最简单的一种，可以用于快速生成界面。

| pack() 方法提供的选项 | | |
|----------------|--|--|
| 名称 | 描述 | 取值范围 |
| expand | 当值为“yes”时，side 选项无效。组件显示在父配件中心位置；若 fill 选项为” both” ,则填充父组件的剩余空间 | “yes”，自然数, ”no”，0（默认值” no” 或 0） |
| fill | 填充 x(y) 方向上的空间，当属性 side=” top” 或” bottom” 时，填充 x 方向；当属性 side=” left” 或” right” 时，填充” y” 方向；当 expand 选项为” yes” 时，填充父组件的剩余空间 | “x”， “y”， “both”， “none”（默认值为 none） |
| ipadx, ipady | 设置子组件之间的间隔， x 方向或者 y 方向，默认单位为像素 | 非负浮点数，默认 0.0 |
| padx, pady | 与之并列的组件之间的间隔， x 方向或者 y 方向，默认单位是像素 | 非负浮点数，默认 0.0 |
| side | 定义停靠在父组件的哪一边上 | “ top ”， “ bottom ”， “left”， “right”（默认为” top”） |
| before | 将本组件于所选组建对象之前 pack, 类似于先创建本组件再创建选定组件 | 已经 pack 后的组件对象 |
| after | 将本组件于所选组建对象之后 pack, 类似于先创建选定组件再本组件 | 已经 pack 后的组件对象 |
| in_ | 将本组件作为所选组建对象的子组件，类似于指定本组件的 master 为选定组件 | 已经 pack 后的组件对象 |
| anchor | 对齐方式，左对齐” w”，右对齐” e”，顶对齐” n”，底对齐” s” | “n”， “s”， “w”， “e”， “nw”， “sw”， “se”， “ne”， “center”（默认） |

【老鸟建议】如上列出了 pack 布局所有的属性，但是不需要挨个熟悉，了解基本的即可。

pack 适用于简单的垂直或水平排布，如果需要复杂的布局可以使用 grid 或 place。

【示例】pack 布局用法，制作钢琴按键布局



```
#coding=utf-8
#测试 pack 布局管理

from tkinter import *

root = Tk();root.geometry("700x220")

#Frame 是一个矩形区域，就是用来防止其他子组件
f1 = Frame(root)
f1.pack()
f2 = Frame(root);f2.pack()

btnText = ("流行风","中国风","日本风","重金属","轻音乐")

for txt in btnText:
    Button(f1,text=txt).pack(side="left",padx="10")

for i in range(1,20):
    Button(f2,width=5,height=10,bg="black" if i%2==0 else
"white").pack(side="left")

root.mainloop()
```

place 布局管理器

place 布局管理器可以通过坐标精确控制组件的位置，适用于一些布局更加灵活的场景。

place() 方法的选项

| 选项 | 说明 | 取值范围 |
|-------------------------|-------------------|---|
| x, y | 组件左上角的绝对坐标（相对于窗口） | 非负整数 x 和 y 选项用于设置偏移（像素），如果同时设置 relx(rely) 和 x(y)，那么 place 将优先计算 relx 和 rely，然后再实现 x 和 y 指定的偏移值 |
| relx rely | 组件左上角的坐标（相对于父容器） | relx 是相对父组件的位置。0 是最左边，0.5 是正中间，1 是最右边； rely 是相对父组件的位置。0 是最上边，0.5 是正中间，1 是最下边； |
| width, height | 组件的宽度和高度 | 非负整数 |
| relwid th, relhei | 组件的宽度和高度（相对于父容器） | 与 relx、rely 取值类似，但是相对于父组件的尺寸 |

| | | |
|--------|----------------------------------|--|
| ght | | |
| anchor | 对齐方式，左对齐”w”，右对齐”e”，顶对齐”n”，底对齐”s” | “n”，“s”，“w”，“e”，“nw”，“sw”，“se”，“ne”，“center”（默认） |

【示例】place 布局管理-基本用法测试

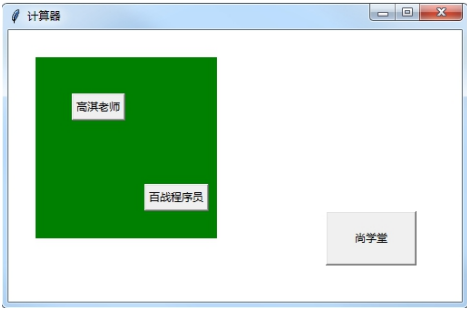
```
#coding=utf-8
from tkinter import *

root = Tk();root.geometry("500x300")
root.title("布局管理 place");root["bg"]="white"

f1 = Frame(root,width=200,height=200,bg="green")
f1.place(x=30,y=30)

Button(root,text="尚学堂").place(relx=0.5,relx=0,y=100,relwidth=0.2,relheight=0.2)
Button(f1,text="百战程序员").place(relx=0.6,relx=0,y=0.7)
Button(f1,text="高淇老师").place(relx=0.2,relx=0,y=0.2)
root.mainloop()
```

