Table of Contents

前言：GUI与tkinter	1.1
认识tkinter	1.2
第一个tkinter程序	1.3
初识组件之标签	1.4
组件介绍与按钮	1.5
Tkinter的三种布局	1.6
事件与绑定	1.7
输入框以及登录程序	1.8
菜单与选择按钮	1.9
对话框和消息框	1.10
常用控件的介绍	1.11
Canvas与图形绘制	1.12
窗口美化效果	1.13
ttk介绍以及总结	1.14
桌面应用项目：小球弹弹弹	1.15

GUI编程

> - 如果同学们有过 C++ 或者 C#或者 VB 等语言的编程经验，甚至是咱们自己国家的易语言编程，它们都有一些可以通过拖拽的方式来编辑界面的工具。
> - 但是很可惜，在 Python 里面这种工具并不多，这也是和 Python 是一个脚本语言有关。
> - 要说 tkinter，先说 tk，它原本是 Tcl 编程语言的界面库， 后来开发出了 Python 接口，Tkinter 在 Python2 和 Python3 中的变化较大，在 Python2 中叫做 Tkinter，在 Python3 中叫做 tkinter，而且在导入类库的时候也有些许 的变化，请读者稍加注意。

tkinter的定位

> - 掌握一门 Python 的界 面编程也很有意思。有时候我们需要一些界面程序，但是又不想用那些庞大 而繁杂的类库的时候，tkinter 的优势就显现出来了，那就是简洁简单。
> - 我感觉 tkinter 是一个还不错的界面库，但是我感觉 Python 并不是特别擅长编写界面应用，而且我感觉比如 wxPython 或者 pyQt 等更胜一筹，但是不管怎么样，我们可以通过Python自带的tkinter，进入GUI编程的世界。

传智播客 大猫

Tkinter 简介

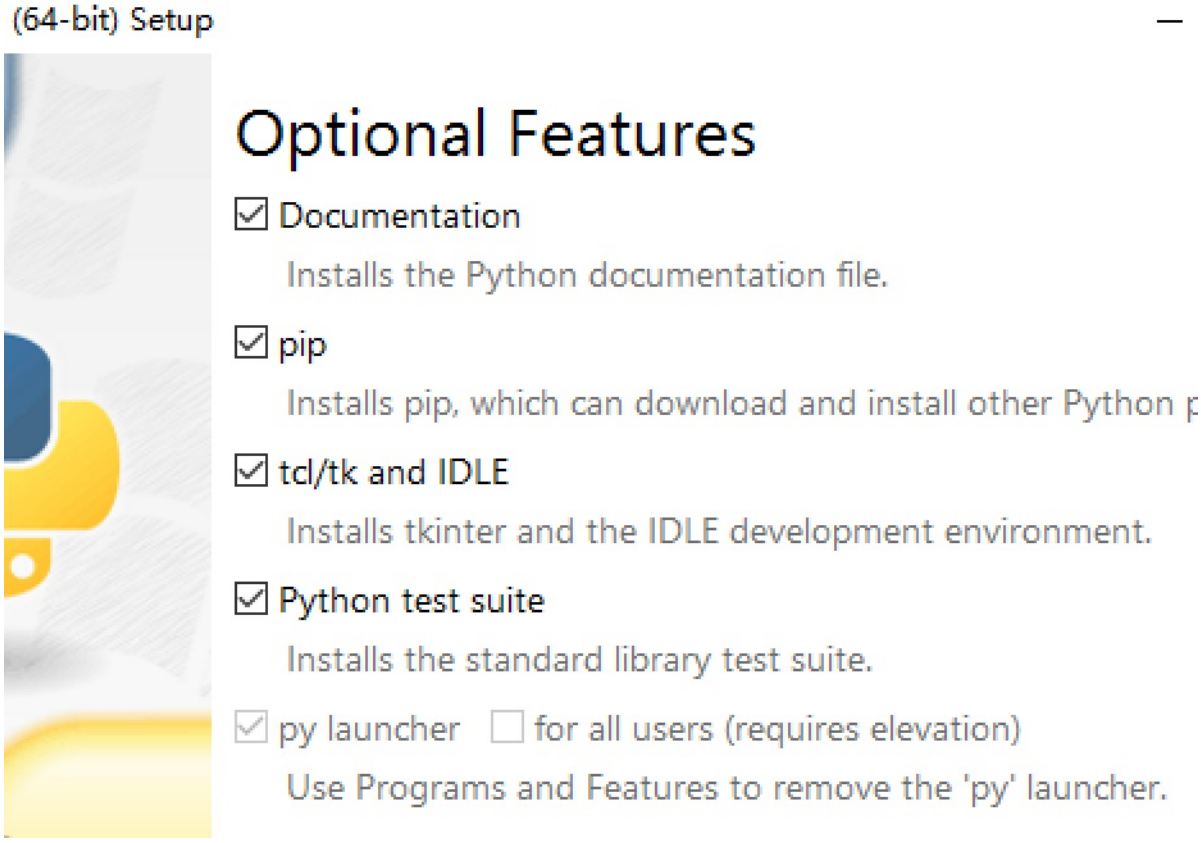
1. 提到 Tkinter,不得不提一下 GUI。GUI 即 graphical user interface,即图形界面用户接口，是相对于我们学习绝大多数编程语言所接触到的“控制台”的那个黑乎乎的命令行来说，区分的图形界面编程模式。
2. Tkinter 是 Tk 的 Python 接口,而 Tk 原本是 Tcl 的 GUI 库,Tcl 是一门语言,具体我就不多说了。

Tkinter 的优点和缺点

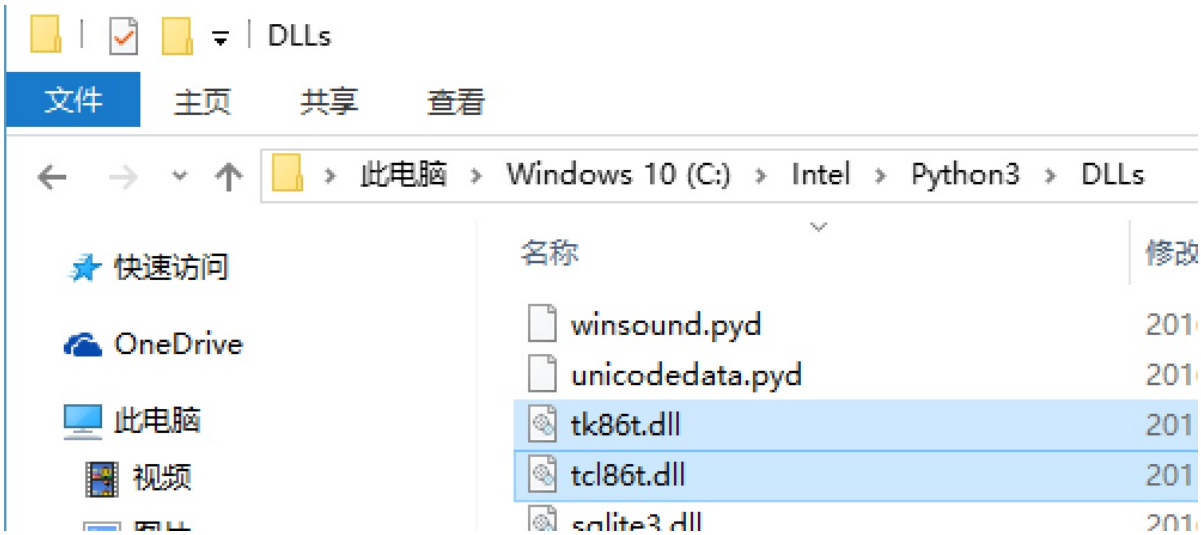
1. 简单易学，相对于其他的比如 wxPython，PyQt 等等,学起来会非常轻松。
2. 可以使用少量的代码产生功能强大的 GUI 界面，这一点 是相对于 C++等语言来说的，这也是作为一门脚本语言令人自豪的地方。
3. 跨平台，基于Python优良的平台通用性，同样的代码可以在不同的平台下编译执行，当然这一点也可能会涉及到“跨平台同时也会丧失平台优势”等问题。
4. 内置与 Python 里面的，不需要我们单独去安装, 方便了很多，而且Tkinter也是一个 Python 里面的标准。
5. 缺点就是功能过于简单，对于一些很大型的应用，Tkinter 提供的功能还是太少了。但是 Tkinter会不断进步，而且他是Python内置的GUI库，有点嫡系的感觉，我们有理由相信它会越来越棒！
6. 还有一个缺点是相对于 C++来说的,它的效率问题还是不 算太高,对于一些对画面要求很高的，可能效果会有影响。

认识 tkinter

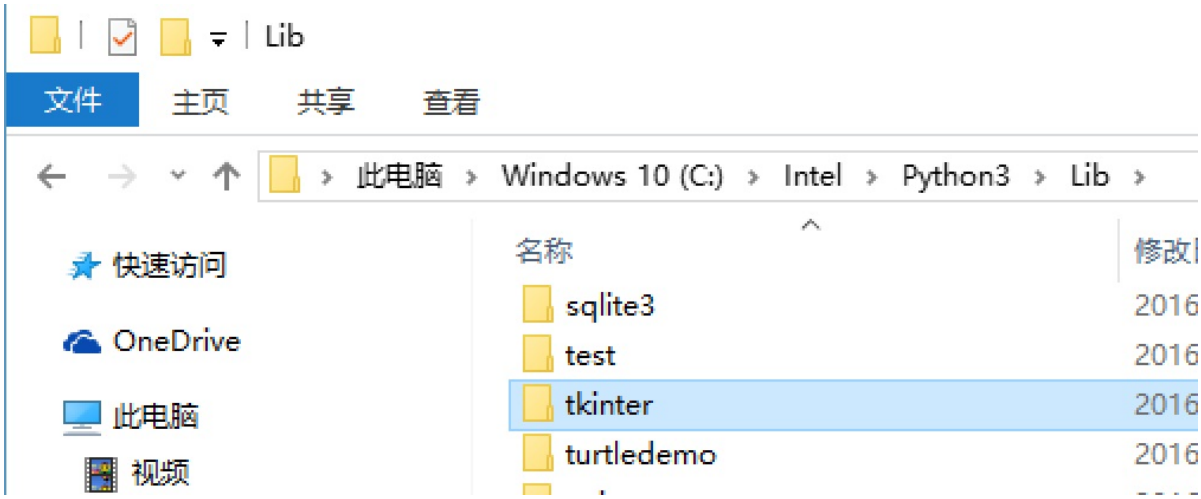
1. Tkinter 读的时候,T 和 K 分别发音,inter 单独发音, 即读音如下“tea-key-inter”。该模块在我们安装 python 的时候，就可以选择一起安装。



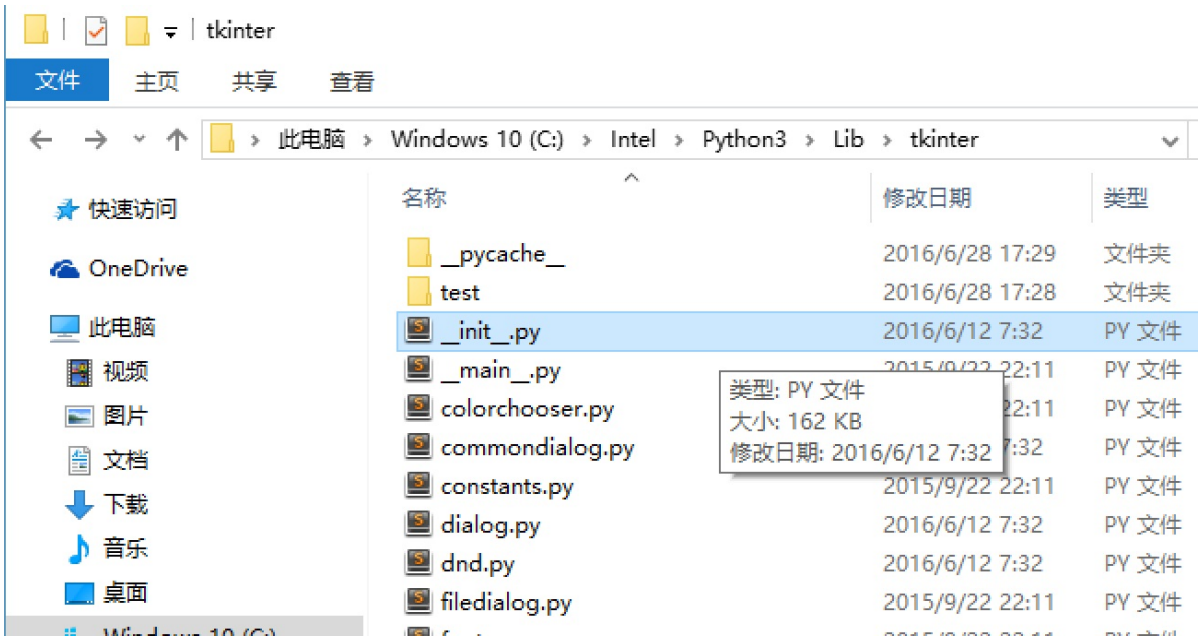
2. 比如说，我的Python3安装目录是: `C:\Intel\Python3` ,则进入该目录，可以发现有 不少的 Tkinter 的影子。比如在 DLLs 子目录下，可以看到如下动态链接库文件，截图如下(注意 tk86t.dll 是与该模块直接相关的):



3. 在 Lib 子目录下,我们可以看到一个名为 tkinter 的子 文件夹,截图如下:



4. 进入该文件夹,可以看到里面的全部文件,大多数都 是.py 文件,还有两个文件夹,第一个是和缓存有关的,第 二个是和测试有关的,剩下的都是 py 文件,下面是截图:



5. 这些文件分别负责不同的功能,我们下面进入这个 `__init__.py`,它有 163K 大小,且代码有 3857 行。**init**.py 里有我们平时所频繁使用的绝大多数类,如果以后大家有 什么新问题,可以通过阅读该文件来得到答案,因为源代码都给我们了,掌握了源代码就是掌握了第一手资料,解决问题会更加方便一些。

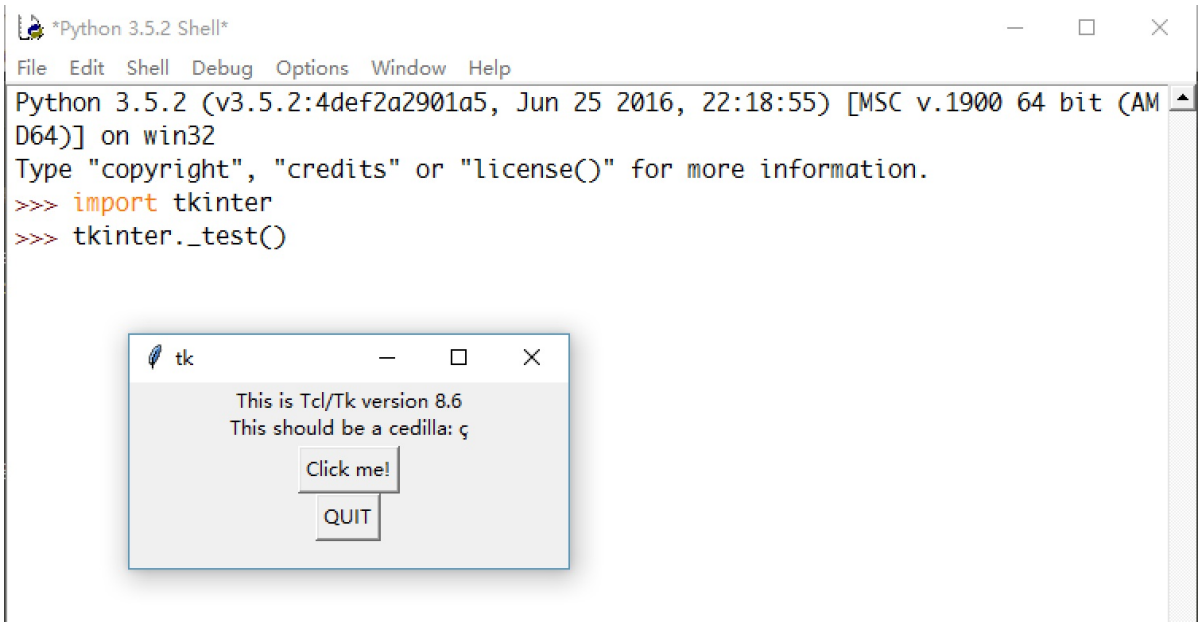
6. 其实在我们的安装模块中,还有一些其他的东西有着 tkinter 的影子,我就不一一介绍了。

开始前的测试

1. 在我们编程之前,先开始一个小测试把,它只需要两行 代码。

2. 我们在 IDLE 中书写两行,第一行是导入 tkinter 这个模 块,注意大小写,因为 Python 是区分大小写的,第二行是 使用一个测试函数。

3. 代码和运行效果截图:



