

## 2014 年度 辛星 Tkinter 教程 第一版

说明：该教程是第一版，较为浅显，大家可以搜索更新的版本来进行更深入的研究学习。

编辑时间：2014 年 6 月 2 日星期一（如果时间超过三年，请无视该教程）

建议：百度或者谷歌“辛星 python tkinter”可以得到更新更全面更系统的资料。

### 第1章 第0章：写在前面

第1章：走进 tkinter 的大门·····	2
第2章：标签与按钮·····	5
第3章：输入框·····	11
第4章：菜单·····	13
第5章：消息篇·····	18
第6章：完整的窗口应用程序·····	21
第7章：布局管理·····	24
第8章：总结·····	27

**前进的路上，辛星陪伴您。**

**只要星哥在，编程充满爱。**

## 第 0 章 写在前面（声明）

如果你学习过 c++或者 c#或者 vb 这样的编程语言，甚至是易语言，它们有不少类库是通过拖拽的方式来编辑界面，但是很可惜的是，我们的 python 目前还没有一个类库是可以拖拽来编写界面的，都是通过手写来完成，这就使得 python 虽然很简洁，但是用 python 写界面在一定程度上也会有所麻烦。

首先 Tkinter 是 Python 的界面编程的一种技术，它是一个标准库，是内置在 Python 标准版中的。Tkinter 原本是 Tcl 编程语言的图形库，后来开发了 Python 接口。Tkinter 是跨平台的用户界面开发工具，我并不认为 Tkinter 是必须掌握的，但是掌握一个 Python 的标准图形库还是一件非常有意思的事情。

我只是简单的介绍这个库能够做些什么事情，即只是介绍这个库包含什么，能干什么，至于如何灵活的使用该库，来做出简洁优秀的界面，则需要读者自行探讨或者搜索我写的更深入一部的教程了。

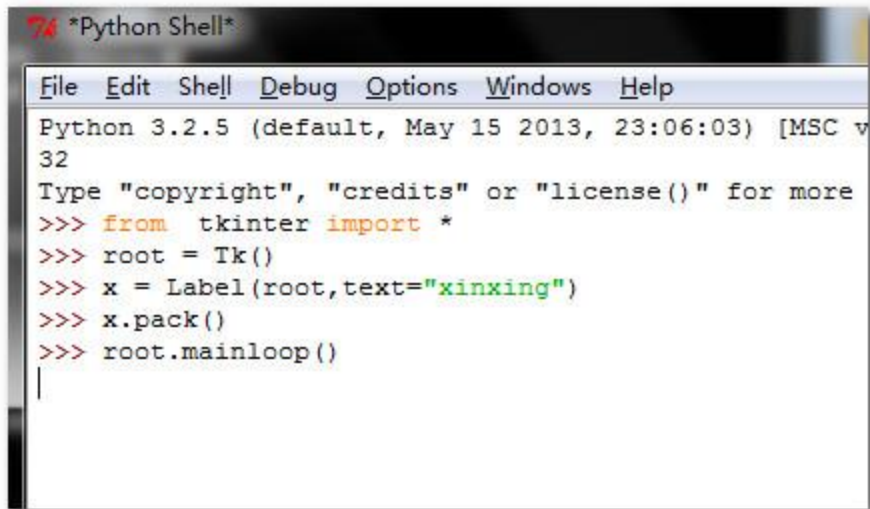
笔者这里使用的是 Python3.2，如果读者和我的版本不一样，可能会有稍微的语法变化或者什么的，但是应该没问题。下面的代码都是在 windows7 系统 Python3.2 下调试通过的。

## 第 1 章 走进 Tkinter 的大门

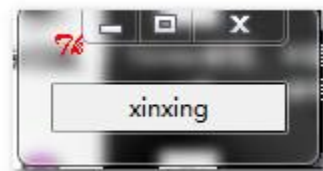
我们来开始我们的第一个程序来看一下效果，第一个程序并不会做什么事，只是简单的显示“xinxing”这个英文字母。下面是我使用 IDLE 在交互式模式下的代码示例。