运行结果：



【示例】place 布局管理-扑克牌游戏 demo



"""扑克牌游戏的界面设计"""

```
from tkinter import *
```

```
class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)          # super()代表的是父类的定义，而不是父类
        对象
        self.master = master
        self.pack()
        self.createWidget()

    def createWidget(self):
        """通过 place 布局管理器实现扑克牌位置控制"""
        # self.photo = PhotoImage(file="imgs/puke/puke1.gif")
        # self.puke1 = Label(self.master, image=self.photo)
        # self.puke1.place(x=10, y=50)

        self.photos = [PhotoImage(file="imgs/puke/puke"+str(i+1)+".gif") for i in
range(10)]
        self.pukes = [Label(self.master, image=self.photos[i]) for i in range(10)]

        for i in range(10):
            self.pukes[i].place(x=10+i*40, y=50)

        # 为所有的 Label 增加事件处理
        self.pukes[0].bind_class("Label", "<Button-1>", self.chupai)

    def chupai(self, event):
        print(event.widget.winfo_geometry())
        print(event.widget.winfo_y())

        if event.widget.winfo_y() == 50:
            event.widget.place(y=30)
        else:
            event.widget.place(y=50)

if __name__ == '__main__':
    root = Tk()
    root.geometry("600x270+200+300")
    app = Application(master=root)
    root.mainloop()
```

事件处理

一个 GUI 应用整个生命周期都处在一个消息循环（event loop）中。它等待事件的发生，并作出相应的处理。

Tkinter 提供了用以处理相关事件的机制。处理函数可被绑定给各个控件的各种事件。

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

如果相关事件发生，handler 函数会被触发，事件对象 event 会传递给 handler 函数。

鼠标和键盘事件

| 代码 | 说明 |
|--------------------------------------|-------------------------------------|
| <Button-1> <ButtonPress-1> <1> | 鼠标左键按下。 2 表示右键，3 表示中键； |
| <ButtonRelease-1> | 鼠标左键释放 |
| <B1-Motion> | 按住鼠标左键移动 |
| <Double-Button-1> | 双击左键 |
| <Enter> | 鼠标指针进入某一组件区域 |
| <Leave> | 鼠标指针离开某一组件区域 |
| <MouseWheel> | 滚动滚轮； |
| <KeyPress-a> | 按下 a 键，a 可用其他键替代 |
| <KeyRelease-a> | 释放 a 键。 |
| <KeyPress-A> | 按下 A 键（大写的 A） |
| <Alt-KeyPress-a> | 同时按下 alt 和 a；alt 可用 ctrl 和 shift 替代 |
| <Double-KeyPress-a> | 快速按两下 a |
| <Control-V> | CTRL 和 V 键被同时按下，V 可以换成其它键位 |

event 对象常用属性

| 名称 | 说明 |
|---------|--|
| char | 按键字符，仅对键盘事件有效 |
| keycode | 按键编码，仅对键盘事件有效 |
| keysym | 按键名称，仅对键盘事件有效 比如按下空格键： 键的 char: 键的 keycode: 32 键的 keysym: space 比如按下 a 键： 键的 char: a 键的 keycode: 65 键的 keysym: a |
| num | 鼠标按键，仅对鼠标事件有效 |

| | |
|---------------|-------------------------|
| type | 所触发的事件类型 |
| widget | 引起事件的组件 |
| width,height | 组件改变后的大小，仅 Configure 有效 |
| x, y | 鼠标当前位置，相对于父容器 |
| x_root,y_root | 鼠标当前位置，相对于整个屏幕 |

【示例】鼠标事件和键盘事件用法测试

```
# coding=utf-8
# 测试键盘和鼠标事件

from tkinter import *

root = Tk();root.geometry("530x300")

c1 = Canvas(root,width=200,height=200,bg="green")
c1.pack()

def mouseTest(event):
    print("鼠标左键单击位置(相对于父容器): {0}, {1}".format(event.x, event.y))
    print("鼠标左键单击位置(相对于屏幕): {0}, {1}".format(event.x_root, event.y_root))
    print("事件绑定的组件: {0}".format(event.widget))

def testDrag(event):
    c1.create_oval(event.x, event.y, event.x+1, event.y+1)

def keyboardTest(event):
    print("键的 keycode: {0}, 键的 char: {1}, 键的 keysym: {2}"
          .format(event.keycode, event.char, event.keysym))

def press_a_test(event):
    print("press a")

def release_a_test(event):
    print("release a")

c1.bind("<Button-1>", mouseTest)

c1.bind("<B1-Motion>", testDrag)

root.bind("<KeyPress>", keyboardTest)
root.bind("<KeyPress-a>", press_a_test) #只针对小写的a，大写的A不管用
```

```
root.bind("<KeyRelease-a>", release_a_test)

root.mainloop()
```

lambda 表达式详解

lambda 表达式定义的是一个匿名函数，只适合简单输入参数，简单计算返回结果，不适合功能复杂情况。

lambda 定义的匿名函数也有输入、也有输出，只是没有名字。语法格式如下：

lambda 参数值列表：表达式

参数值列表即为输入。
表达式计算的结构即为输出。

我们写一个最简单的案例：

```
add3args = lambda x, y, z: x+y+z
#print(add3args(10, 20, 30))
```