4. 说明:如果点击 `Click me!` 按钮,则按钮两边会不断的增加中括号，而点击下面的 `QUIT` 按钮,则退出程序。

5. 现在，让我们进入tkinter的世界吧！

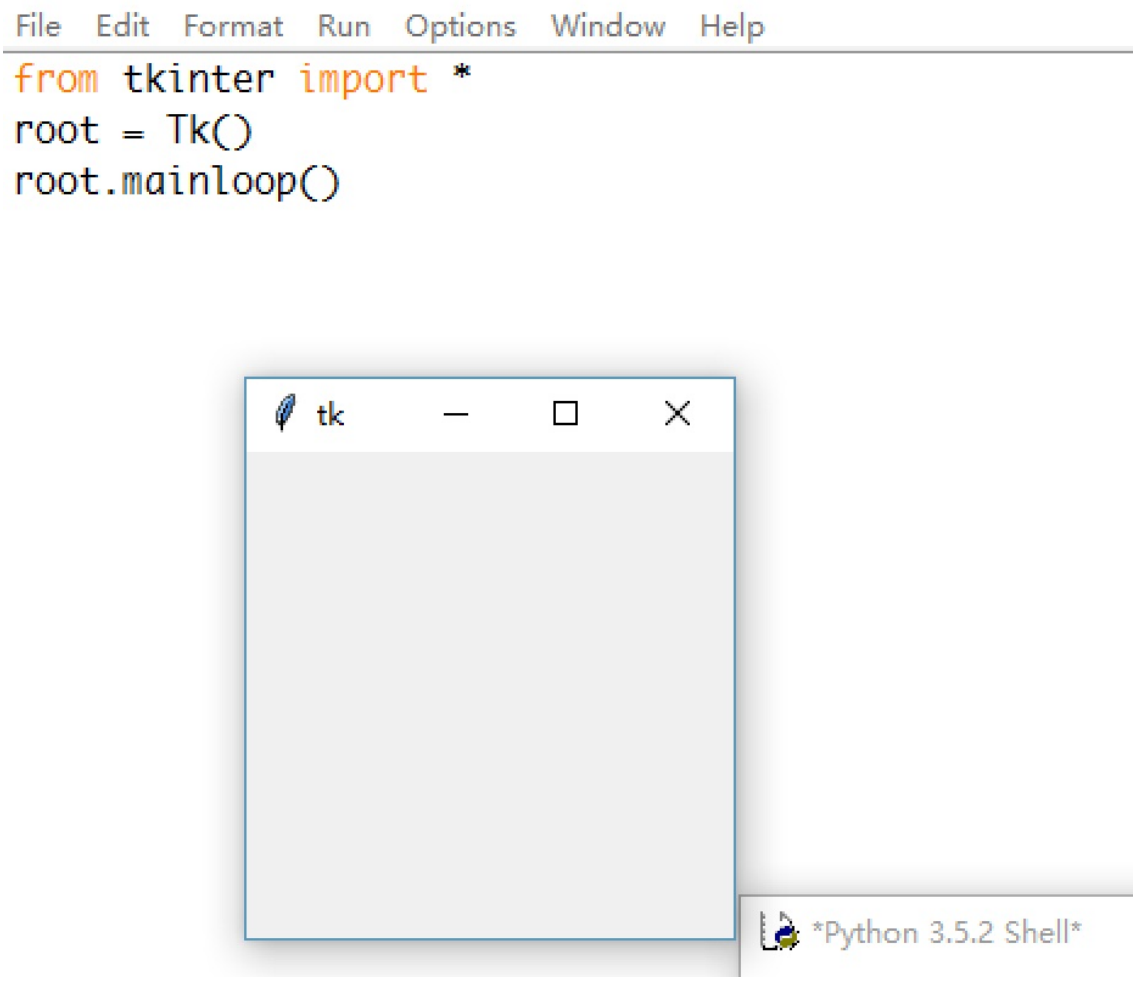
开始第一个 tkinter 程序

第一个程序

1. 我们新建一个无格式的文本文件，通常是.txt 文件，然后修改名称为 demo1.py，然后用文本编辑器打开，然后写三行代码。
2. 这三行代码如下：

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

i. 这是执行后的结果：



代码解读

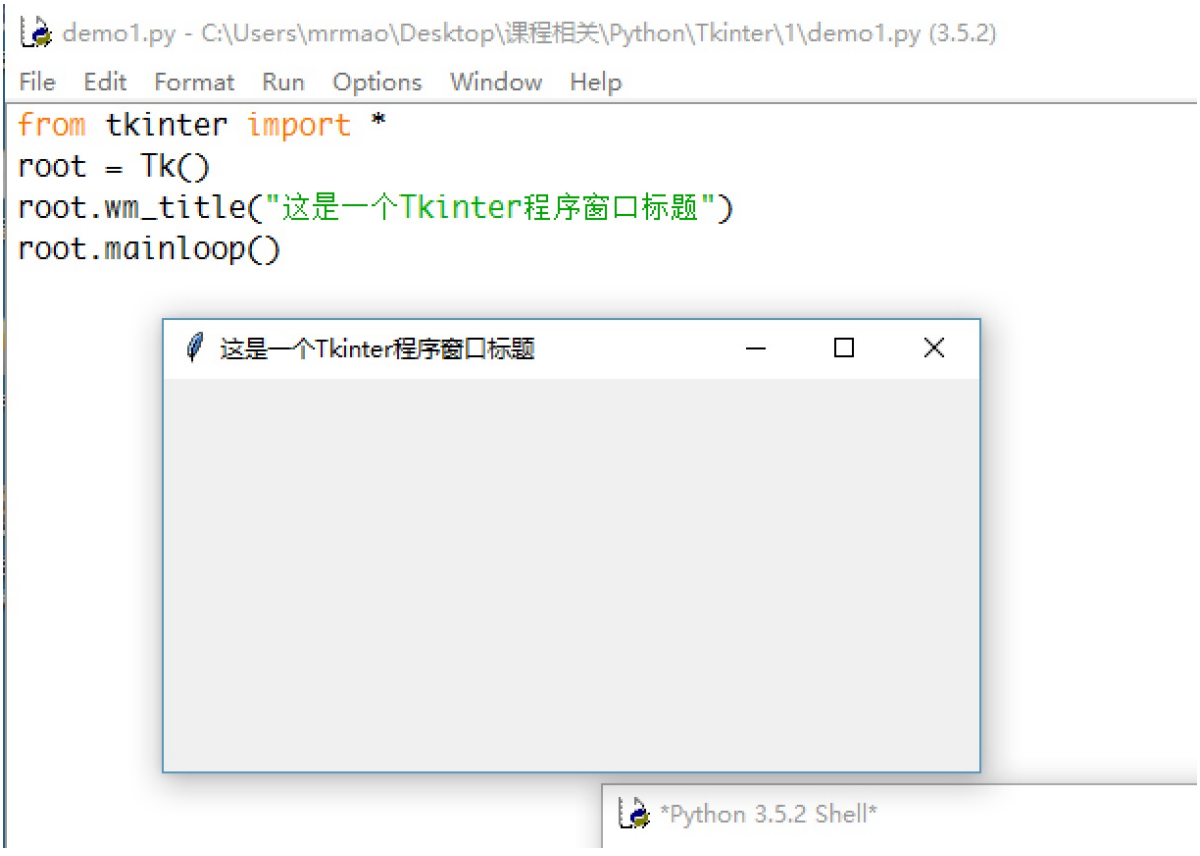
1. 代码的第一行是引入了该模块，即 `from tkinter import *` 所做的工作。

2. 第二行是对 Tk 这个类进行实例化，即 root 是它的一个实例，是代码 `root = Tk()` 的作用，root即是整个窗口程序对象。
3. 第三行进行事件循环，即 `root.mainloop()` 的作用，会让窗口循环接收下一个事件。

Tk 类解读

1. 官方对于该类的描述是 “代表绝大多数应用程序主窗口的高层空间，它会和一个 Tcl 解释器相关联”，它继承自两个类，一个是 Misc，另一个是 Wm，我们这里使用的 `mianloop` 方法，就是从 Misc 里面继承来的。
2. 对于 Tk，我们这么理解，就把它当做一个 tkinter 程序 的所有窗口的老大即可，即所有的部件都需要先创建它。
3. 我们可以通过阅读这两个类的源代码 来得到一些信息，比如在 Wm 的源代码发现了一个方法，可以修改这个窗口的标题，即 `wm_title` 方法，下面我们用这个方法修改这个窗口的标题，或者直接用 `title` 也是可以的。
4. 代码如下:

```
from tkinter import *
root = Tk()
root.wm_title("这是一个Tkinter程序窗口标题")
#root.title("这是一个Tkinter程序窗口标题") #这样也是可以的
root.mainloop()
```



认识新组件---标签

组件

1. 如果大家有过 Windows GUI 编程经验，那么对于 `控件` 一次 一定不会陌生，其实组件和控件是一回事，在这里我们不加以区分。
2. 组件的英文表述是 `widget`，我们常见的按钮，文本 框等等都可以当做组件。
3. 我们可以为一个窗口增加若干组件来丰富该窗口的功能， 我们往往需要指定这个组件的大小和位置，我们还需要通过编程的方式，来指定该组件能完成什么功能。

Label 类

标签

1. 所谓 `标签`，就是贴在物品前面的一个简短的说明， 它用于说明一些文字信息。
2. 标签可以说是最简单的窗口组件了，它不需要执行任何功能，只是用来显示信息。
3. 下面是代码示例:

```
from tkinter import *

root = Tk()
root.wm_title("这是一个Tkinter程序窗口标题")

wl= Label(root, text = "欢迎来到传智公开课")
wl.pack()

root.mainloop()
```

   i. 演示效果

demo1.py - C:\Users\mrmao\Desktop\课程相关\Python\Tkinter\1\demo1.py (3.5.2)

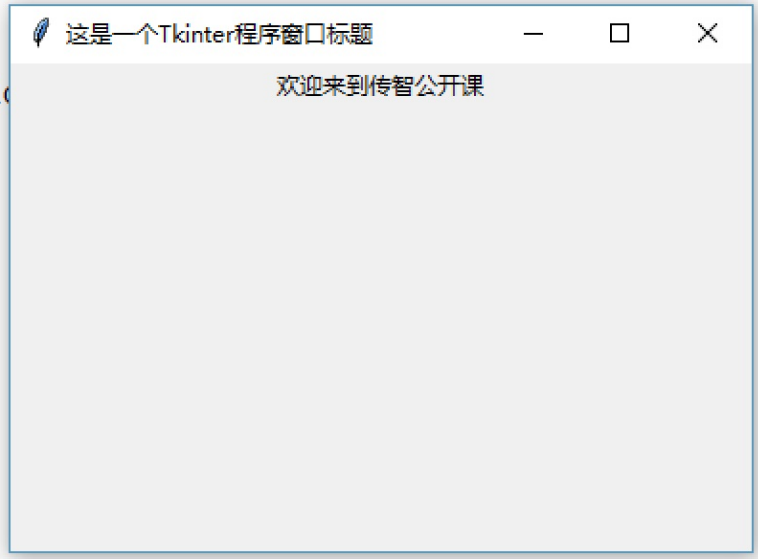
File Edit Format Run Options Window Help

```
from tkinter import *

root = Tk()
root.wm_title("这是一个Tkinter程序窗口标题")

wl = Label(root, text = "欢迎来到传智公开课")
wl.pack()

root.mainl
```



代码分析

1. 这里的 `Label` 是一个类，可以在 `__init__.py` 文件里查看 相应的源代码。
2. `wl` 是一个 `Label` 的实例，它有一个 `text 属性`，用来指定 它的文本内容。
3. 该类的源代码也非常简单，它的源代码截图:

```
2585
2586  class Label(Widget):
2587      """Label widget which can display text and bitmaps."""
2588      def __init__(self, master=None, cnf={}, **kw):
2589          """Construct a label widget with the parent MASTER.
2590
2591          STANDARD OPTIONS
2592
2593              activebackground, activeforeground, anchor,
2594              background, bitmap, borderwidth, cursor,
2595              disabledforeground, font, foreground,
2596              highlightbackground, highlightcolor,
2597              highlightthickness, image, justify,
2598              padx, pady, relief, takefocus, text,
2599              textvariable, underline, wraplength
2600
2601          WIDGET-SPECIFIC OPTIONS
2602
2603              height, state, width
2604
2605          """
2606          Widget.__init__(self, master, 'label', cnf, kw)
2607
```

4. 大家可以看到它的标准属性，比如有 `background`， `font`， `bitmap`， `padx`， `relief` 等等， 还有 `underline` 等等。

5. 该类有个 pack 方法，没错，这个 pack 方法我们后面会 讲，大家可以理解为它的作用就是找个合适的位置进行放置即可，即 pack() 之后就选定位置放上去了。

6. 这里的初始化的时候，需要先指定 root，是说 wl 这个组 件是在 root 这个窗口的，不是属于别的窗口的，以后我们 有了多窗口的应用程序，不会混淆。

Widget 类

1. 我们上面发现，Label 类是继承自 Widget 类的，并且自 己只有一个方法，就是用来初始化自身的，那么要想彻底 理解 Label，就必须彻底理解 Widget。

2. 我们找到了 Widget 的代码，发现它更加短小，甚至连个 方法都没有，下面是 Widget 的截图:

```
2152
2153  class Widget(BaseWidget, Pack, Place, Grid):
2154      """Internal class.
2155
2156      Base class for a widget which can be positioned with the geometry managers
2157      Pack, Place or Grid."""
2158      pass
2159
```

3. 通过这个注释，我们可以理解如下:它是一个可以指定 位置并且可以使用 `pack`， `place` 和 `grid` 来布局管理的窗口 组件。

4. 然后我们找到 `BaseWidget`，发现它的代码也很短小，代 码截图如下:

```
2098
2099  class BaseWidget(Misc):
2100      """Internal class."""
2101      def _setup(self, master, cnf): ...
2126      def __init__(self, master, widgetName, cnf={}, kw={}, extra=()): ...
2142      def destroy(self): ...
2149      def _do(self, name, args=()): ...
2152
```

5. 该类继承自 `Misc`，包含了四个函数，函数的功能通过名 字就可以知道了。至于Misc 类，我们后面会讲解的。

***Misc*其实是英文 `Miscellaneous` 的前四个字母，杂项、混合体、大杂烩的意思。在软件里经常可以看到与*misc*相关的文件或函数名，使用*misc*来命名主要是表示该文件目前还没归类好，不知道将它归到哪个方面或者放置在哪个地方比较好，所以暂时用*misc*。**

总结

1. 我们这一节学习了 `Label` 这个组件，它的创建使用 `Label` 这个类，实例化的时候首先要指定它的父窗口，然后就是 用字典的方式设置自己的一些特征，这些特征上面都列出 了，但是没有实例演示。
2. 然后使用 `pack()` 方法布局上去，关于布局，我们后面会详 细讲，这里我们统一使用 `pack()` 就可以了。

多标签的应用程序

1. 如果我们想要多个标签，该怎么办呢?
2. 很简单，只需要声明多个标签的实例，然后分别 pack 到 窗口上即可。
3. 下面是使用了三个标签的代码截图:

```
from tkinter import *

root = Tk()
root.wm_title("这是一个Tkinter程序窗口标题")

w1 = Label(root, text = "欢迎来到传智公开课！", background = "white")
w2 = Label(root, text = "好好学习，天天向上！", background = "green")
w3 = Label(root, text = "轻松愉快，就选Python！", background = "red")

w1.pack()
w2.pack()
w3.pack()

root.mainloop()
```

demo1.py - C:\Users\mrmao\Desktop\课程相关\Python\Tkinter\1\demo1.py (3.5.2)

File Edit Format Run Options Window Help

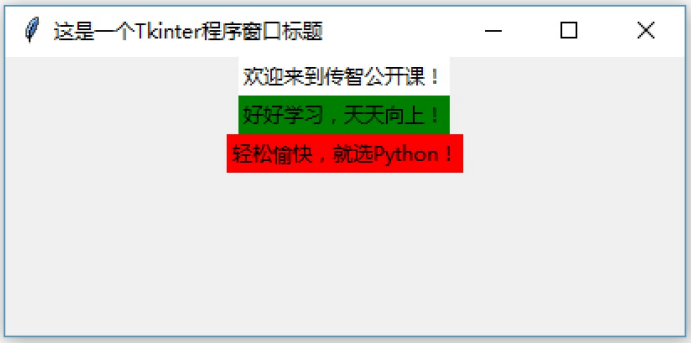
```
from tkinter import *

root = Tk()
root.wm_title("这是一个Tkinter程序窗口标题")

w1 = Label(root, text = "欢迎来到传智公开课！", background = "white")
w2 = Label(root, text = "好好学习，天天向上！", background = "green")
w3 = Label(root, text = "轻松愉快，就选Python！", background = "red")

w1.pack()
w2.pack()
w3.pack()

root.mainloop()
```



Python 3.5.2 Shell

组件讲解以及按钮

tkinter 的核心组件

1. 在 Python 的 tkinter 中，有 21 个核心组件，它们提供 了最基本的功能，虽然简单，因为使用频率较高，因此特 别重要。
2. 这 21 个核心组件是 : Toplevel、Label、Button、 Canvas、Checkbutton、Entry、Frame、LabelFrame、 Listbox、Menu、Menubutton、Message、OptionMenu、PaneWindow 、 Radiobutton 、 Scale 、Scrollbar 、Spinbox 、Text、Bitmap、Image。

组件的使用

1. 各个组件都有相应的类，我们可以通过面向对象的方式 去使用它们。
2. 这些组件的使用也很相似，在实例化这些组件的时候， 第一个参数都是父窗口或者父组件，后面跟着的就是该组 件的一些属性，比如上面我们学到的 `Label` 的 `text` 属性和 `background` 属性。
3. 多个组件的位置控制方式也很相似，我们可以用 `pack` 方法来进行简单的布局，具体的个例我们后面再说。
4. 组件也会有些方法是共用的，比如 `configure` 方法来设置属性等等。

按钮

1. 按钮也是非常重要的组件， 按钮的重要性在于它可以执行相应 的功能。
2. 按钮的英文表述是 `button` ，它随处可见，当我们单 击的时候，它可以执行相应的功能。
3. 按钮在 tkinter 中有一个类专门负责它，叫做 `Button` ，该类也非常简短。

```
2187
2188 class Button(Widget):
2189     """Button widget."""
2190     def __init__(self, master=None, cnf={}, **kw):
2210
2211     def flash(self):
2222
2223     def invoke(self):
2232
```

按钮与功能的绑定

1. 上文说到，按钮可以执行相应的功能，这里的功能我们 可以理解为一个函数，或者这些功能通过相应的函数去实 现。
2. 绑定方式通常有如下几种:第一种，在按钮组件被声明 的时候用 `command` 属性声明， `command` 属性接受一个函数名， 注意函数名不要加双引号。第二种，使用 `bind` 方法，该方 法是 `Misc` 这个类的一个方法，下面我们仔细讲解。