## \*\*\*\*\*第一个程序\*\*\*\*\*

第一个是代码示例，下面是我得到的窗口。



```
*Python Shell*
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v
32
Type "copyright", "credits" or "license()" for more
>>> from tkinter import *
>>> root = Tk()
>>> x = Label(root, text="xinxing")
>>> x.pack()
>>> root.mainloop()
|
```



下面是写完最后一行所得到的窗口

下面开始解释一下这些代码，首先我们导入 `tkinter` 模块，这个模块包括了我們所需要的所有的类和函数等一些东西。这也是我们的 `from tkinter import *` 所做的工作。

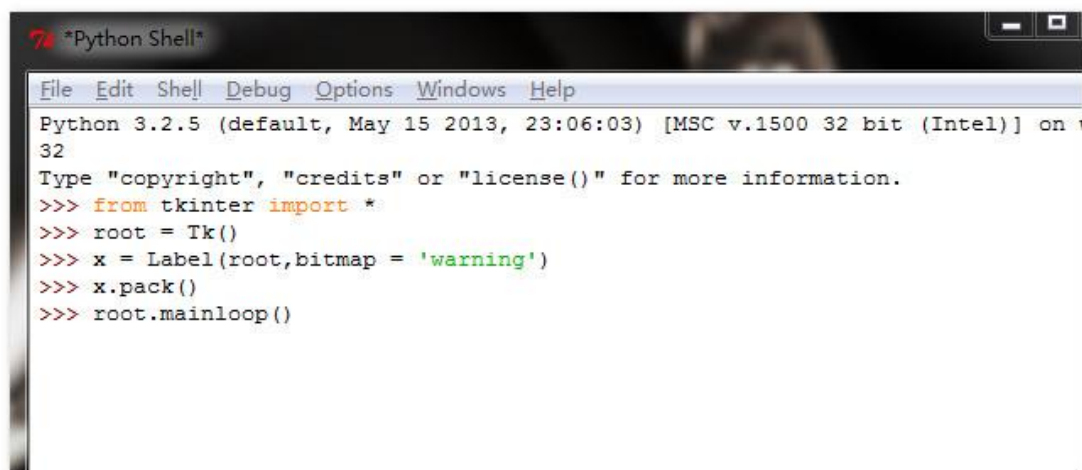
然后我们需要创建一个（根）窗口，它是一个普通的窗口，带有标题栏、最小化按钮、最大化按钮等。即我们的工作：`root = Tk()`

然后我们就可以在该窗口上创建一个 `Label`，即贴一个标签，这个 `Label` 可以显示文本、图标、图像等等。我们这里使用 `text` 来指定要显示的文本，而第一个参数 `root` 则指定了这个标签是属于谁的。