上面的 lambda 表达式相当于如下函数定义：

```
def add3args(x, y, z):
    return x+y+z
```

lambda 表达式的参数值列表可以为如下内容：

| lambda 格式 | 说明 |
|-------------------------|----------------------------|
| lambda x, y: x*y | 函数输入是 x 和 y，输出是它们的积 x*y |
| lambda:None | 函数没有输入参数，输出是 None |
| lambda:aaa(3, 4) | 函数没有输入参数，输出是 aaa(3, 4) 的结果 |
| lambda *args: sum(args) | 输入是任意个数的参数，输出是它们的和 |
| lambda **kwargs: 1 | 输入是任意键值对参数，输出是 1 |

我们在平时使用时，注意 lambda 只是一个匿名函数（没有名字的函数），功能不强，不要过度使用；

使用 lambda 表达式实现传参

【示例】使用 lambda 帮助 command 属性绑定时传参

```
# coding=utf-8
# 测试 command 属性绑定事件，测试 lambda 表达式帮助传参

from tkinter import *
```



```
root = Tk();root.geometry("270x50")

def mouseTest1():
    print("command 方式，简单情况：不涉及获取 event 对象，可以使用")

def mouseTest2(a,b):
    print("a={0}, b={1}".format(a,b))

Button(root, text="测试 command1",
        command=mouseTest1).pack(side="left")

Button(root, text="测试 command2",
        command=lambda: mouseTest2("gaoqi", "xixi")).pack(side="left")

root.mainloop()
```

多种事件绑定方式汇总

• 组件对象的绑定

1. 通过 command 属性绑定（适合简单不需获取 event 对象）
Button(root, text=" 登录 ", command=login)
2. 通过 bind() 方法绑定（适合需要获取 event 对象）
c1 = Canvas(); c1.bind("<Button-1>", drawLine)

• 组件类的绑定

调用对象的 bind_class 函数，将该组件类所有的组件绑定事件：
w.bind_class("Widget", "event", eventhanler)

比如：btn01.bind_class("Button", "<Button-1>", func)

【示例】多种事件绑定方式总结

```
# coding=utf-8
# 多种事件绑定方式汇总
from tkinter import *

root = Tk();root.geometry("270x30")

def mouseTest1(event):
    print("bind() 方式绑定，可以获取 event 对象")
```

```
print(event.widget)

def mouseTest2(a, b):
    print("a={0},b={1}".format(a, b))
    print("command 方式绑定, 不能直接获取 event 对象")

def mouseTest3(event):
    print("右键单击事件, 绑定给所有按钮啦!!")
    print(event.widget)

b1 = Button(root, text="测试 bind() 绑定")
b1.pack(side="left")
# bind 方式绑定事件
b1.bind("<Button-1>", mouseTest1)

# command 属性直接绑定事件
b2 = Button(root, text="测试 command2",
             command=lambda: mouseTest2("gaoqi", "xixi"))
b2.pack(side="left")

# 给所有 Button 按钮都绑定右键单击事件<Button-2>
b1.bind_class("Button", "<Button-2>", mouseTest3)

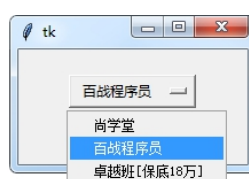
root.mainloop()
```

其他组件

我们在前面介绍了最常用的几个组件，接下来我们介绍其他一些组件。

OptionMenu 选择项

OptionMenu(选择项)用来做多选一，选中的项在顶部显示。显示效果如下：



【示例】OptionMenu(选择项)的基本用法

```
"""optionmenu 的使用测试"""

from tkinter import *

root = Tk();root.geometry("200x100")
v = StringVar(root)
v.set("百战程序员")
om = OptionMenu(root, v, "尚学堂", "百战程序员", "卓越班[保底 18 万]")

om["width"] = 10
om.pack()

def test1():
    print("最喜爱的机构:", v.get())
    # v.set("尚学堂") # 直接修改了 optionmenu 中选中的值

Button(root, text="确定", command=test1).pack()

root.mainloop()
```

Scale 移动滑块

Scale(移动滑块)用于在指定的数值区间，通过滑块的移动来选择值。

【示例】使用 Scale(移动滑块)控制字体大小变化

```
"""scale 滑块的使用测试"""

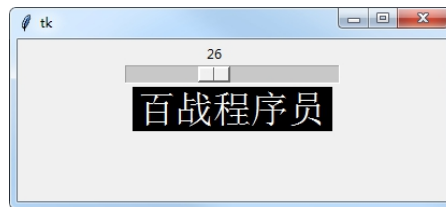
from tkinter import *

root = Tk();root.geometry("400x150")

def test1(value):
    print("滑块的值:", value)
    newFont = ("宋体", value)
    a.config(font=newFont)
```