1. 第一种方法绑定事件

我们要完成的功能是我们按下这个按钮的时候，就会在 窗口上增加一个 Label，它显示“我爱 python”。

代码示例:

```
from tkinter import *

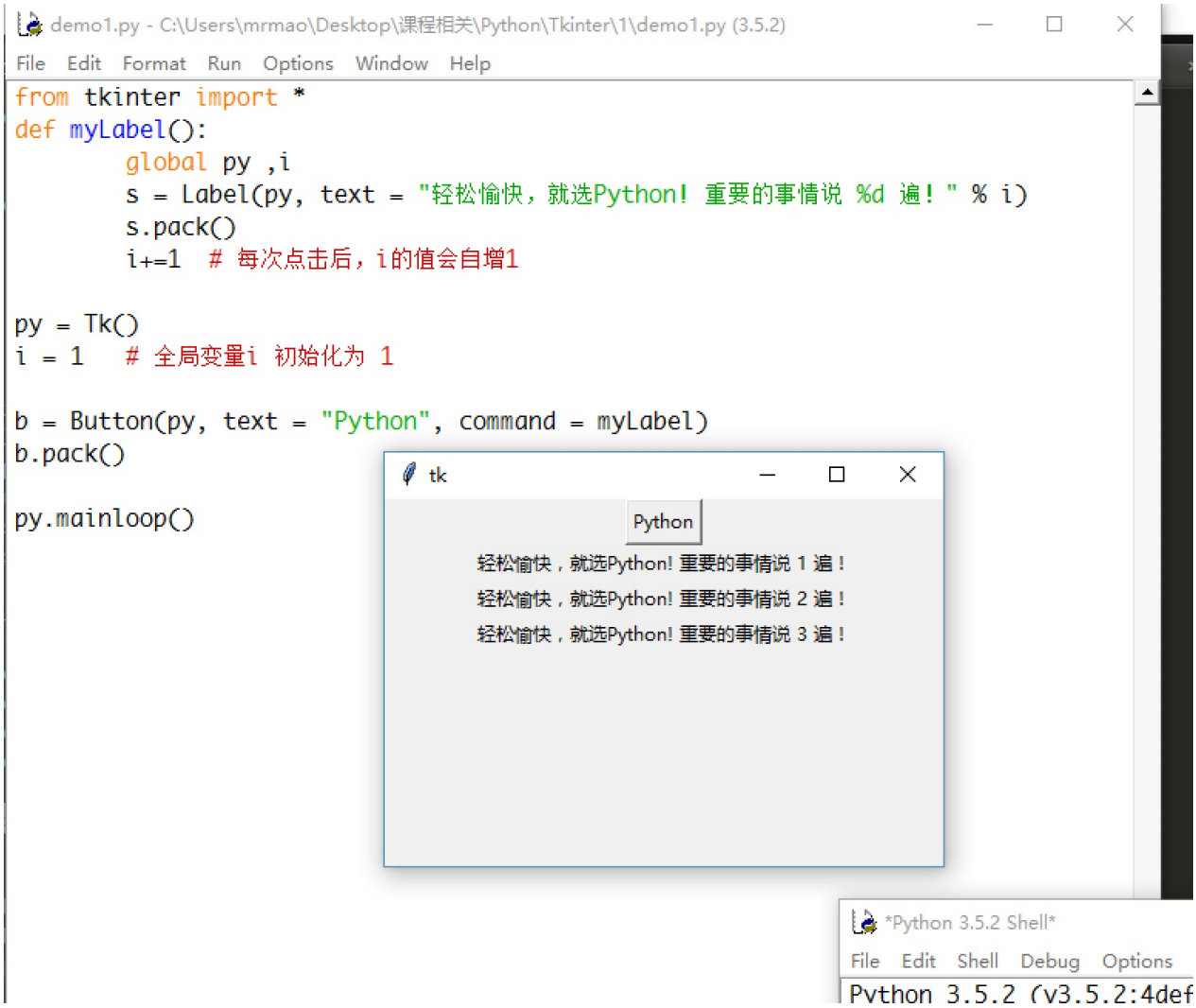
def myLabel():
    global py, i
    s = Label(py, text = "轻松愉快，就选Python！ 重要的事情说 %d 遍！ " % i)
    s.pack()
    i+=1  # 每次点击后，i的值会自增1

py = Tk()
i = 1   # 全局变量i 初始化为 1

b = Button(py, text = "Python", command = myLabel)
b.pack()

py.mainloop()
```

运行结果： （一开始是只有一个按钮的，点击一下，就会在这个窗口上多一个标签，下面是我点击了三次之后的截图):



代码解读

1. 其实很简单，这里只需要注意 `command` 属性后面不要加任何的标点符号。
2. 这里的 myLabel 函数中，使用了全局的 `py` 和 `i` 变量，需要注意。

2. 第二种方式绑定事件

下面使用第二种方式来绑定事件，这个事件完成同样的功能。

代码示例:

```
from tkinter import *

def myLabel(event): # 这里多了一个event参数用来接收
    global py, i
    s = Label(py, text = "轻松愉快，就选Python！ 重要的事情说 %d 遍！ " % i)
    s.pack()
    i+=1

py = Tk()
i = 1

b = Button(py, text = "Python") # command = myLabel 木有了
b.bind("<Button-1>", myLabel)      # 多了这一句

b.pack()
py.mainloop()
```

运行结果：（和第一种方法的结果一样）

代码解读：

1. bind这个方法是在 Misc 类中的，可以接受三个参数，但是本例中我们只传递了两个参数。
2. 第一个参数可能对刚使用它的人来说有点复杂，常见的鼠标左键单击如下: `<Button-1>` （也就是上面的代码用到的），或者 `<Control-Button-1>` 等。
3. 第二个参数可以是一个函数名，记住，不要加任何的标 点符号，否则运行时会报错的。
4. 使用 bind 函数的时候，第二个参数是一个函数名，该函数必须接受一个参数，即表示该事件。 这个参数通常用 event 来表示，如果我们调用的函数不接 受任何参数，则会报错如下： `TypeError: myLabel() takes no arguments (1 given)`

---

bind的第一个参数是用字符串包含的事件类型，它采用的描述方式是: `<MODIFIER-MODIFIER-TYPE-DETAIL>`

1. 这里的 `MODIFIER` 即键盘或者鼠标修饰符，它的全部取值如下: Control， Mod2， M2， Shift， Mod3， M3， Lock， Mod4， M4， Button1， B1， Mod5， M5 Button2， B2， Meta， M， Button3，B3， Alt， Button4， B4， Double， Button5， B5 Triple ， Mod1， M1

2. `TYPE` 表示类型，它的全部取值如下: Activate， Enter， Map， ButtonPress， Button， Expose， Motion， ButtonRelease， FocusIn， MouseWheel， Circulate， FocusOut， Property， Colormap， Gravity Reparent， Configure，

KeyPress， Key， Unmap， Deactivate， KeyRelease Visibility， Destroy，Leave

3. `DETAIL` 表示细节，其实也就是对第二个参数的一些辅助说明。

设置属性

下面我们可以设置一些属性，这些东西随着大量的代码示例，大家会接触不少的。

比如我们可以设置背景色，这里直接设置属性，代码截 图:

```
from tkinter import *

py = Tk()

b1 = Button(py, text = "传智播客")
b1["width"] = 20
b1["height"] = 4
b1.pack()

b2 = Button(py, text = "Python学院")
b2["width"] = 20
b2["background"] = "white"
b2.pack()

py.mainloop()
```

运行结果:

demo1.py - C:\Users\mrmao\Desktop\课程相关\Python\Tkinter\1\demo1.py (3.5.2)

File Edit Format Run Options Window Help

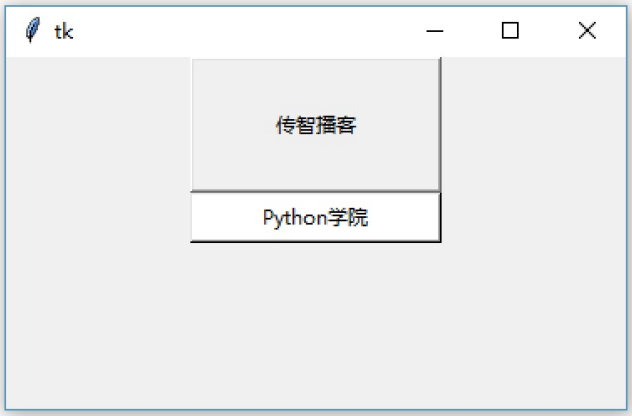
```
from tkinter import *

py = Tk()

b1 = Button(py, text = "传智播客")
b1["width"] = 20
b1["height"] = 4
b1.pack()

b2 = Button(py, text = "Python学院")
b2["width"] = 20
b2["background"] = "white"
b2.pack()

py.mainloop()
```



总结

1. 关于按钮，我们重点理解的就是它如何和事件进行绑定的。
2. 当然，使用一些其他属性来美化按钮也很重要。
3. 下面要讲一讲布局方面的东西了。

布局的讲解

> 对于任何一门图形界面编程来说，布局都是非常重要 的一关，它的英文翻译叫做“layout”。不管是MFC、Java、还是Qt等图形界面编程， 都会有有布局的相关知识。 Python 的 tkinter 也一样。

tkinter的三种布局：

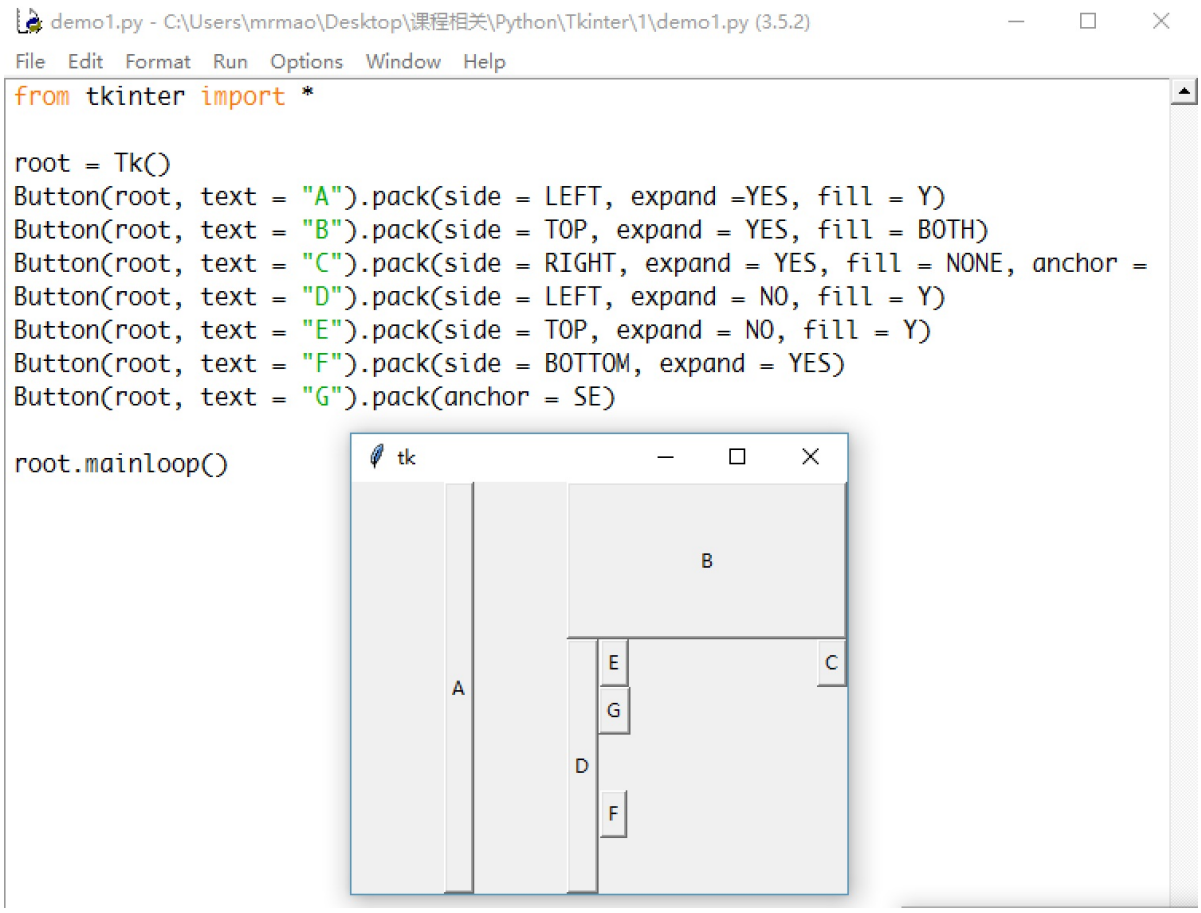
1. 其实我们已经接触过 tkinter 的一种布局，就是 `pack` 布局，它非常简单，我们不用做过多的设置，直接使用一个 `pack` 函数就可以了。
2. grid 布局: grid 可以理解为网格，或者表格，它可以把 界面设置为几行几列的网格，我们在网格里插入我们想要 的元素。这种布局的好处是不管我们如何拖动窗口，相对 位置是不会变化的，而且这种布局也超简单。
3. place 布局:它直接使用死板的位置坐标来布局，这样做 的最大的问题在于当我们向窗口添加一个新部件的时候， 又得重新测一遍数据，且我们不能随便地变大或者缩小窗口，否则可能会导致混乱。

pack 布局

1. 我们使用 pack 函数的时候，默认先使用的放到上面，然 后 依次向下排，它会给我们的组件一个自认为合适的位置 和大小，这是默认方式，也是我们上面一直采用的方式。
2. pack 函数也可以接受几个参数：
   - `side` 参数指定了它停 靠在哪个方向，可以为 LEFT，TOP，RIGHT，BOTTOM，分别代表 左，上，右，下
   - `fill` 参数可以是 X，Y，BOTH 和 NONE，即在水平方向填充，竖直方向填充，水平和竖直方向填充和不填充。
   - `expand` 参数可以是 YES 和 NO，它的 anchor 参数可 以是 N，E，S，W(这里的 NESW 分别表示北东南西，这里分别 表示上右下左)以及他们的组合或者是 CENTER(表示中 间)。
   - `ipadx` 表示的是内边距的 x 方向，它的 ipady 表示 的是内边距的 y 方向，padx 表示的是外边距的 x 方向， pady 表示的是外边距的 y 方向。

pack 的布局实例

1. 根据上面的介绍，我们可以做出如下布局的样子:



2. 按理说做的这么复杂本身没什么意思,只是想让大家看 一下其实 pack 也可以完成相对复杂的布局,它的源代码如下:

```
from tkinter import *

root = Tk()
Button(root, text = "A").pack(side = LEFT, expand =YES, fill = Y)
Button(root, text = "B").pack(side = TOP, expand = YES, fill = BOTH)
Button(root, text = "C").pack(side = RIGHT, expand = YES, fill = NONE, anchor = NE)
Button(root, text = "D").pack(side = LEFT, expand = NO, fill = Y)
Button(root, text = "E").pack(side = TOP, expand = NO, fill = Y)
Button(root, text = "F").pack(side = BOTTOM, expand = YES)
Button(root, text = "G").pack(anchor = SE)

root.mainloop()
```

grid 布局

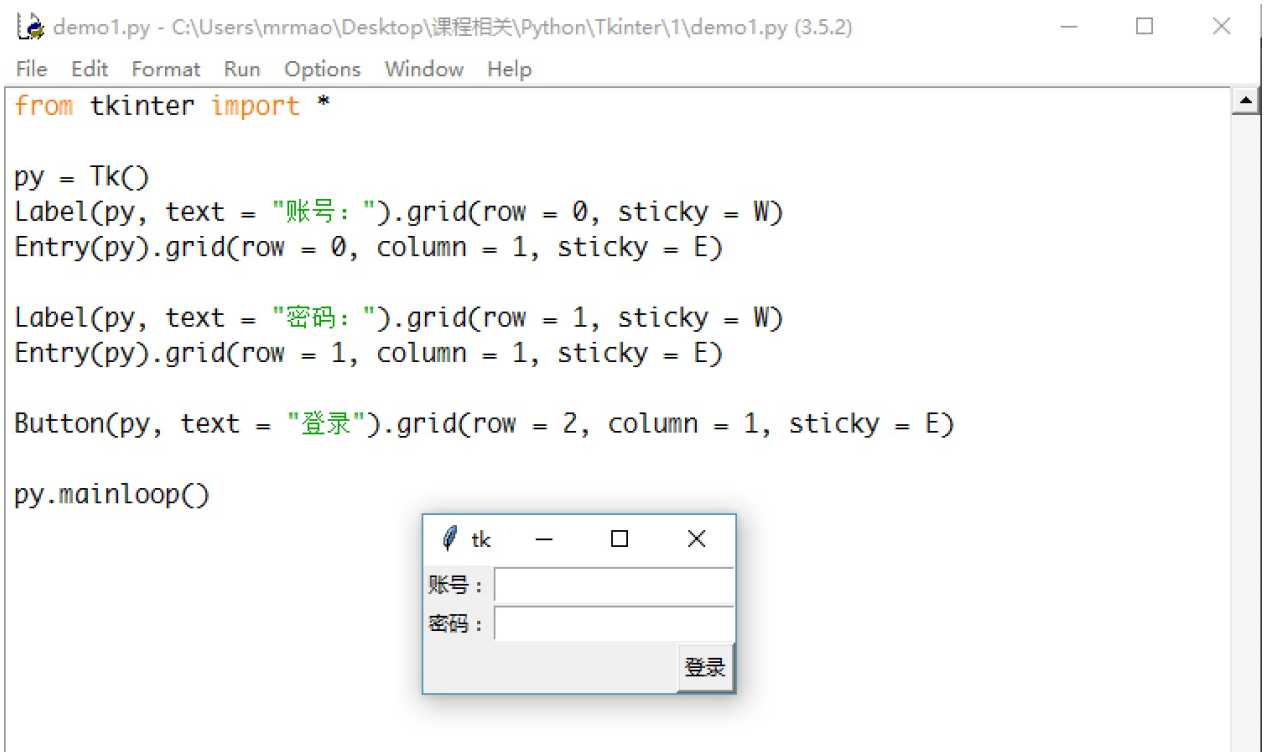
1. 由于我们的程序大多数都是矩形,因此特别适合于网格 布局,也就是 grid 布局。
2. 使用 grid 布局的时候,我们使用 grid 函数,在里面指 定两个参数,用 `row` 表示行,