而接下来我们调用这个 `Label` 的 `pack` 方法，该方法是让 `Label` 去调整自己的大小，即我们的 `x.pack()`。

在下面的一行代码即 `root.mainloop()` 即可进入事件循环了，这样一个最简单的界面程序就完成了。

接下来我们用用 `Label` 来显示一个内置的图片“warning”。



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)] on
32
Type "copyright", "credits" or "license()" for more information.
>>> from tkinter import *
>>> root = Tk()
>>> x = Label(root, bitmap = 'warning')
>>> x.pack()
>>> root.mainloop()
```

下面是效果图。



我们这一节认识了一下 `tkinter` 的基本执行流程和很重要的一个“`label`”，它是一个标签，可以用来显示各种信息。

接下来说一下事件循环，不明白的读者可以看一下，比较清楚的可以直接跳过了。

Windows 操作系统是一个事件驱动的操作系统，即每当发生一个事件，比如鼠标点击，键盘按下等等，这些都可以作为一个消息被操作系统所接收，操作系统把这些消息传给应用程序，而我们再根据这些消息做出相应的响应。

比如在 FPS 射击游戏中，我们点一下鼠标左键，则操作系统接收该消息，然后把它传递给我们的程序，我们的程序会执行射出一颗子弹的步骤。关于事件，还有很多知识，我们下面会有专门的一节来说明。

当我们点击了应用程序右上角的关闭按钮的时候，该消息循环会接收到一个应用程序关闭的消息，消息循环结束，应用程序退出。

#### \*\*\*\*\*控件\*\*\*\*\*

最早接触“控件”我也不知道是在什么时候了，不过在 Windows 界面编程中它会频繁的出现。一个控件可以理解为一个嵌入在其他应用程序中的小型应用程序，它并不独立起作用，它可以和它嵌入的父应用程序通过交互起作用。

我们上面提到的 Label 就是一个控件，它通常被翻译为“标签”，通常用来显示信息，比如显示文字，图片，图标等等。关于 Label，下面还会介绍。

一开始的控件很少，而且大多数是微软定义的，比如标签，按钮，单选框，复选框，下拉列表，菜单，滚动条，文本框，密码框，滑块等等，但是随着不断的开发，控件也逐渐复杂起来，比如日历控件，拖动控件，对话框控件，文件控件，颜色控件等很多第三方控件也很多，后来一些超链接控件，高级超链接控件，而且很多人开发的第三方控件也逐渐被人所熟知，很多叫不上名字的控件，总之我们只需要了解控件的基本原理和使用，并且自己能开发控件就 Ok 了。

## 第 2 章 标签与按钮

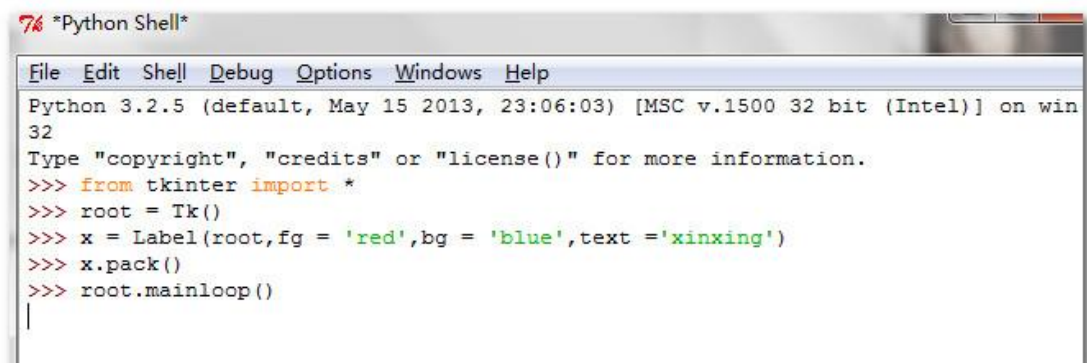
控件的种类如此的多，我们不需要全部知道，但是有些还是太重要了，我认为我们有必要认识他们。

\*\*\*\*\*标签\*\*\*\*\*

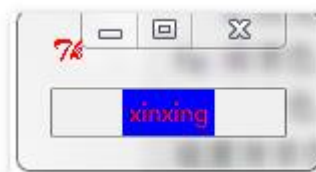
几乎所有的界面编程都从标签开始，因为它太简单了，它只是用来显示一些信息，主要是文字信息和图片信息。

通过第一节我们知道了 Label，下面说说如何美化标签以及干什么。

- 可以用 fg (foreground)，bg (background) 来调整前景色和背景色。示例代码如下：

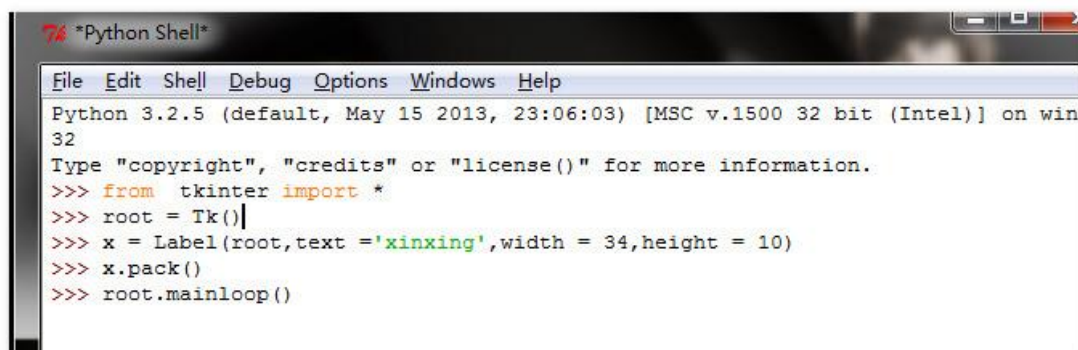