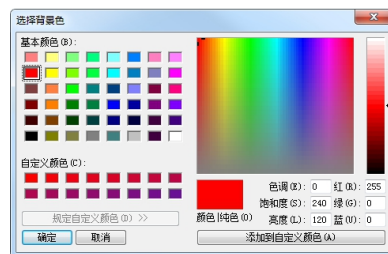
```
s1 =  
Scale(root, from_=10, to=50, length=200, tickinterval=5, orient=HORIZONTAL, command=test1)  
s1.pack()  
  
a = Label(root, text="百战程序员", width=10, height=1, bg="black", fg="white")  
a.pack()  
  
root.mainloop()
```

运行结果：



颜色选择框

颜色选择框可以帮助我们设置背景色、前景色、画笔颜色、字体颜色等等。



【示例】颜色选择框基本用法

"""askcolor 颜色选择框的测试，改变背景色"""

```
from tkinter import *  
from tkinter.colorchooser import *  
  
root = Tk();root.geometry("400x150")  
  
def test1():  
    s1 = askcolor(color="red", title="选择背景色")  
    print(s1)  
    # s1 的值是: ((0.0, 0.0, 255.99609375), '#0000ff')  
    root.config(bg=s1[1])
```

```
Button(root, text="选择背景色", command=test1).pack()

root.mainloop()
```

运行结果：



文件对话框

文件对话框帮助我们实现可视化的操作目录、操作文件。最后，将文件、目录的信息传入到程序中。文件对话框包含如下一些常用函数：

| 函数名 | 对话框 | 说明 |
|------------------------------|-------|--------------|
| askopenfilename(**options) | 文件对话框 | 返回打开的文件名 |
| askopenfilenames(**options) | | 返回打开的多个文件名列表 |
| askopenfile(**options) | | 返回打开文件对象 |
| askopenfiles(**options) | | 返回打开的文件对象的列表 |
| askdirectory(**options) | 目录对话框 | 返回目录名 |
| asksaveasfile(**options) | 保存对话框 | 返回保存的文件对象 |
| asksaveasfilename(**options) | | 返回保存的文件名 |

命名参数 options 的常见值如下：

| 参数名 | 说明 |
|---|--------------------------|
| defaultextension | 默认后缀：.xxx 用户没有输入则自动添加 |
| filetypes=[(label1, pattern1), (labe2, pattern2)] | 文件显示过滤器 |
| initialdir | 初始目录 |
| initialfile | 初始文件 |
| parent | 父窗口，默认根窗口 |
| title | 窗口标题 |

【示例】文件对话框基本用法

```
"""文件对话框获取文件"""

from tkinter import *
from tkinter.filedialog import *

root = Tk();root.geometry("400x100")

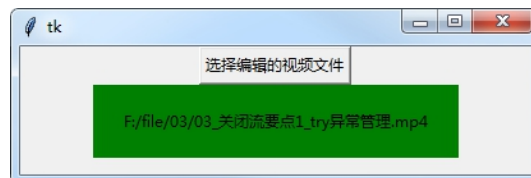
def test1():
    f = askopenfilename(title="上传文件",
                        initialdir="f:/file",filetypes=[("视频文件", ".mp4")])
    #print(f)
    show["text"]=f

Button(root, text="选择编辑的视频文件", command=test1).pack()

show = Label(root, width=40, height=3, bg="green")
show.pack()

root.mainloop()
```

运行结果：



【示例】打开指定 txt 文件，并读出文件内容到窗口

```
#coding=utf-8
#askcolor 颜色选择框的测试，改变背景色

from tkinter import *
from tkinter.filedialog import *

root = Tk();root.geometry("400x100")

def test1():
    with askopenfile(title="上传文件",
                    initialdir="d:",filetypes=[("文本文件", ".txt")]) as f:
        show["text"]=f.read()
```

```
Button(root, text="选择读取的文本文件", command=test1).pack()

show = Label(root, width=40, height=3, bg="green")
show.pack()

root.mainloop()
```

简单输入对话框

simplesdialog(简单对话框)包含如下常用函数：

| 函数名 | 说明 |
|---------------------------------|----------|
| askfloat(title, prompt, **kw) | 输入并返回浮点数 |
| askinteger(title, prompt, **kw) | 输入并返回整数 |
| askstring(title, prompt, **kw) | 输入并返回字符串 |

参数中，title 表示窗口标题；prompt 是提示信息；命名参数 kw 为各种选项：initialvalue（初始值）、minvalue（最小值）、maxvalue（最大值）。