用 `column` 表示列，注意的是 `row` 和 `column` 的编号都从 0 开始。

3. grid 函数还有个 `sticky` 参数,它可以用 `N,S,W,E` 表示 `上,下,左,右`，它决定了这个组件是从哪个方向开始的, 下面的例子可以很好的解释这一点。
4. grid 布局直接用后面的行和列的数字来指定了它位于哪个位置,而不必使用其他参数。
5. grid 函数也支持诸如 `ipadx,ipady,padx,pady`,它们的意思和 `pack` 函数是一样的,默认边距是 0。
6. 它还支持参数比如 `rowspan`,表示跨越的行数, columnspan 表示跨越的列数。
7. 它还有一些属性,可以在以后我们的demo中慢慢使用来 看出其重要性。

账号登陆例子的截图

1. 界面如下：



2. 代码如下：

```
from tkinter import *

py = Tk()
Label(py, text = "账号: ").grid(row = 0, sticky = W)
Entry(py).grid(row = 0, column = 1, sticky = E)

Label(py, text = "密码: ").grid(row = 1, sticky = W)
Entry(py).grid(row = 1, column = 1, sticky = E)

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

py.mainloop()
```

3. 代码说明: 代码非常简单，参考grid布局介绍理解，其中 `Entry` 表示“输入框”。

place 布局

1. 关于 place 布局,这个的几何管理器组织放置在一个特定的位置
2. 它使用 place 函数,它分为 `绝对布局` 和 `相对布局` ，绝对布局使用 x 和 y 参数,相对布局使用 relx,rely, relheight 和 relwidth 参数。
3. 该方法用的极少，而且极度不推荐大家用，这里就不详细说明了。

总结

1. place 不推荐用，pack 和 grid 布局更常用一些。
2. 但是 pack 和 grid 不能同时用。而且通常对于较为复杂点的界面, 还是建议大家用 gird；如果布局相对简单，使用pack 也很不错。

事件

> - 如果大家学习过 Windows 编程，经常会听到“Windows 是 基于消息的”，而绝大多数界面编程也都会涉及到“事件”。
> - 我们的很多操作，比如我们点击了一下鼠标，这就是一个事件，而操作系统会根据我们的相应的事件产生相应的 消息，操作系统把消息传递给我们的应用程序，然后我们 的应用程序根据操作系统传入的数据执行相应的命令。
> - 事件是我们触发的，消息是操作系统根据我们的事件产生的，我们通常对于“消息”并不多关注，我们重视的是 “事件”。
> - 事件的英文表述是 `event` 。

事件及其绑定

1. 其实，我们在按钮那一节就接触到了事件的绑定，使用 的函数是 `bind` 。
2. `bind` 函数的调用规则: `窗体对象.bind(事件类型，回调函数)`
3. 所谓的“回调函数”，就是这个函数我们不用去调用它， 当相应的事件发生的时候，它会自动取调用。比如当我们 的按钮被按下的时候，它会被自动调用。

常用的事件

1. 我们在使用 bind 函数的时候，它的第一个参数就是事件 的类型了。
2. `<Button-1>` 表示鼠标左键单击，其中的 1 换成 3 表示右 键被单击，为 2 的时候表示鼠标中键，感觉不算常用。
3. `<KeyPress-A>` 表示 A 键被按下，其中的 A 可以换成其他 的键位。
4. `<Control-V>` 表示按下的是 Ctrl 和 V 键，V 可以换成其他 键位。
5. `<F1>` 表示按下的是 F1 键，对于 Fn 系列的，都可以随便 换。

再看绑定

1. 事件不仅可以与 Button 绑定，我们之前看过源代码，发 现 bind 函数是定义在 Misc 类里面的，也就是说，这个 bind 可以被绝大多数组件类所使用。
2. 也就是说，我们可以让“标签”来模拟“按钮”的作用。
3. 因为标签是 Label 类，而 Label 类继承自 Widget，而 Widget 继承自 BaseWidget，而 Basewidget 继承自 Misc。
4. 其实不仅是标签可以模拟 button，任何组件都可以模拟 它，只是那么有用。

标签模拟按钮

1. 代码如下:

```
from tkinter import *

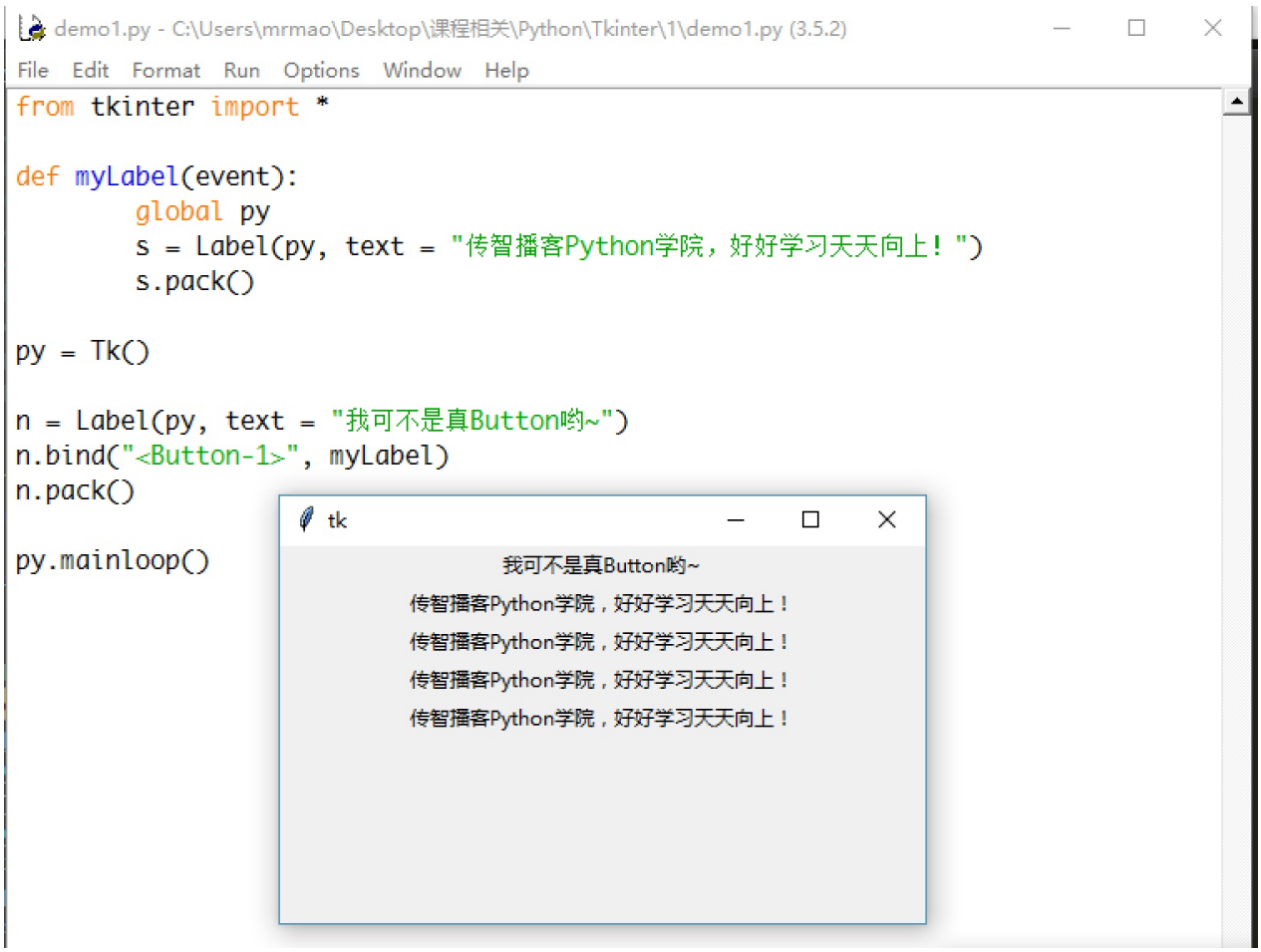
def myLabel(event):
    global py
    s = Label(py, text = "传智播客Python学院，好好学习天天向上！")
    s.pack()

py = Tk()

n = Label(py, text = "我可不是真Button哟~")
n.bind("<Button-1>", myLabel)
n.pack()

py.mainloop()
```

2. 运行效果(下面是点击了四次之后的效果图):



关于 bind 函数

1. 关于 bind 函数，还有两个版本的，不能说高级低级，只是使用的方面不同。
2. 可以在全程序级别的绑定，使用 `bind_all`，它的参数类 型和 `bind` 一样，它通常用于全局的快捷键，比如 F1 通常 是用来打开帮助文档。
3. 还可以绑定某些类别，使用 `bind_class`，它接受三个参数， 第一个参数是类名，第二个参数是事件类型，第三个参数 是相应的操作，比如 `w.bind_class("Entry", "<Control-V>", my_paste)`。它就是绑定了所有的所有的输入框的 *Ctrl+V* 表示粘贴。

解除绑定

1. 接触绑定我们使用 `unbind` 方法，它和 `bind` 的使用很相似。
2. 不过 `unbind` 方法只需要一个参数就可以了，它只需要解除绑定的事件类型，因为它会解除该绑定事件类型的所有回调函数。

输入框以及一个登录程序

输入框的重要性

> 应用程序要取得用户的信息，输入框是必不可少的，虽然执行命令可以使用按钮，但是总不能让用户一直点击按钮把。

Entry

1. 输入框是 Entry，可以阅读它的源代码。
2. 源代码截图:

```
2502
2503  class Entry(Widget, XView):
2504      """Entry widget which allows displaying simple text."""
2505      def __init__(self, master=None, cnf={}, **kw):
2517      def delete(self, first, last=None):
2520      def get(self):
2523      def icursor(self, index):
2526      def index(self, index):
2530      def insert(self, index, string):
2533      def scan_mark(self, x):
2536      def scan_dragto(self, x):
2541      def selection_adjust(self, index):
2544      select_adjust = selection_adjust
2545      def selection_clear(self):
2548      select_clear = selection_clear
2549      def selection_from(self, index):
2552      select_from = selection_from
2553      def selection_present(self):
2558      select_present = selection_present
2559      def selection_range(self, start, end):
2562      select_range = selection_range
2563      def selection_to(self, index):
2566      select_to = selection_to
2567
```

3. 比较重要的也就是get 函数，get 函数使用的时候不需要任何参数，它的返回值就是该输入框的内容。

密码框

1. 其实密码框和输入框基本是一样的，都是向里面输入信 息用的。
2. 如果要说不一样，也就一个地方不一样:密码框需要输 入的信息的显示字符比较单一。
3. 比如 e 是一个输入框，我们可以设置它的 show 属性让它 变成一个密码框，即 `e['show'] = '*'` 就可以了。

小型登录程序

1. 下面是一个小型登录程序，它的用户名是 `itcast`，密码是 `python`，如果输入正确，那么点击“登录”按钮之后，就会显 示“登录成功”，如果输入不符合，那么就会显示“用户 名或者密码错误”，并且清空两个输入框。

2. 运行结果：

```
* 输入正确的用户名和密码，显示`登录成功`：
![](../images/6.0_1.png)

* 输入错误的用户名或密码，显示`用户名错误`或`密码错误`：
![](../images/6.0_2.png)
```

3. 代码演示：

```
from tkinter import *

def reg():
    myAccount = a_entry.get()    # 获取用户输入的用户名
    myPassword = p_entry.get()    # 获取用户输入的密码
    a_len = len(myAccount)        # 获取输入的用户名长度
    p_len = len(myPassword)        # 获取输入的密码长度

    if myAccount == "itcast" and myPassword == "python":
        msg_label["text"] = "登录成功"      # 用户名和密码全部正确    elif myAccount == "itca
st" and myPassword != "python":
        msg_label["text"] = "密码错误"      # 用户名正确密码错误
        p_entry.delete(0, p_len)
    else:
        msg_label["text"] = "用户名错误"     # 用户名错误
        a_entry.delete(0, a_len)
        p_entry.delete(0, p_len)

root = Tk()

# 用户名
a_label = Label(root, text = "用户名：")
a_label.grid(row = 0, column = 0, sticky = W)
a_entry = Entry(root)
a_entry.grid(row = 0, column = 1, sticky = E)

# 密码
p_label = Label(root, text = "密码：")
p_label.grid(row = 1, column = 0, sticky = W)
p_entry = Entry(root)
p_entry["show"] = "*"    # 密码显示为 *
p_entry.grid(row = 1, column = 1, sticky = E)

# 登录按钮
btn = Button(root, text = "登录", command = reg)
btn.grid(row = 2, column = 1, sticky = E)

# 提示信息
msg_label = Label(root, text = "")
msg_label.grid(row = 3)

root.mainloop()
```

菜单

菜单简介

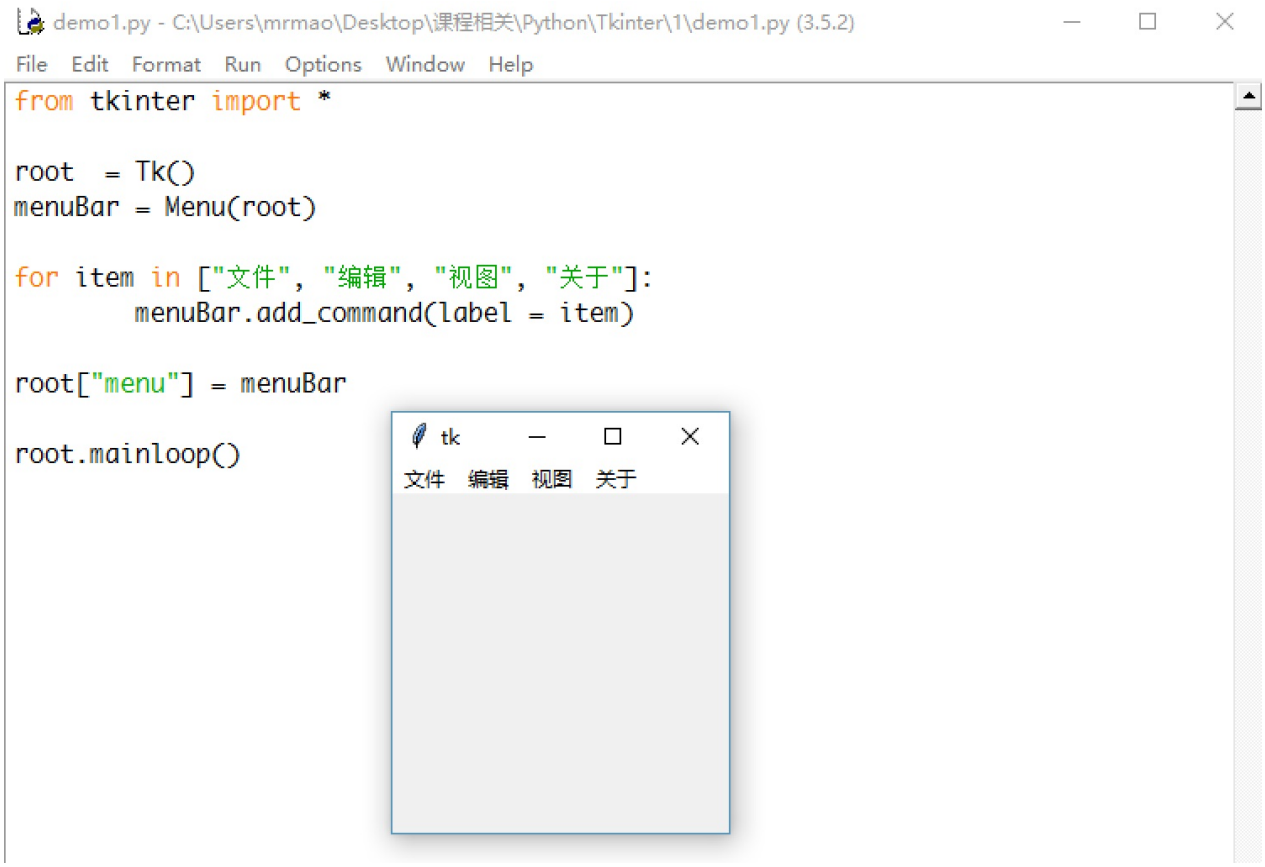
> - 菜单的信息量是非常巨大的，由于菜单又可以有子菜单，因此菜单的信息量非常大。
> - 菜单的分类也较多，通常可以分为下拉菜单、弹出菜单 等等。

添加顶层菜单

1. 我们可以使用 `Menu` 类来新建一个菜单， Menu 和其他的组件一样，第一个是 `parent` ，这里通常可以为窗口。
2. 然后我们可以用 `add_commmand` 方法来为它添加菜单项， 如果该菜单是顶层菜单，则添加的菜单项依次向右添加。 如果该菜单时顶层菜单的一个菜单项，则它添加的是下拉 菜单的菜单项。
3. `add_command` 中的参数常用的有 `label` 属性，用来指定的 是菜单项的名称， `command` 属性用来指定被点击的时候调用 的方法， `acceletor` 属性指定的是快捷键， `underline` 属性 是是否拥有下划线。
4. 最后可以用窗口的 `menu` 属性指定我们使用哪一个作为它 的顶层菜单。

代码演练

1. 这里只是做出了顶级菜单，它们四个是并列的一行，并 没有实现什么功能，效果图如下:



2. 下面是代码演示:

```
from tkinter import *

root  = Tk()
menuBar = Menu(root)

for item in ["文件", "编辑", "视图", "关于"]:
    menuBar.add_command(label = item)

root["menu"] = menuBar

root.mainloop()
```

3. 可以看到，非常简单的方式，我们使用 `add_command` 来 添加菜单项即可。

有子菜单的情况

1. 如果有子菜单，则情况稍微复杂点，这个时候，我们需 要使用 `add_cascade`，cascade 可以理解为“级联”，即它 的作用只是为了引出后面的菜单。