```
*Python Shell*
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> from tkinter import *
>>> root = Tk()
>>> x = Label(root,fg = 'red',bg = 'blue',text = 'xinxing')
>>> x.pack()
>>> root.mainloop()
|
```



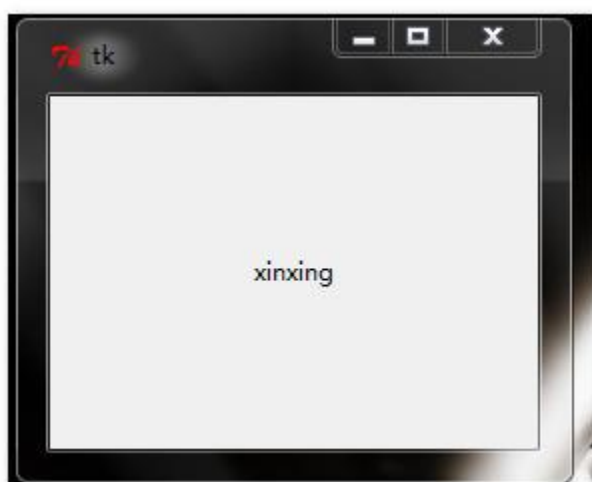
下面是执行效果：

- 可以用 width 和 height 来调整该标签的宽和高。示例代码：



```
*Python Shell*
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> from tkinter import *
>>> root = Tk()
>>> x = Label(root, text='xinxing', width = 34, height = 10)
>>> x.pack()
>>> root.mainloop()
```

示例效果：




- 其他可以调的属性还是蛮多的，这里不一一列举了，如果想要做出很优美且简洁的界面，确实需要仔细的研究他们。

\*\*\*\*\*按钮\*\*\*\*\*

按钮即“Button”，它也是一个标准的 tkinter 窗口部件，用来实现按钮。按钮可以包含文本或者图像，当我们点击该按钮的时候，操作系统把这一信息传递给我们的应用程序，然后我们的应用程序有必要做出相应的反应，我们往往把一个函数或者方法与该按钮相关联。当按钮被按下的时候，tkinter 就会自动调用相关联的函数或者方法，即一个简单的“回调函数”。

按钮重要就重要在它是最简单的可以关联一个函数的控件，这个控件可以让我们执行一些用户的命令，因此它尤其重要。按钮通常是关联一个函数的，当用户点击该按钮的时候，该函数会自动执行，该函数在 `command` 属性中设置。

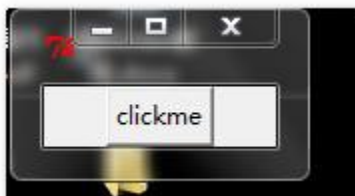
看下面的代码：



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> from tkinter import *
>>> def xin():
>>>     print('xinguimeng')

>>> root = Tk()
>>> #我们通过指定command属性来让该按钮知道它该调用哪个函数
>>> x = Button(root, text = 'clickme', command = xin)
>>> x.pack()
>>> root.mainloop()
```