【示例】简单对话框基本用法

```
"""简单对话框"""
from tkinter.simplesdialog import *

root = Tk();root.geometry("400x100")

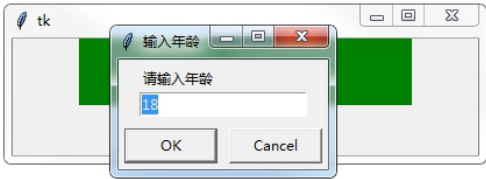
def test1():
    a = askinteger(title="输入年龄", prompt="请输入年龄",
initialvalue=18,minvalue=1,maxvalue=150)
    # askstring、askfloat 框使用方式一样
    show["text"]=a

Button(root, text="老高你多大了? 请输入", command=test1).pack()

show = Label(root, width=40, height=3, bg="green")
show.pack()

root.mainloop()
```

运行结果：



通用消息框

messagebox（通用消息框）用于和用户简单的交互，用户点击确定、取消。如下列出了messagebox 的常见函数：

| 函数名 | 说明 | 例子 |
|---|--------------------|----|
| askokcancel(title,message,**options) | OK/Cancel 对话框 | |
| askquestion(title,message,**options) | Yes/No 问题对话框 | |
| askretrycancel(title,message,**options) | Retry/Cancel 问题对话框 | |
| showerror(title,message,**options) | 错误消息对话框 | |
| showinfo(title,message,**options) | 消息框 | |
| showwarning(title,message,**options) | 警告消息框 | |

【示例】通用消息框的基本用法

"""简单对话框"""

from tkinter import *


```
from tkinter.messagebox import *

root = Tk();root.geometry("400x100")

a1 = showinfo(title="尚学堂",message="Python400 集从零开始，深入底层，\
深入算法，打好基础。还手写神经网络")
print(a1)

root.mainloop()
```

ttk 子模块控件

我们再前面学的组件是 tkinter 模块下的组件，整体风格较老较丑。为了弥补这点不足，推出了 ttk 组件。ttk 组件更加美观、功能更加强大。 新增了 LabeledScale(带标签的 Scale)、Notebook(多文档窗口)、Progressbar(进度条)、Treeview(树)等组件。

使用 ttk 组件与使用普通的 Tkinter 组件并没有多大的区别，只要导入 ttk 模块即可。

ttk 子模块的官方文档：

<https://docs.python.org/3.7/library/tkinter.ttk.html>

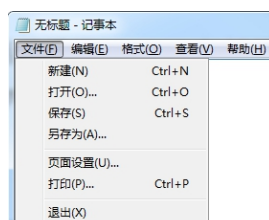
【注】此处我们不展开细讲 ttk。如果你的项目确实需要用到复杂的界面，推荐大家使用 wxpython 或者 PyQt。

菜单

GUI 程序通常都有菜单，方便用户的交互。我们一般将菜单分为两种：

1. 主菜单

主菜单通常位于 GUI 程序上方。例如：



2. 快捷菜单（上下文菜单）

通过鼠标右键单击某个组件对象而弹出的菜单，一般是与该组件相关的操作。



主菜单

主菜单一般包含：文件、编辑、帮助等，位于 GUI 窗口的上面。创建主菜单一般有如下 4 步：

1. 创建主菜单栏对象

```
menubar = tk.Menu(root)
```

2. 创建菜单，并添加到主菜单栏对象

```
file_menu = tk.Menu(menubar)
menubar.add_cascade(label="文件", menu=file_menu)
```

3. 添加菜单项到 2 步中的菜单

```
file_menu.add_command(label="打开")
file_menu.add_command(label="保存", accelerator="^p" command=mySaveFile)
file_menu.add_separator()
file_menu.add_command(label="退出")
```

4. 将主菜单栏添加到根窗口

```
root["menu"] = menubar
```

【示例】记事本软件的主菜单

```
"""开发记事本软件的菜单
"""

from tkinter.filedialog import *
from tkinter.colorchooser import *

class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)          # super() 代表的是父类的定义，而不是父类
        # 对象
        self.master = master
        self.textpad = None                # textpad 表示 Text 文本框对象
        self.pack()
        self.createWidget()

    def createWidget(self):
        # 创建主菜单栏
        menubar = Menu(root)
```

```
# 创建子菜单
menuFile = Menu(menubar)
menuEdit = Menu(menubar)
menuHelp = Menu(menubar)

# 将子菜单加入到主菜单栏
menubar.add_cascade(label="文件(F)", menu=menuFile)
menubar.add_cascade(label="编辑(E)", menu=menuEdit)
menubar.add_cascade(label="帮助(H)", menu=menuHelp)

# 添加菜单项
menuFile.add_command(label="新建", accelerator="ctrl+n",
command=self.test)
menuFile.add_command(label="打开", accelerator="ctrl+o",
command=self.test)
menuFile.add_command(label="保存",
accelerator="ctrl+s", command=self.test)
menuFile.add_separator() # 添加分割线
menuFile.add_command(label="退出",
accelerator="ctrl+q", command=self.test)

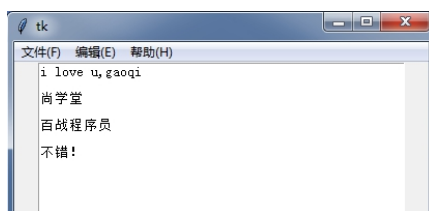
# 将主菜单栏加到根窗口
root["menu"] = menubar

#文本编辑区
self.textpad = Text(root, width=50, height=30)
self.textpad.pack()

def test(self):
    pass

if __name__ == '__main__':
    root = Tk()
    root.geometry("450x300+200+300")
    root.title("百战程序员的简易记事本")
    app = Application(master=root)
    root.mainloop()
```