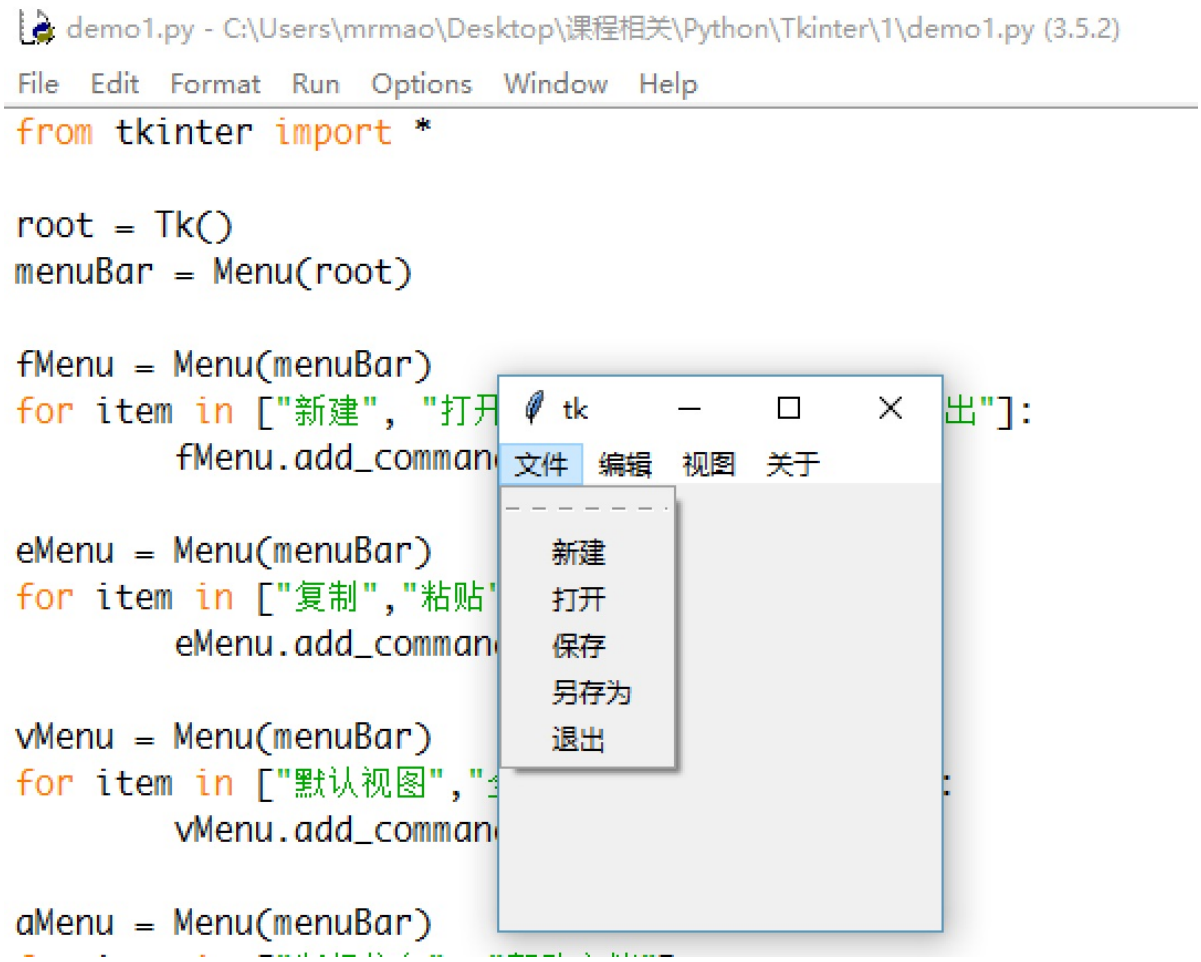
2. `add_cascade` 的一个很重要的属性就是 `menu` 属性，它指 明了要把那个菜单级联到该菜单项上，当然，还必不可少 的就是 label 属性，用于指定该菜单项的名称。

3. 我们先新建一个 Menu 的实例，然后使用 `add_command` 来 添加菜单项，这样等该菜单建立完毕，我们要把它作为另 一个菜单项的子菜单，就需要使用 `add_cascade` 方法。

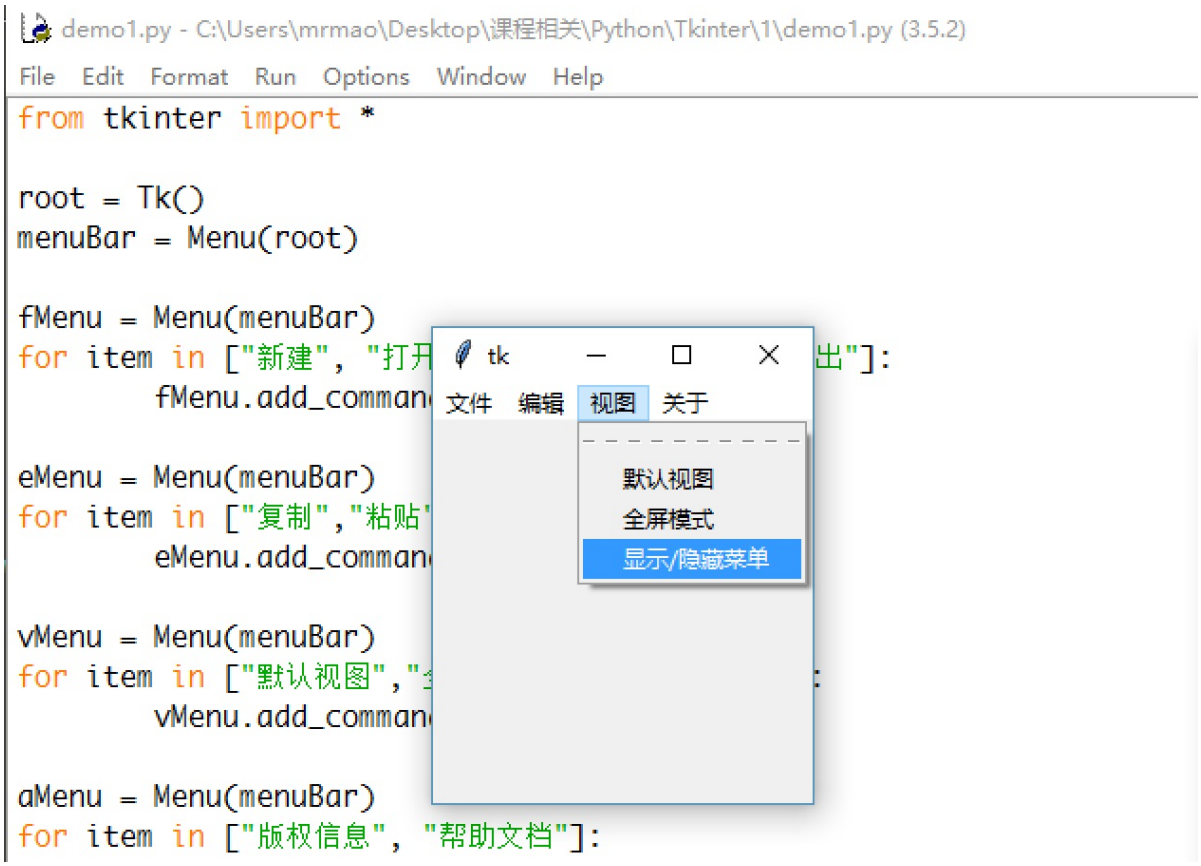
代码演练

1. 下面我们使用了下拉菜单，这也是我们通常使用的菜单 形式，即一行菜单项，点击之后会产生一个下拉菜单。

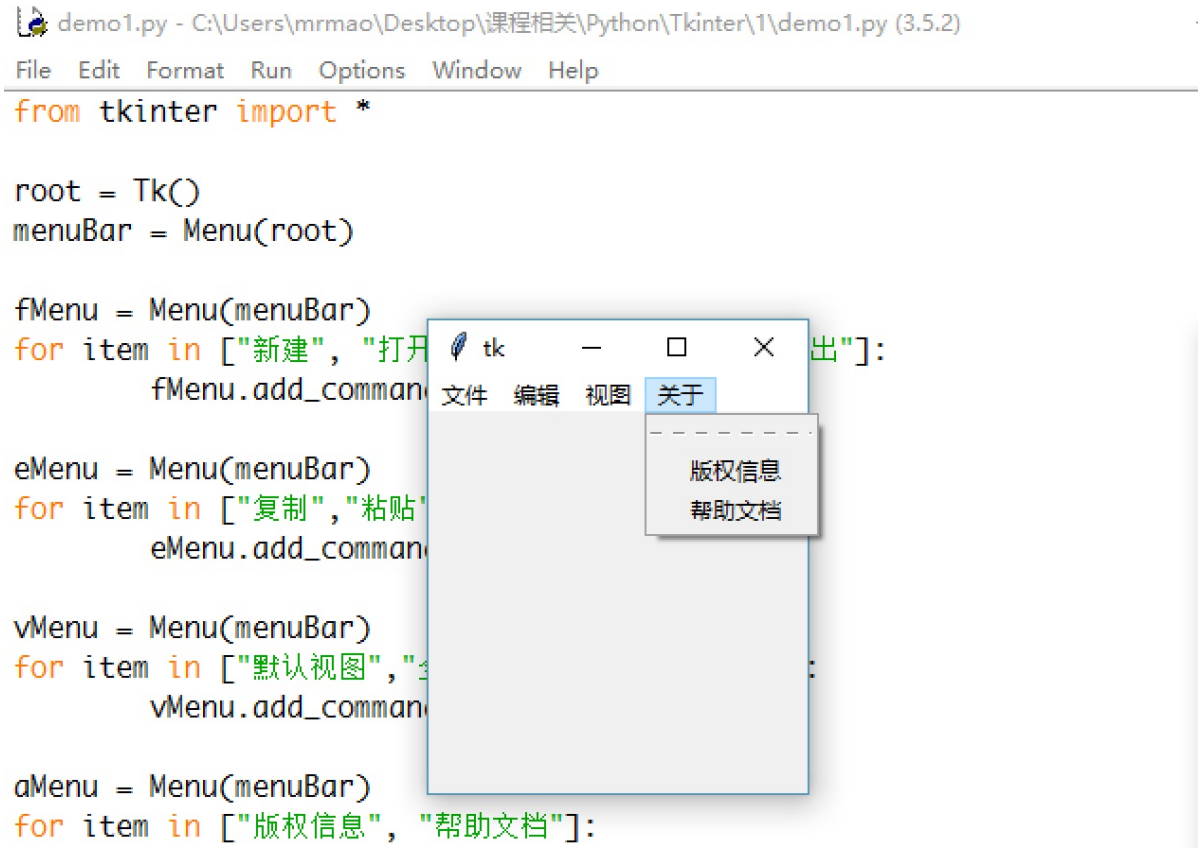
2. 效果截图一:



3. 截图效果二:



4. 截图效果三:



5. 代码演示:

```
from tkinter import *

root = Tk()
menuBar = Menu(root)

fMenu = Menu(menuBar)    # 文件
for item in ["新建", "打开", "保存", "另存为", "退出"]:
    fMenu.add_command(label = item)

eMenu = Menu(menuBar)    # 编辑
for item in ["复制", "粘贴", "剪切", "撤销"]:
    eMenu.add_command(label = item)

vMenu = Menu(menuBar)    # 视图
for item in ["默认视图", "全屏模式", "显示/隐藏菜单"]:
    vMenu.add_command(label = item)

aMenu = Menu(menuBar)    # 关于
for item in ["版权信息", "帮助文档"]:
    aMenu.add_command(label = item)

menuBar.add_cascade(label = "文件", menu = fMenu)
menuBar.add_cascade(label = "编辑", menu = eMenu)
menuBar.add_cascade(label = "视图", menu = vMenu)
menuBar.add_cascade(label = "关于", menu = aMenu)

root["menu"] = menuBar

root.mainloop()
```

这里要注意的是我们可以先把子菜单做好，然后再做上层菜单。

弹出菜单

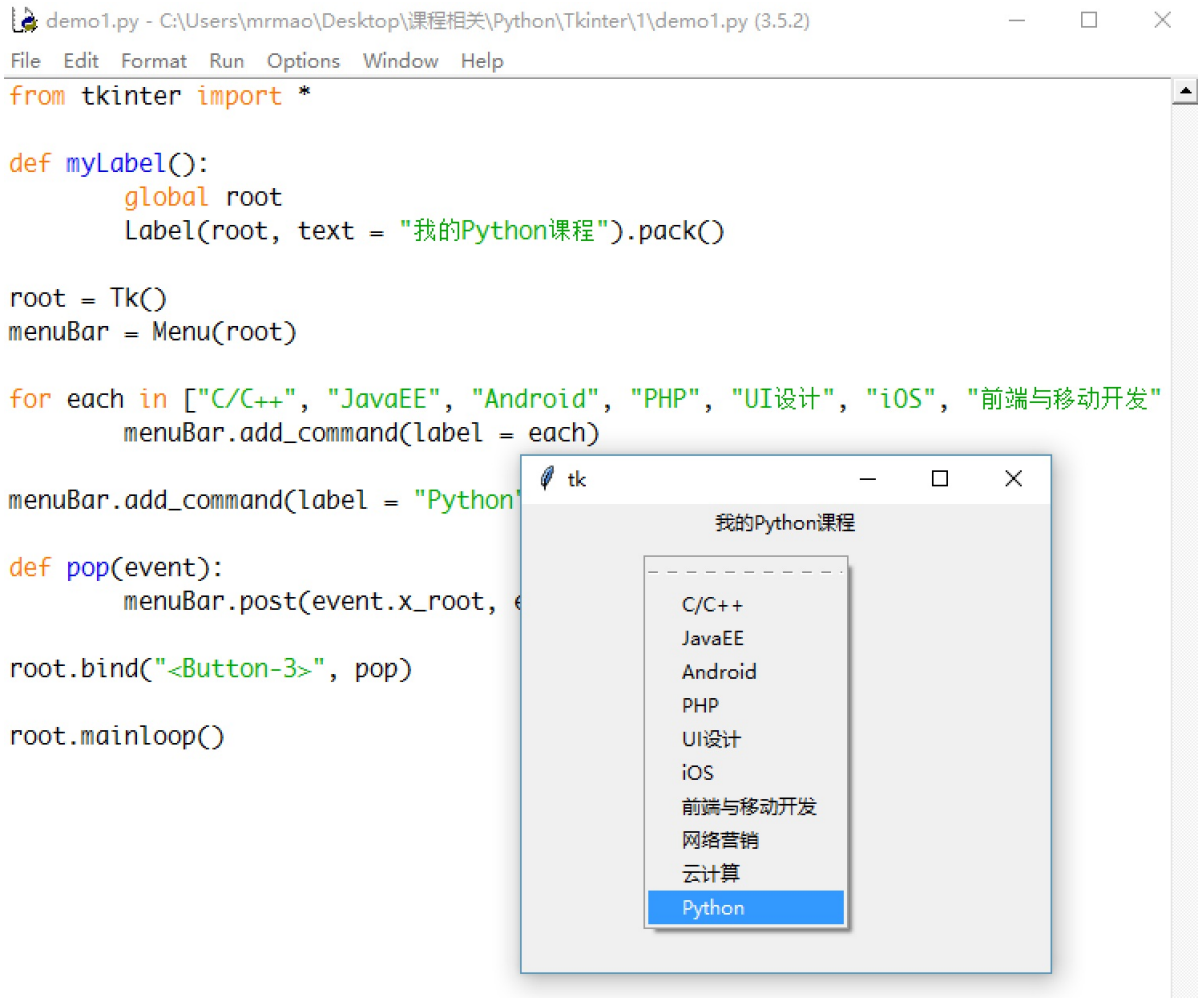
1. 弹出菜单又叫“上下文菜单”，也叫“右键菜单”，它 通常是鼠标单击右键产生的菜单，因此会有“右键菜单” 的说法。
2. 其实很多界面库里面都是给出了弹出菜单的简单的制作 方法的，但是 tkinter 里面我们却只能使用比较原始的事 件绑定的方式去做。
3. 大体思路就是:我们先新建一个菜单，然后向菜单项中 添加各种功能，最后我们监听鼠标右键消息，如果是鼠标 右键被单击，此时可以根据需要判断下鼠标位置来确定是 哪个弹出菜单被弹出，然后使用 `Menu` 类的 `pop` 方法来弹出 菜单。
4. 大体思路就是如此，至于具体的细节，让我们到代码实 战中一探究竟。

5. Menu 类里面有一个 `post` 方法，它接收两个参数，即 `x` 和 `y` 坐标，它会在相应的位置弹出菜单。

6. 还记得用 `bind` 方法来绑定事件吗?而且要记得鼠标右键 是用的 `<Button-3>`

代码演练

1. 界面效果(该右键菜单中如果点击 python 选项，则会新 建一个标签，标签内容是“我的Python课程”):