下面是它的界面：



当我们点击那个“clickme”按钮的时候，会发现这时候的命令行会多出来一行“xinguimeng”，效果如下：



A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The text area shows the following code: 

```
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (
32
Type "copyright", "credits" or "license()" for more information.
>>> from tkinter import *
>>> def xin():
    print('xinguimeng')

>>> root = Tk()
>>> #我们通过指定command属性来让该按钮知道它该调用哪个函数
>>> x = Button(root, text = 'clickme', command = xin)
>>> x.pack()
>>> root.mainloop()
xinguimeng
```

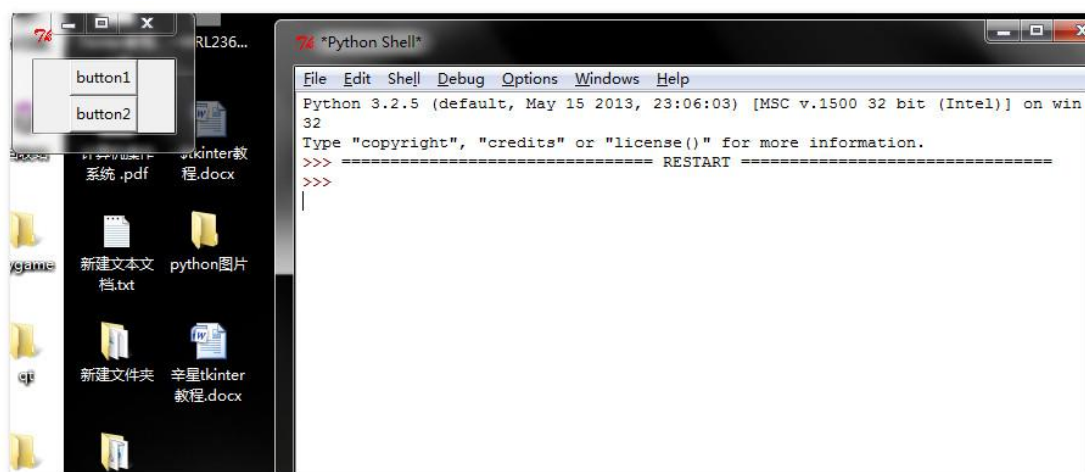
我们还可以用按钮的 `bind` 方法把相应的按钮和函数关联起来，`bind` 翻译为汉语的意思是“绑定”，而且调用的函数可以带有参数。

下面是代码：

A screenshot of a Python IDE window titled "my.py - C:/Python32/新建文件夹/my.py". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Windows", and "Help". The text area shows the following code: 

```
from tkinter import *
def x1():
    print("xinguimeng")
def x2(event):
    print(event.time, event.type)
root = Tk()
b1 = Button(root, text = "button1", command = x1)
b1.pack()
b2 = Button(root, text = "button2")
b2.bind("<Enter>", x2)
b2.pack()
root.mainloop()
```