运行结果：



上下文菜单

快捷菜单（上下文菜单）是通过鼠标右键单击组件而弹出的菜单，一般是和这个组件相关的操作，比如：剪切、复制、粘贴、属性等。创建快捷菜单步骤如下：

1. 创建菜单

```
menubar = tk.Menu(root)
menubar.add_command(label=" 字体" )
```

2. 绑定鼠标右键单击事件

```
def test(event):
    menubar.post(event.x_root, event.y_root) #在鼠标右键单击坐标处显示菜单
root.bind("<Button-3>", test)
```

【示例】为记事本程序增加上下文菜单

```
#coding=utf-8

from tkinter import *
from tkinter.colorchooser import *
from tkinter.filedialog import *

root = Tk();root.geometry("400x400")

def openAskColor():
    s1 = askcolor(color="red", title="选择背景色")
    root.config(bg=s1[1])

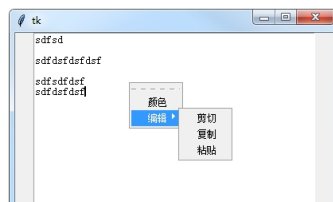
#创建快捷菜单
menubar2 = Menu(root)
menubar2.add_command(label="颜色", command=openAskColor)

menuedit = Menu(menubar2, tearoff=0)
menuedit.add_command(label="剪切")
menuedit.add_command(label="复制")
menuedit.add_command(label="粘贴")

menubar2.add_cascade(label="编辑", menu=menuedit)
```

```
def test(event):  
    #菜单在鼠标右键单击的坐标处显示  
    menubar2.post(event.x_root, event.y_root)  
  
#编辑区  
w1 = Text(root, width=50, height=30)  
w1.pack()  
  
w1.bind("<Button-3>", test)  
  
root.mainloop()
```

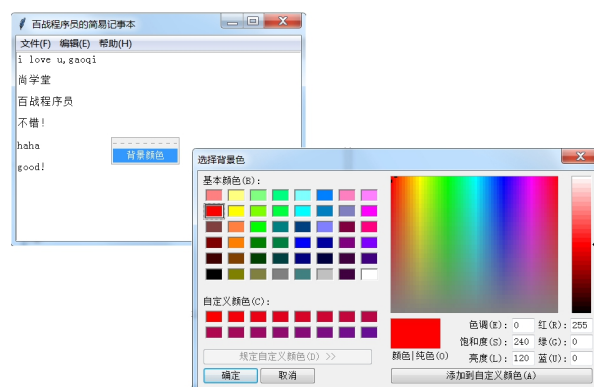
运行结果：



【项目】记事本软件开发

结合所学 GUI 知识，开发一款模仿 windows 记事本的软件。包含了基本的功能：

1. 新建文本文件
2. 保存文件
3. 修改文件内容
4. 退出
5. 各种快捷键处理
6. 修改文本区域背景色



```

"""开发一个简单的记事本。
    包含：新建、保存、修改文本内容、退出
    包含：各种快捷键的处理
    version 1.0
"""

from tkinter.filedialog import *
from tkinter.colorchooser import *

class Application(Frame):

    def __init__(self, master=None):
        super().__init__(master)          # super() 代表的是父类的定义，而不是父类
        # 对象
        self.master = master
        self.textpad = None               # textpad 表示 Text 文本框对象
        self.filename = None              # filename 表示打开文本文件的名字
        self.contextMenu = None           # contextMenu 上下文菜单对象
        self.pack()
        self.createWidget()

    def createWidget(self):
        # 创建主菜单栏
        menubar = Menu(root)

        # 创建子菜单
        menuFile = Menu(menubar)
        menuEdit = Menu(menubar)
        menuHelp = Menu(menubar)

        # 将子菜单加入到主菜单栏
        menubar.add_cascade(label="文件(F)", menu=menuFile)
        menubar.add_cascade(label="编辑(E)", menu=menuEdit)
        menubar.add_cascade(label="帮助(H)", menu=menuHelp)