2. 代码演示:

```
from tkinter import *

def myLabel():
    global root
    Label(root, text = "我的Python课程").pack() # 点击Python后添加这个标签

root = Tk()
menuBar = Menu(root)

for each in ["C/C++", "JavaEE", "Android", "PHP", "UI设计", "iOS", "前端与移动开发", "网络营
销", "云计算"]:
    menuBar.add_command(label = each)

menuBar.add_command(label = "Python", command = myLabel)

def pop(event):
    menuBar.post(event.x_root, event.y_root)

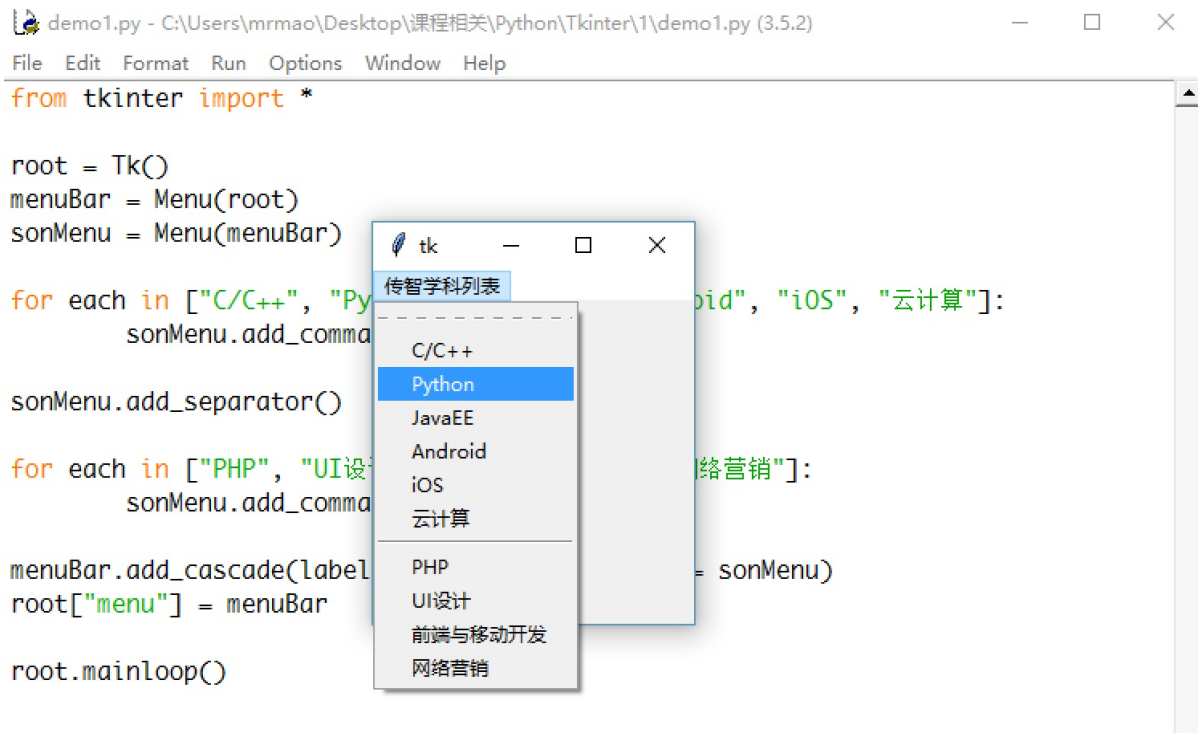
root.bind("<Button-3>", pop)  # 鼠标右击绑定

root.mainloop()
```

此处利用了 *Menu* 的 *post* 方法，还有 *bind* 方法，一定要 记住鼠标右键的事件名称，这些用多了之后自然能记住。

分割线

1. 有时候，一个菜单项的各个菜单可能并不是一个类型， 有可能是两种类型，在它们中间可以插一个分割线来界定 界限。
2. 插入分割线和插入正常的菜单项操作很相似，只是使用 的方法是 add_separator，该方法无需参数。
3. 插入分割线效果示例(在列表中插入了一个分割线):



4. 插入分割线代码示例:

```
from tkinter import *

root = Tk()
menuBar = Menu(root)
sonMenu = Menu(menuBar)

for each in ["C/C++", "Python", "JavaEE", "Android", "iOS", "云计算"]:
    sonMenu.add_command(label = each)

sonMenu.add_separator()        # 插入分割线

for each in ["PHP", "UI设计", "前端与移动开发", "网络营销"]:
    sonMenu.add_command(label = each)

menuBar.add_cascade(label = "传智学科列表", menu = sonMenu)
root["menu"] = menuBar

root.mainloop()
```

radiobutton 与 checkbutton

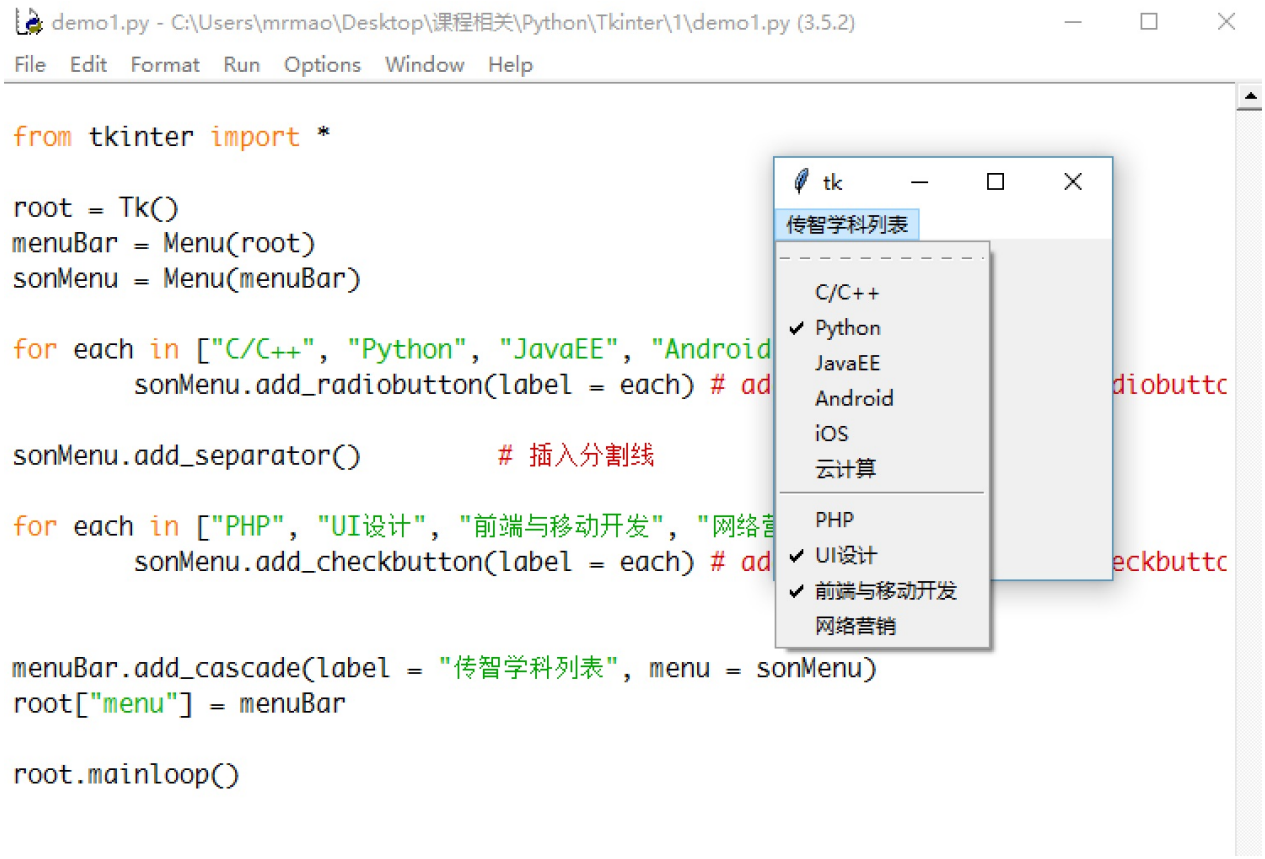
1. `radiobutton` 是指“单选按钮”，`checkbutton` 是指“复选按钮”。
2. 单选按钮与复选按钮是相对来说的，即在这一组中，单选按钮中只能有一个是被选定的，即一个人的性别是女的， 就一定不是男的，但是复选按钮对应的情况则是可以有多个同时被选定，比如一个人即可以喜欢篮球，也可以喜欢 足球，还有可能喜欢

乒乓球，等等。

3. 在菜单中，也有类似的概念，即 `单选菜单` 和 `复选菜单` 。 它们分别用 `add_radiobutton` 和 `add_checkbutton` 来分别 添加。
4. 这两种菜单都是如果一旦被选定，那么前面会有一个类 似于对号的标记出现，checkbutton 可以多个同时被选定， 但是 radiobutton 却只能被选定一个，即这个被选定了， 会自动取消前一个的选定。

代码与效果截图

效果图(上面是必须单选的，下面则是可以复选):



代码演示:

```
from tkinter import *

root = Tk()
menuBar = Menu(root)
sonMenu = Menu(menuBar)

for each in ["C/C++", "Python", "JavaEE", "Android", "iOS", "云计算"]:
    sonMenu.add_radiobutton(label = each) # add_command() 改成 add_radiobutton()

sonMenu.add_separator()        # 插入分割线

for each in ["PHP", "UI设计", "前端与移动开发", "网络营销"]:
    sonMenu.add_checkbutton(label = each) # add_command() 改成 add_checkbutton()


menuBar.add_cascade(label = "传智学科列表", menu = sonMenu)
root["menu"] = menuBar

root.mainloop()
```

对话框和消息框

对话框

1. 关于对话框，tkinter 有它自己的布局，在 Python 的安 装目录下的 Lib 文件夹的 tkinter 子文件夹下，有个 dialog.py，它只有 2KB，它就是我们要介绍的重点。
2. 开该文件，只有五十行，还包括用来测试的代码，里 面只有一个 Dialog 类，它继承自 Widget，它的方法也很简 单，基本上也就一个**init**方法有实质性内容，下面的 _test 是用来测试的。
3. 该类可用的属性很少，基本上真正比较好用的也就一个 num 属性了，它用于返回用户的点击。
4. 我们可以通过它给出的例子来模仿写出一个对话框，注 意它的 title 属性，text 属性，strings 属性。
5. 下面是该文件的截图:

```
# dialog.py -- Tkinter interface to the tk_dialog script.

from tkinter import *
from tkinter import _cnfmerge

if TkVersion <= 3.6:
    DIALOG_ICON = 'warning'
else:
    DIALOG_ICON = 'questhead'


class Dialog(Widget):
    def __init__(self, master=None, cnf={}, **kw):
    def destroy(self): pass

def _test():


if __name__ == '__main__':
    t = Button(None, {'text': 'Test',
                      'command': _test,
                      Pack: {}})
    q = Button(None, {'text': 'Quit',
                      'command': t.quit,
                      Pack: {}})
    t.mainloop()

```

调查Demo

1. 下面是界面:



2. 代码演示:

```
from tkinter.dialog import *
from tkinter import *

def myDialog():
    d = Dialog(None, title = "Python调查", text = "喜欢Python吗? ", bitmap = DIALOG_ICON, d
efault = 0, strings = ("喜欢", "很喜欢", "非常喜欢"))
    print(d.num)

btn_begin = Button(None, text = "Python调查", command = myDialog)
btn_begin.pack()

btn_quit = Button(None, text = "关闭", command = btn_begin.quit)
btn_quit.pack()

btn_begin.mainloop()
```

3. 代码说明:

在新建一个对话框的时候，title 属性，text 属性，bitmap 属性等等都是不可缺少的，否则会报错，我 第一次写的时候就报错了，后来看了下源代码，发现这些 都必须自己填写，因为没有设置默认值。

其他对话框

1. 其实我们也看到了，由于源代码中给出的关于对话框的 代码过于简短，真正用起来并不方便。
2. 于是，根据使用频率，源代码又给出了几个标准对话框， 比如 simpledialog(简单对话框)，commondialog(一般 对话框)，filedialog(文件对话框)，其实 colorchooser 也算对话框的内容。
3. 我看了下它们的源代码，但是考虑到我们的程序的主要 用途，所以就不在此节深入介绍了，具体的大家可以深入 源代码一探究竟。

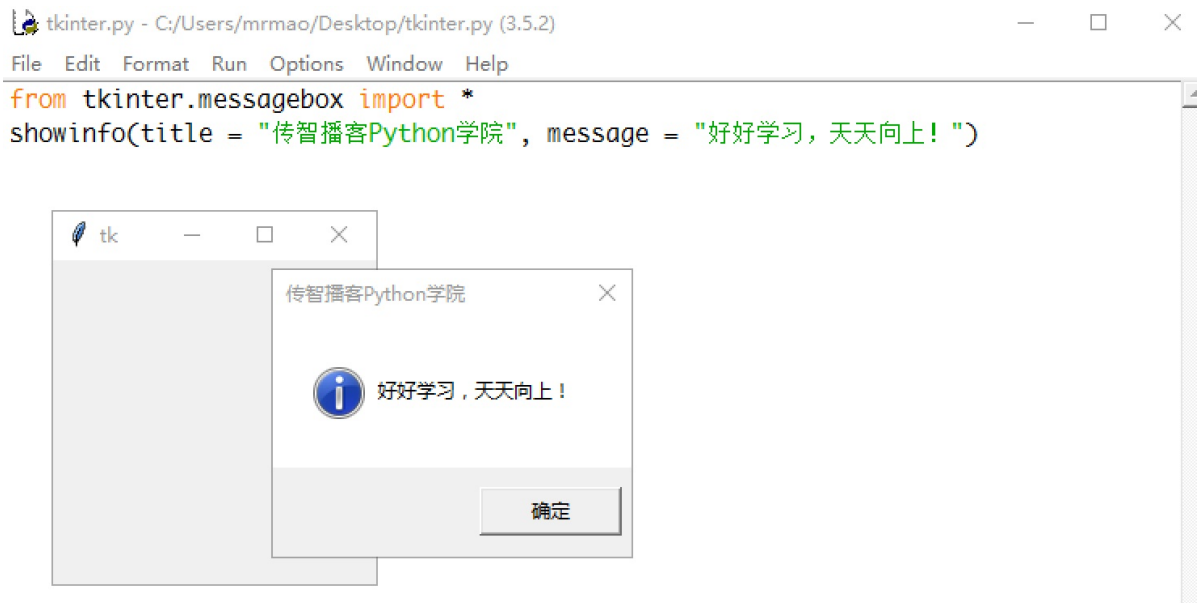
消息框

1. 和上面的对话框一样，在同一目录下会看到有个 messagebox.py，打开一看，发现也很简短，只有一百多行。
2. 且里面主要是一些函数，这些函数的参数都给了默认值， 我们只需要填充那些我们感兴趣的值即可。

3. 下面是源代码中该文件的主要函数截图:

```
56
57  class Message(Dialog):
61
62
63  #
64  # convenience stuff
65
66  # Rename _icon and _type options to allow overriding them in options
67  def _show(title=None, message=None, _icon=None, _type=None, **options):
80
81  def showinfo(title=None, message=None, **options):
84
85  def showwarning(title=None, message=None, **options):
88
89  def showerror(title=None, message=None, **options):
92
93  def askquestion(title=None, message=None, **options):
96
97  def askokcancel(title=None, message=None, **options):
101
102 def askyesno(title=None, message=None, **options):
106
107 def askyesnocancel(title=None, message=None, **options):
115
116 def askretrycancel(title=None, message=None, **options):
120
121 # --------------------------------------------------------------------
122 # test stuff
123
124 if __name__ == "__main__":
125
126     print("info", showinfo("Spam", "Egg Information"))
127     print("warning", showwarning("Spam", "Egg Warning"))
128     print("error", showerror("Spam", "Egg Alert"))
129     print("question", askquestion("Spam", "Question?"))
130     print("proceed", askokcancel("Spam", "Proceed?"))
131     print("yes/no", askyesno("Spam", "Got it?"))
132     print("yes/no/cancel", askyesnocancel("Spam", "Want it?"))
133     print("try again", askretrycancel("Spam", "Try again?"))
134
```

4. 使用消息框的效果图:



5. 使用消息框的代码截图:

```
from tkinter.messagebox import *
showinfo(title = "传智播客Python学院", message = "好好学习，天天向上！")
```

常用控件介绍

控件

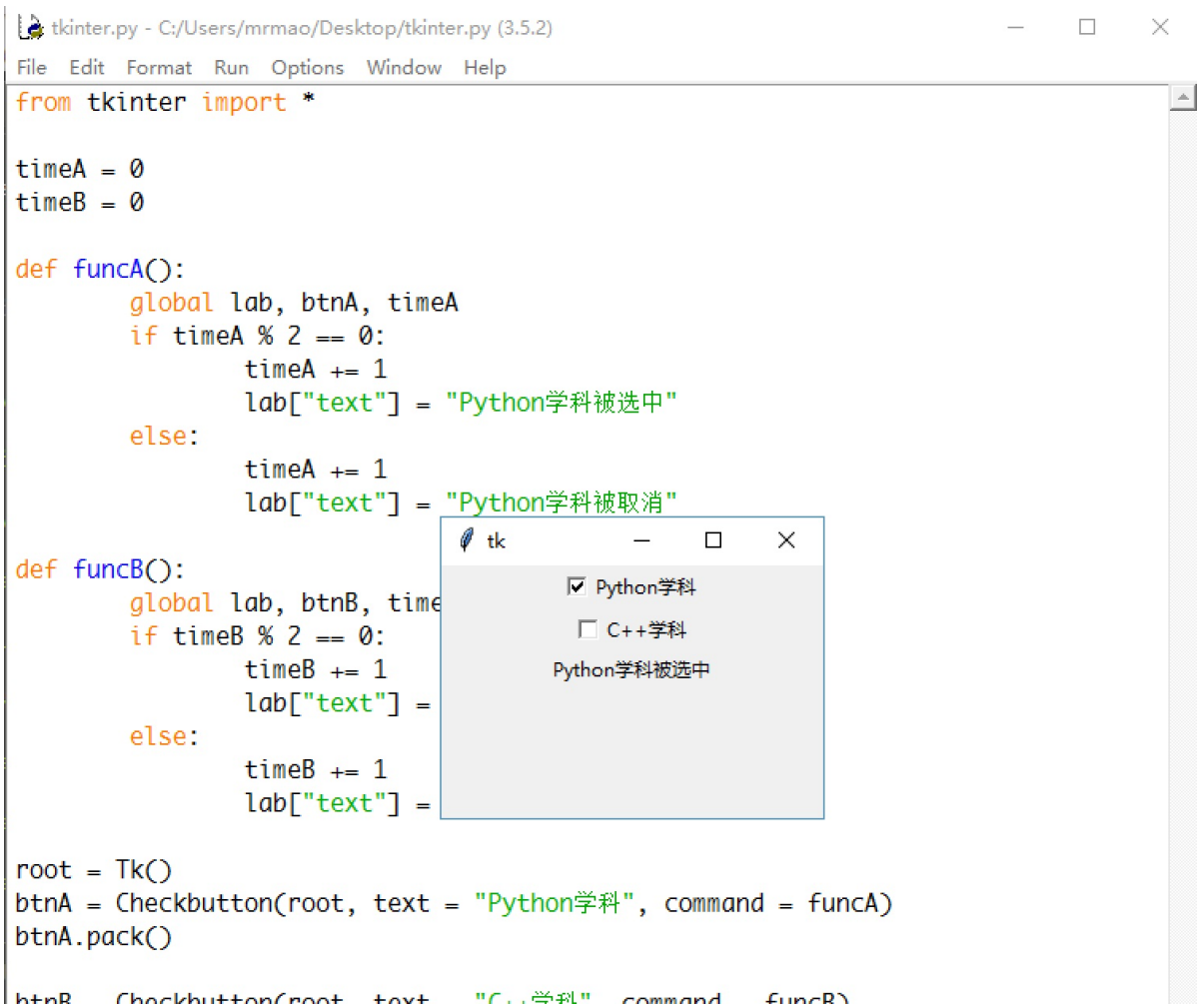
> 1. 在前面几节中，介绍了几个简单的控件，比如标签，比 如按钮，比如输入框等等。
> 2. 说实话，标准的 tkinter 中给出的控件数目并不多，只 有 21 个，虽然后来的 ttk 中又增加了几个，但是还是跟不 上需要。关于 ttk 后面会介绍。
> 3. 但是这里还是介绍下常用的控件把，毕竟这些控件还是 蛮重要的。

复选按钮

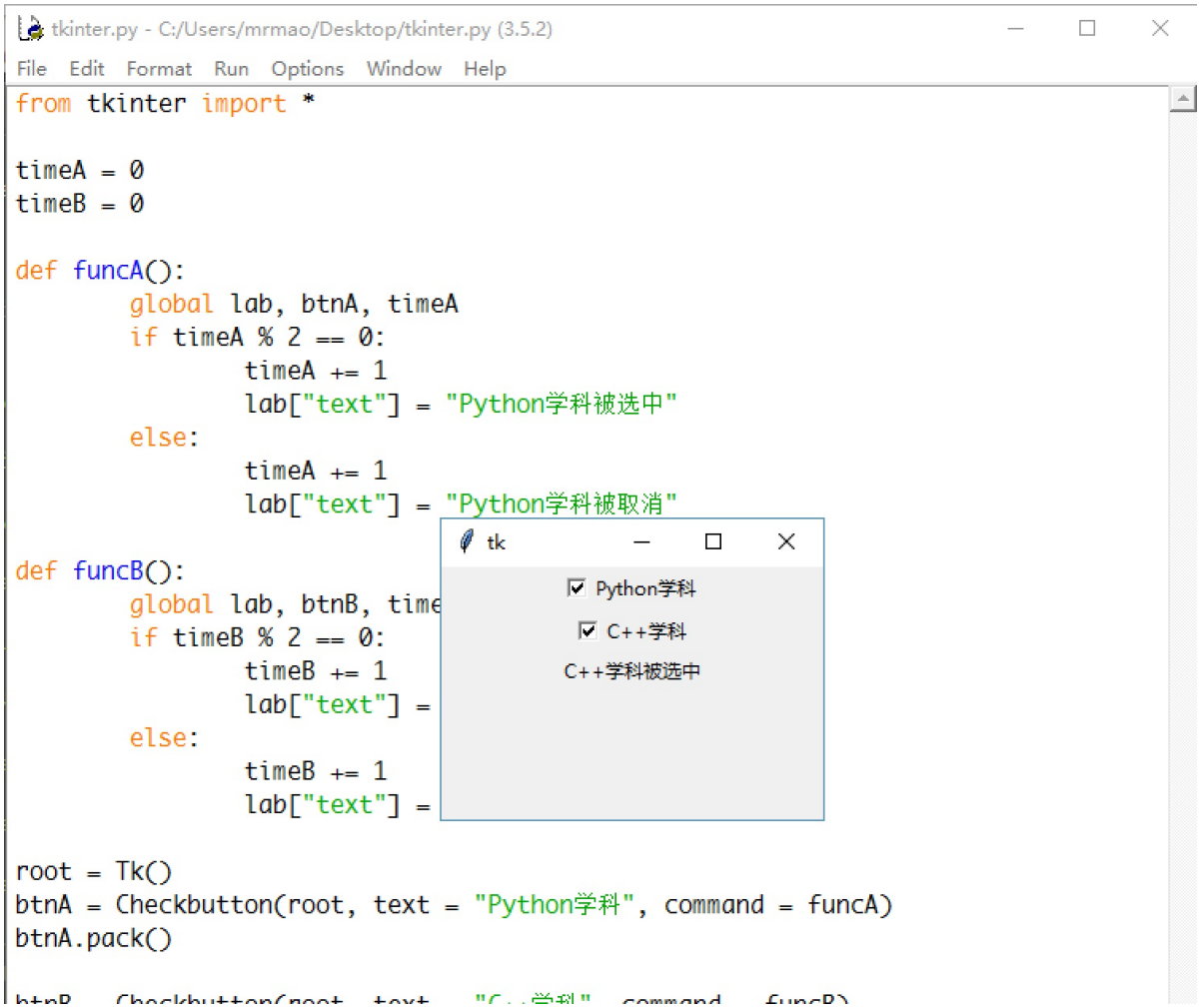
1. 复选按钮就是 Checkbutton 类，它的实例化和 Button 很 相似。
2. 既然是按钮，那就可以有 command 属性，该属性可以对 应到一个函数上去来执行某些功能。
3. 复选框通常是用来选择信息的时候的一种选择，它前面 有个小正方形的方块，如果选中则有一个对号，也可以再 次点击以取消该对号来取消选中。

复选框代码实例

1. 该实例，使用了两个复选框，点击那个复选框，如果处 于选中状态，则在下面的标签中显示被选中的字样，如果 没有被选中，则显示未被选中的字样。
2. 效果截图一:


tkinter.py - C:/Users/mrmao/Desktop/tkinter.py (3.5.2)
File Edit Format Run Options Window Help
from tkinter import *
timeA = 0
timeB = 0
def funcA():
global lab, btnA, timeA
if timeA % 2 == 0:
timeA += 1
lab["text"] = "Python学科被选中"
else:
timeA += 1
lab["text"] = "Python学科被取消"
def funcB():
global lab, btnB, time
if timeB % 2 == 0:
timeB += 1
lab["text"] =
else:
timeB += 1
lab["text"] =
root = Tk()
btnA = Checkbutton(root, text = "Python学科", command = funcA)
btnA.pack()
tk
Python学科
C++学科
Python学科被选中

3. 效果截图二:


tkinter.py - C:/Users/mrmao/Desktop/tkinter.py (3.5.2)
File Edit Format Run Options Window Help
from tkinter import *
timeA = 0
timeB = 0
def funcA():
global lab, btnA, timeA
if timeA % 2 == 0:
timeA += 1
lab["text"] = "Python学科被选中"
else:
timeA += 1
lab["text"] = "Python学科被取消"
def funcB():
global lab, btnB, time
if timeB % 2 == 0:
timeB += 1
lab["text"] =
else:
timeB += 1
lab["text"] =
root = Tk()
btnA = Checkbutton(root, text = "Python学科", command = funcA)
btnA.pack()
tk
Python学科
C++学科
C++学科被选中

4. 代码示例:

```
from tkinter import *

timeA = 0
timeB = 0

def funcA():
    global lab, btnA, timeA
    if timeA % 2 == 0:
        timeA += 1
        lab["text"] = "Python学科被选中"
    else:
        timeA += 1
        lab["text"] = "Python学科被取消"

def funcB():
    global lab, btnB, timeB
    if timeB % 2 == 0:
        timeB += 1
        lab["text"] = "C++学科被选中"
    else:
        timeB += 1
        lab["text"] = "C++学科被取消"

root = Tk()
btnA = Checkbutton(root, text = "Python学科", command = funcA)
btnA.pack()

btnB = Checkbutton(root, text = "C++学科", command = funcB)
btnB.pack()

lab = Label(root, text = " ")
lab.pack()

root.mainloop()
```

代码也很简单，用于控制次数的 *timeA* 和 *timeB*

单选框

1. 单选框和复选框非常相似，只是把 Checkbutton 换成 Radiobutton。
2. 我就不代码示例了，因为实在是太相似且简单了。
3. 可以参考单选框类源代码。

文本域

1. 所谓文本域，也就是文本，其实它可以看做一个大型的文本框，它的属性也更多一些。
2. 文本域效果展示:



3. 代码示例:

```
from tkinter import *

root = Tk()
t = Text(root, width = 50, height = 30)
t.pack()

root.mainloop()
```

Canvas

1. canvas 直接翻译就是“帆布”，其实可以理解为一个画布， 用于在上面绘制图形。
2. 它也是一个控件，但是在它上面的区域，可以用于手工 绘制，对于这个手工绘图的东西，说实话，本人并不感兴 趣，但是该类有很大， 供了很多绘制的函数。
3. 后面的项目会用到它。

Toplevel

1. 一个 toplevel 可以理解为一个新的窗口，它是一个顶层 窗口。

2. 新建一个 toplevel 很简单，只需要实例化该类即可，这 也是创建多窗口 应用的一个途径。

3. 下面是一个多窗口应用程序的截图:



4. 代码截图:

```
from tkinter import *

root = Tk()
root.title("我是root窗口")
lr = Label(root, text = "我属于root")
lr.pack()

top = Toplevel(root, width = 30, height = 20)
top.title("我是toplevel窗口")
lt = Label(top, text = "我属于toplevel")
lt.pack()

root.mainloop()
```

手绘图形

窗口重绘

1. 第一次认识到手绘图形的重要性还是在学习 MFC 的时候, 因为 MFC 自带的绘图功能实在过于丑陋,我们可以重绘标题栏、菜单栏、最小化按钮、最大化按钮、关闭按钮等窗 口组件来让窗口得到美化,除了这些,还可以手绘按钮, 后来出了一门技术, 叫做“Direct UI”,也是这种思想的进一步发扬光大把。
2. 但是 tkinter 没有这些方面的接口,我也深感遗憾。但是 tkinter 有一个绘图功能的组件,即 Canvas,翻译成汉 语即“帆布”,可以理解为“画布”,即用于绘制图形。

canvas

1. 下面是该类的部分截图:

```
class Canvas(Widget, XView, YView):
    """Canvas widget to display graphical elements like lines or text."""
    def __init__(self, master=None, cnf={}, **kw): ...
    def addtag(self, *args): ...
    def addtag_above(self, newtag, tagOrId): ...
    def addtag_all(self, newtag): ...
    def addtag_below(self, newtag, tagOrId): ...
    def addtag_closest(self, newtag, x, y, halo=None, start=None): ...
    def addtag_enclosed(self, newtag, x1, y1, x2, y2): ...
    def addtag_overlapping(self, newtag, x1, y1, x2, y2):
        """Add tag NEWTAG to all items which overlap the rectangle
        defined by X1,Y1,X2,Y2."""
        self.addtag(newtag, 'overlapping', x1, y1, x2, y2)
    def addtag_withtag(self, newtag, tagOrId): ...
    def bbox(self, *args): ...
    def tag_unbind(self, tagOrId, sequence, funcid=None): ...
    def tag_bind(self, tagOrId, sequence=None, func=None, add=None): ...
    def canvasx(self, screenx, gridspacing=None): ...
    def canvasy(self, screeny, gridspacing=None): ...
    def coords(self, *args):
        """Return a list of coordinates for the item given in ARGS."""
        # XXX Should use _flatten on args
        return [self.tk.getdouble(x) for x in
                                self.tk.splitlist(
                    self.tk.call((self._w, 'coords') + args))]
    def _create(self, itemType, args, kw): # Args: (val, val, ..., cnf={}) ...
    def create_arc(self, *args, **kw): ...
    def create_bitmap(self, *args, **kw): ...
    def create_image(self, *args, **kw): ...
    def create_line(self, *args, **kw): ...
    def create_oval(self, *args, **kw): ...
    def create_polygon(self, *args, **kw): ...
    def create_rectangle(self, *args, **kw): ...
    def create_text(self, *args, **kw): ...
    def create_window(self, *args, **kw): ...
    def dchars(self, *args): ...
    def delete(self, *args): ...
    def dtag(self, *args): ...
```

2. 其实我们主要的也就是用上面的这些绘图函数来进行手 工的绘制一些东西。

简单示例

1. 我们第一个示例随便绘制了,先把背景用 rgb 格式刷成 蓝色,然后画一条线,然后写几个字。
2. 代码截图:

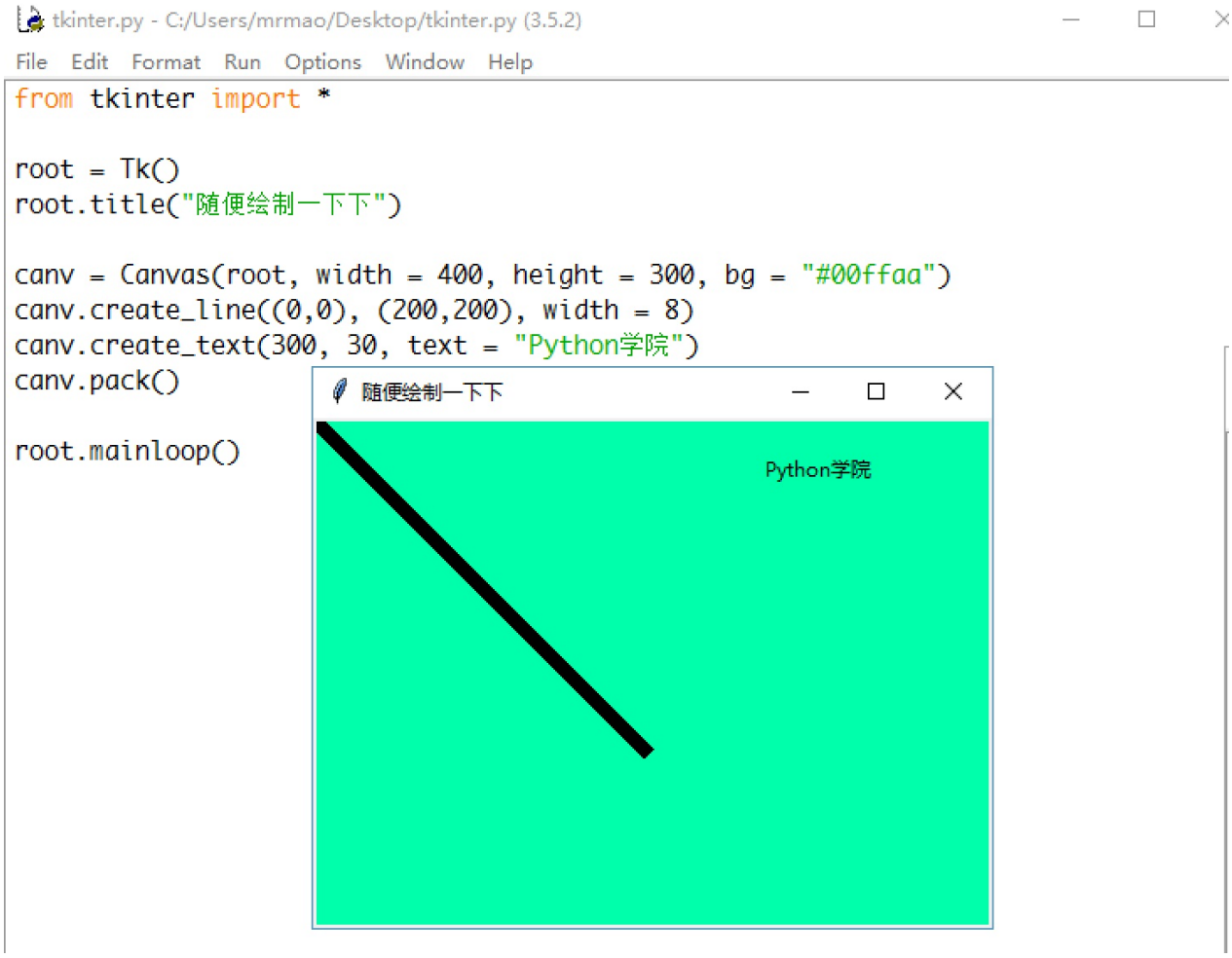
```
from tkinter import *

root = Tk()
root.title("随便绘制一下下")

canv = Canvas(root, width = 400, height = 300, bg = "pink")
canv.create_line((0,0), (200,200), width = 8)
canv.create_text(300, 30, text = "Python学院")
canv.pack()

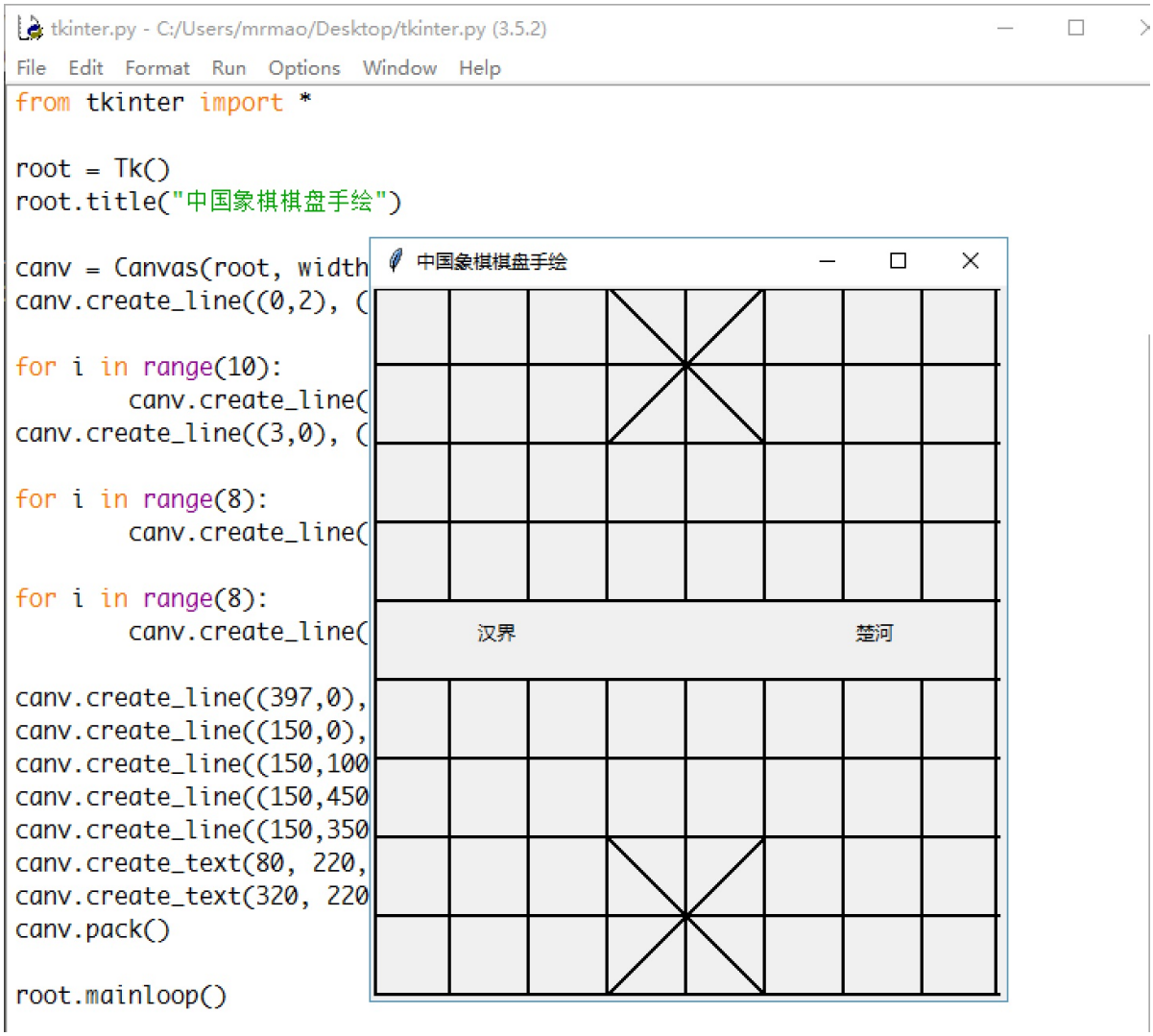
root.mainloop()
```