```

```
# 添加菜单项
menuFile.add_command(label="新建", accelerator="ctrl+n",
command=self.newfile)
menuFile.add_command(label="打开", accelerator="ctrl+o",
command=self.openfile)
menuFile.add_command(label="保存",
accelerator="ctrl+s", command=self.savefile)
menuFile.add_separator() # 添加分割线
menuFile.add_command(label="退出",
accelerator="ctrl+q", command=self.exit)
# 将主菜单栏加到根窗口
root["menu"] = menubar

#添加快捷键事件处理
root.bind("<Control-n>", lambda event:self.newfile())
root.bind("<Control-o>", lambda event:self.openfile())
root.bind("<Control-s>", lambda event:self.savefile())
root.bind("<Control-q>", lambda event:self.exit())

#文本编辑区
self.textpad = Text(root, width=50, height=30)
self.textpad.pack()

# 创建上下菜单
self.contextMenu = Menu(root)
self.contextMenu.add_command(label="背景颜色",
command=self.openAskColor)

#为右键绑定事件
root.bind("<Button-3>", self.createContextMenu)

def newfile(self):
    self.textpad.delete('1.0', 'end') # 把 Text 控件中的内容清空

    self.filename = asksaveasfilename(title='另存为', initialfile='未命名.txt',
                                     filetypes=[("文本文档", "*.txt")],
                                     defaultextension='.txt')
    print(self.filename)

    self.savefile()

def openfile(self):
```

```
self.textpad.delete('1.0', 'end') # 先把 Text 控件中的内容清空
with askopenfile(title="打开文件") as f:
    self.textpad.insert(INSERT, f.read())
    self.filename = f.name
    print(f.name)

def savefile(self):
    with open(self.filename, "w") as f:
        c = self.textpad.get(1.0, END)
        f.write(c)

def exit(self):
    root.quit()

def openAskColor(self):
    s1 = askcolor(color="red", title="选择背景色")
    self.textpad.config(bg=s1[1])

def createContextMenu(self, event):
    # 菜单在鼠标右键单击的坐标处显示
    self.contextMenu.post(event.x_root, event.y_root)

if __name__ == '__main__':
    root = Tk()
    root.geometry("450x300+200+300")
    root.title("百战程序员的简易记事本")
    app = Application(master=root)
    root.mainloop()
```

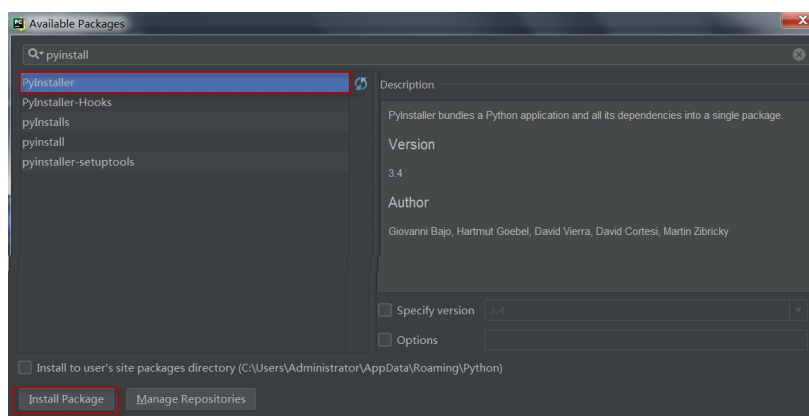
运行结果：

【项目】将 python 程序打包成 exe 文件

我们可以使用 pyinstaller 模块实现将 python 项目打包成 exe 文件。操作步骤如下：

1. 安装 pyinstaller 模块

在 pycharm 中操作：file-->setting-->Project: xxx -->Project interpreter, 再点击+即可。




2. 在 pycharm 的 Terminal 终端输入如下命令：

```
pyinstaller -F xxxx.py
```

【注】相关参数如下：

- icon=图标路径 (pyinstaller -F --icon=my.ico XXXX.py)
- F 打包成一个 exe 文件
- w 使用窗口，无控制台
- c 使用控制台，无窗口
- D 创建一个目录，里面包含 exe 以及其他一些依赖性文件

3. 在项目的 dist 目录下可以看到生成了 exe 文件，直接在 windows 系统中使用即可。

| 名称 | 类型 | 大小 |
|--|------|----------|
|  notebook.exe | 应用程序 | 7,757 KB |

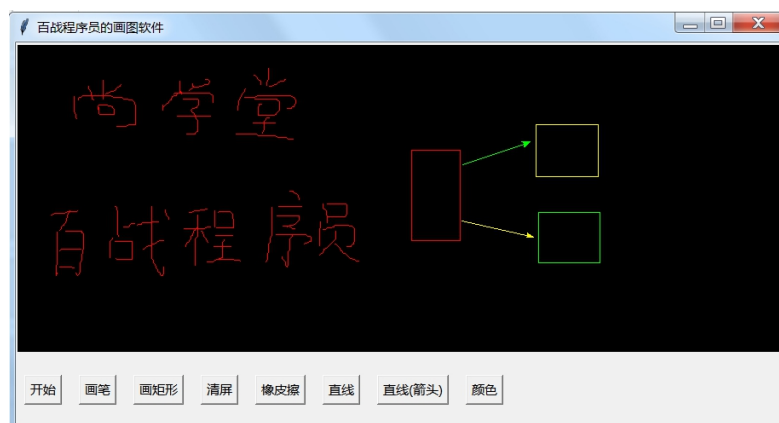
【注】exe 文件本质是将 python 解释器和程序打包到了一起，这样我们执行程序时就不用管 windows 系统是不是有 python 解释器。