1. 效果图如下:



中国象棋棋盘

1. 其实手绘可以做很多事,很多东西都可以手绘,只不过 有些东西确实手绘挺累的。
2. 下面是手绘了一个中国象棋棋盘的截图(关于中国象棋, 我还是挺喜欢玩的):



3. 其实没啥技术难度,下面是代码截图,不过我还是希望 读者先亲自动手做一个,毕竟不难:

```
from tkinter import *

root = Tk()
root.title("中国象棋棋盘手绘")

canv = Canvas(root, width = 400, height = 450)
canv.create_line((0,2), (400,2), width = 2)

for i in range(10):
    canv.create_line((0, i * 50), (400, i * 50), width = 2)
canv.create_line((3,0), (3,450), width = 2)

for i in range(8):
    canv.create_line((i * 50,0), (i * 50,200), width = 2)

for i in range(8):
    canv.create_line((i * 50,250), (i * 50,450), width = 2)

canv.create_line((397,0), (397,450), width = 2)
canv.create_line((150,0), (250,100), width = 2)
canv.create_line((150,100), (250,0), width = 2)
canv.create_line((150,450), (250,350), width = 2)
canv.create_line((150,350), (250,450), width = 2)
canv.create_text(110, 220, text = "汉界")
canv.create_text(290, 220, text = "楚河")
canv.pack()

root.mainloop()
```

窗口的一些美化

美化

> 1. GUI 编程自从出现的那一天开始，就凸显除了强劲的魅 力 ，因为它对用户来说实在是太方便了。
> 2. 但是 GUI 编程尤其的繁琐且啰嗦，因为界面编写确实令 人头疼，可能初学者对于写界面会很兴奋，但是写一段时 间就会感觉很烦了。
> 3. 但是如果你致力于创造更加简洁大方的界面，那么还是努力吧，毕竟，清爽的界面谁都喜欢用。

窗口的一些控制

1. 我们可以用 title 函数来修改窗口的默认标题，该参数 直接接收一个字符串参数。
2. 我们还可以用 geometry 函数来控制窗口大小，它接受一 个字符串类型的参数，但是它的格式很严格，是这样的 ‘width x height + xoffset + yoffset’，比如我写成 '300x280+150+200' 就是一个合格的参数，注意的是这里 的 x 就是 xyz 的 `x`，不要擅自改成 `*`。
3. 如果想去除边框，还可以用 overrideredirect 方法，只 需要把参数设置为 1 即可。 4. 如果想修改标题栏的默认图标，可以用 iconbitmap 方法 或者是 wm_iconbitmap 方法，它接受一个 ico 文件的文件 名。

一些效果图

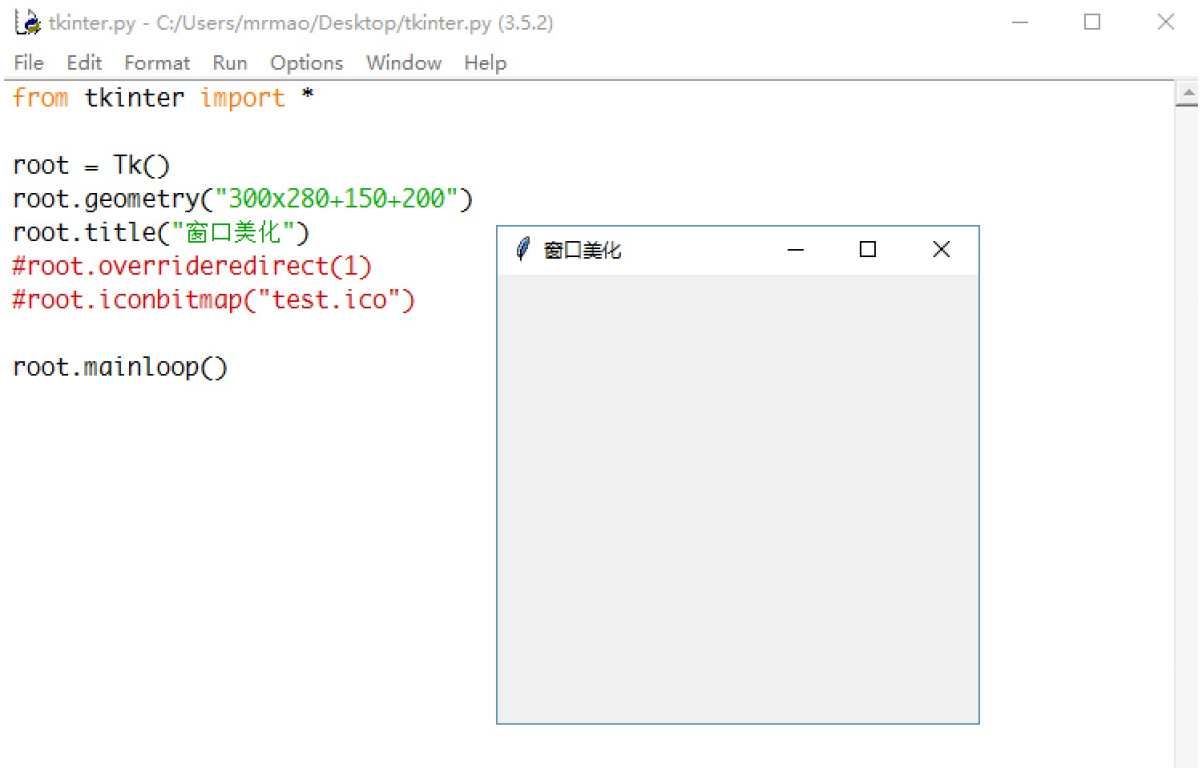
1. 更改图标之后的效果(由于我用gitbook做的课件，所以顺手用了gitbook的图标): **注意左上角的图标，不是红色的 tk 字样，而是新的的图片：**



2. 去除了边框之后的效果(感觉有点吓人，其实这种界面 方式很有名，用于制作异形窗口很有帮助，在此我就不深 入讲解了):



3. 最原始的窗口截图:



4. 代码示例：

```
from tkinter import *

root = Tk()
root.geometry("300x280+150+200")
root.title("窗口美化")
root.overrideredirect(1)
root.iconbitmap("test.ico")

root.mainloop()
```

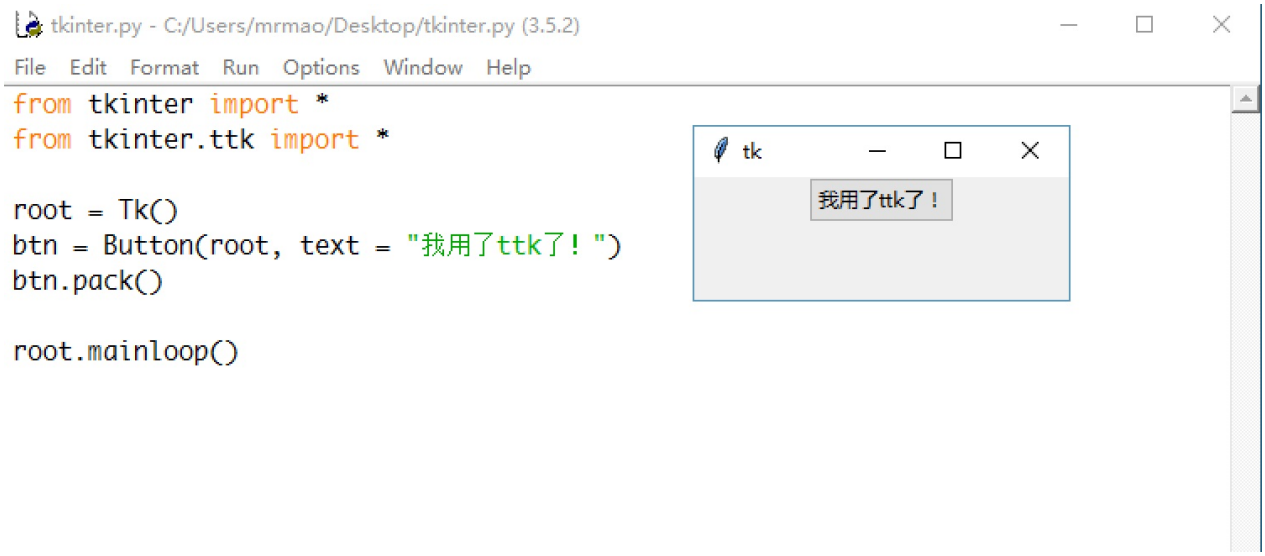
ttk 以及第一部分的总结

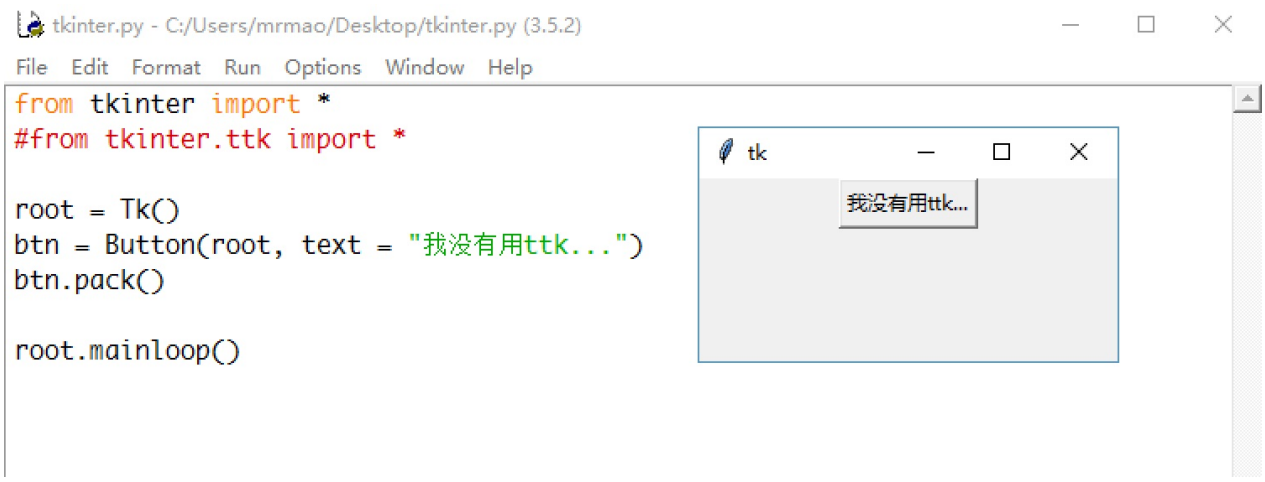
ttk 的产生

> - ttk 可以理解为“Tk tookit”，在 tkinter 的目录里可 以找到，它是最后一个文件，大小有 56KB，还算一个比较 大的文件的，毕竟我们的 tkinter 的**init**.py 也只有 158KB。
> - ttk 的出现和大家对 tkinter 的不满有关，因为 tkinter 是一个跨平台的界面库，但是，这也就意味着它丧失了平台优势。为了让它在 Windows 平台下运行更像 Windows，其实是让它支持不同的风格，于是出现了这么一个文件，用于弥补 tkinter 的不足。
> - 很重要的一点就是对标准组件的支持，ttk 支持原来的 21 个组件中的 11 个，并且，它引入了六个新的组件: combobox，notebook，progressbar，separator，sizegrip，tr eeview。

效果对比

1. 下面是同样的一个按钮，效果图如下(上面为导入了 ttk， 下面没有导入 ttk，我的平台是 Win10):





2. 使用 ttk 的代码截图:

```
from tkinter import *
from tkinter.ttk import *

root = Tk()
btn = Button(root, text = "我用了ttk了！ ")
btn.pack()

root.mainloop()
```

3. 不使用 ttf 的代码截图:

```
from tkinter import *

root = Tk()
btn = Button(root, text = "我没有用ttk...")
btn.pack()

root.mainloop()
```

其实大家也可以看到,其实很简单,使用 *ttf* 只需要在 前面加一行代码即可了“`*from tkinter.ttk import* `”即可。*

小球弹弹弹

0. 导入框架

```
from random import randint
from tkinter import *
from tkinter.ttk import *
```

1. 搭建框架

```
#获取屏幕参数，绑定相关事件，以及启动生成小球的迭代器
class MoreBalls:
    '''
        获取屏幕参数，绑定相关事件，以及启动生成小球的迭代器
        @ num: 从__main__里接受小球的数量
    '''
    # 定义一个列表，用来存储所有的球对象
    balls = []

    # num 是球的数量
    def __init__(self, num):

        # 创建一个Tk()窗口实例
        self.root = Tk()

# 1. w 和 h 分别获取了屏幕分辨率的宽度和高度
        scrnw = self.root.winfo_screenwidth()
        scrnh = self.root.winfo_screenheight()


        self.root.title("小球弹弹弹")
        # 去除窗口边框和任务栏显示
        self.root.overrideredirect(1)
        #self.root.iconbitmap("test.ico")
        # 设置窗口的透明度，0-1 之间，1是不透明，0是全透明。
        self.root.attributes("-alpha", 0.4)

        # 绑定退出事件（键盘任意键、鼠标任意点击、鼠标任意移动）
        self.root.bind("<Any-KeyPress>", self.myquit)
        self.root.bind("<Any-Button>", self.myquit)
        self.root.bind("<Motion>", self.myquit)
```

```
        # Canvas提供绘图功能(直线、椭圆、多边形等等)， 宽度和高度是屏幕分辨率大小
        self.canvas = Canvas(self.root, width = scrnw, height = scrnh)

        # 让画布按pack()布局
        self.canvas.pack()

        # 获取球的数量生成迭代器，每次迭代创建一个球
        for i in range(num):
            # ball是SettingBalls()类对象，传入self.canvas画布，以及屏幕的宽高
            ball = SettingBalls(self.canvas, scrnwidth = scrnw, scrnheight = scrnh)

            # 调用 创建球的方法
            ball.create_ball()

            # 将生成的球对象放到 balls 列表里
            self.balls.append(ball)

        # 调用 run_ball()方法，启动小球运动
        self.run_ball()

        # 调用mainloop() 消息循环机制
        self.root.mainloop()

    def run_ball(self):
        for ball in self.balls:
            ball.move_ball()
        # run_ball 每隔20毫秒会被调用一次
        self.canvas.after(20, self.run_ball)

    def myquit(self, event):
        self.root.destroy()
```

2. 设置小球

```
# 创建一个随机球处理类
class SettingBalls:
    def __init__(self, canvas, scrnwidth, scrnheight):
        # __init__函数里设置对象的属性

        # 对象自己的属性，接受canvas参数
        # Canvas是一个长方形的面积，图画或其他复杂的布局。可以放置在画布上的图形，文字，部件，或是帧
        self.canvas = canvas

        # tkinter绘图采用屏幕坐标系，原点在左上角，x从左往右 递增，y从上往下 递增
        # 在绘图区域内，随机产生当前 球的圆心 的 x坐标 和 y坐标，用于制定出现的位置
        self.xpos = randint(10, int(scrnwidth))
        self.ypos = randint(10, int(scrnheight))

        # 在绘图区域内，随机产生当前球的 x坐标 和 y坐标 的向量
        # 在数学中，几何向量（也称矢量），指具有大小和方向的量
        # 这里我们可以用来表示球的速度
        self.xvelocity = randint(6, 12)
        self.yvelocity = randint(6, 12)

        # 随机产生表示当前球的大小，也就是半径长度
        self.radius = randint(40, 70)

        # 通过lambda表达式创建函数对象r，每次调用 r() 都会产生 0 ~ 255之间的数字
        r = lambda : randint(0, 255)

        # 三次调用的数字取前两位，用十六进制数方式存储到 self.color里，作为球的颜色
        # #RRGGBB，前2是红色，中2是绿色，后2是蓝色，最小是0，最大是F，
        # 如 全黑#000000  全白#FFFFFF 全红#FF0000
        self.color = "#%02x%02x%02x" % (r(), r(), r())

        # 获取整个绘图场景的宽度和高度（也就是屏幕分辨率大小）
        self.scrnwidth = scrnwidth
        self.scrnheight = scrnheight
```

3. 上吧小球球！

```
if __name__ == "__main__":
    MoreBalls(14)
```

使用pyinstaller 打包我们的程序

> 升级我们的pip版本 `python -m pip install --upgrade pip`
>
> 通过pip安装 pyinstaller `pip install pyinstaller`
>
> 进入项目文件的文件夹，执行下面命令 `pyinstaller -w MoreBalls`