【注】pyinstaller 目前只支持 python3.6。如果你是 python3.7 会执行失败。

【项目】画图软件开发

开发一款简单的画图软件，包含如下功能：

1. 画笔
2. 矩形/椭圆绘制
3. 清屏
4. 橡皮擦
5. 直线/带箭头的直线
6. 修改画笔颜色、背景颜色



```

"""开发画图软件的菜单
"""

from tkinter.filedialog import *
from tkinter.colorchooser import *

#窗口的宽度和高度
win_width=900
win_height=450

class Application(Frame):

    def __init__(self, master=None, bgcolor="#000000"):
        super().__init__(master)          # super() 代表的是父类的定义，而不是父类
        # 对象
        self.master = master
        self.bgcolor=bgcolor
        self.x = 0
        self.y = 0
        self.fgcolor = "#ff0000"
        self.lastDraw = 0                  # 表示最后绘制的图形的 id
        self.startDrawFlag = False
        self.pack()
        self.createWidget()

    def createWidget(self):
        # 创建绘图区
        self.drawpad =
Canvas(root,width=win_width,height=win_height*0.9,bg=self.bgcolor)
        self.drawpad.pack()

```

```
#创建按钮
btn_start = Button(root, text="开始", name="start")
btn_start.pack(side="left", padx="10")
btn_pen = Button(root, text="画笔", name="pen")
btn_pen.pack(side="left", padx="10")
btn_rect = Button(root, text="矩形", name="rect")
btn_rect.pack(side="left", padx="10")
btn_clear = Button(root, text="清屏", name="clear")
btn_clear.pack(side="left", padx="10")
btn_erasor = Button(root, text="橡皮擦", name="erasor")
btn_erasor.pack(side="left", padx="10")
btn_line = Button(root, text="直线", name="line")
btn_line.pack(side="left", padx="10")
btn_lineArrow = Button(root, text="箭头直线", name="lineArrow")
btn_lineArrow.pack(side="left", padx="10")
btn_color = Button(root, text="颜色", name="color")
btn_color.pack(side="left", padx="10")

#事件处理
btn_pen.bind_class("Button", "<1>", self.eventManager)
self.drawpad.bind("<ButtonRelease-1>", self.stopDraw)

#增加颜色切换的快捷键
root.bind("<KeyPress-r>", self.kuaijiejian)
root.bind("<KeyPress-g>", self.kuaijiejian)
root.bind("<KeyPress-y>", self.kuaijiejian)

def eventManager(self, event):
    name = event.widget.winfo_name()
    print(name)
    if name=="line":
        self.drawpad.bind("<B1-Motion>", self.myline)
    elif name=="lineArrow":
        self.drawpad.bind("<B1-Motion>", self.mylineArrow)
    elif name=="rect":
        self.drawpad.bind("<B1-Motion>", self.myRect)
    elif name=="pen":
        self.drawpad.bind("<B1-Motion>", self.myPen)
    elif name=="erasor":
        self.drawpad.bind("<B1-Motion>", self.myErasor)
    elif name=="clear":
        self.drawpad.delete("all")
    elif name=="color":
        c = askcolor(color=self.fgcolor, title="选择画笔颜色")
```

```
        #[(255, 0, 0), "#ff0000"]
        self.fgcolor = c[1]
    def stopDraw(self, event):
        self.startDrawFlag = False
        self.lastDraw = 0

    def startDraw(self, event):
        self.drawpad.delete(self.lastDraw)

        if not self.startDrawFlag:
            self.startDrawFlag = True
            self.x = event.x
            self.y = event.y

    def myline(self, event):
        self.startDraw(event)
        self.lastDraw =
self.drawpad.create_line(self.x, self.y, event.x, event.y, fill=self.fgcolor)

    def mylineArrow(self, event):
        self.startDraw(event)
        self.lastDraw =
self.drawpad.create_line(self.x, self.y, event.x, event.y, arrow=LAST, fill=self.fgcolor)

    def myRect(self, event):
        self.startDraw(event)
        self.lastDraw =
self.drawpad.create_rectangle(self.x, self.y, event.x, event.y, outline=self.fgcolor)

    def myPen(self, event):
        self.startDraw(event)

self.drawpad.create_line(self.x, self.y, event.x, event.y, fill=self.fgcolor)
        self.x = event.x
        self.y = event.y

    def myEraser(self, event):
        self.startDraw(event)

self.drawpad.create_rectangle(event.x-4, event.y-4, event.x+4, event.y+4, fill=self.bgcolor)
        self.x = event.x
```

```
        self.y = event.y

    def kuaijiejian(self, event):
        if event.char == "r":
            self.fgcolor = "#ff0000"
        elif event.char == "g":
            self.fgcolor = "#00ff00"
        elif event.char == "y":
            self.fgcolor = "#ffff00"

if __name__ == '__main__':
    root = Tk()
    root.geometry(str(win_width)+"x"+str(win_height)+"+200+300")
    root.title("百战程序员的画图软件")
    app = Application(master=root)
    root.mainloop()
```