下面是开始执行的时候的效果



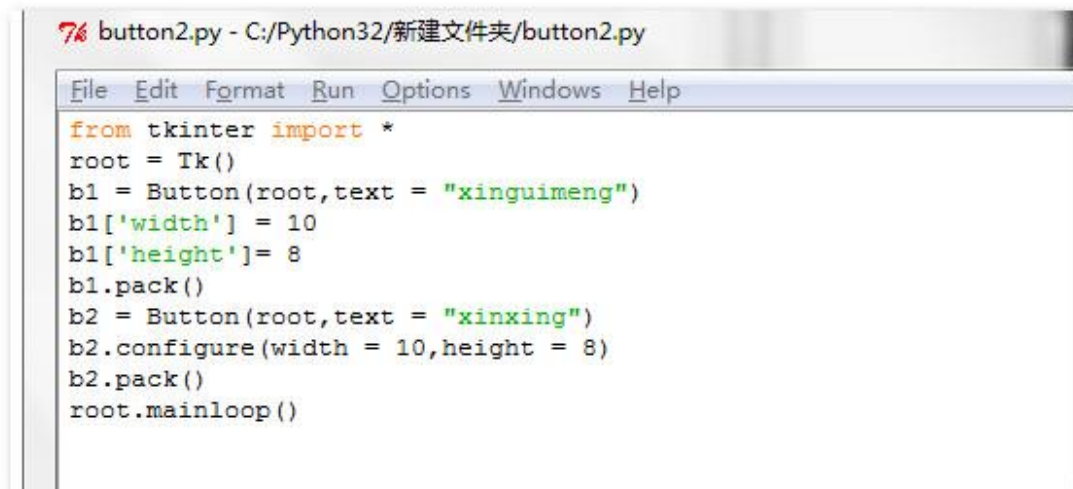
当我们的鼠标进入 button2 的上方的时候，命令行会显示相关信息，当我们点击 button1 的时候，命令行会打印“xinguimeng”这个字母组合，下面是执行效果：



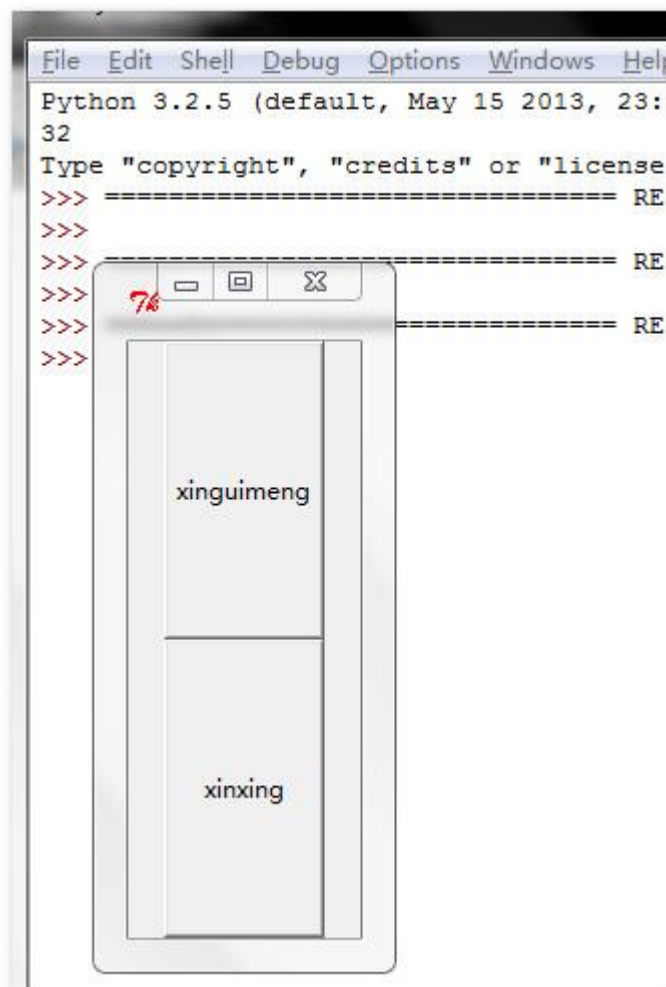
这里的 event 涉及到事件，我们下一节将会详细讲解。而且按钮默认的是单击之后执行相关的函数。

按钮的属性也很多，如果大家有面向对象的基础，那么我们可以在这里说一下按钮的常规属性与方法。

下面的代码中第一个是直接设置属性，第二个则是使用了一个方法来设置宽和高，效果是一样的，代码如下：



```
7% button2.py - C:/Python32/新建文件夹/button2.py
File Edit Format Run Options Windows Help
from tkinter import *
root = Tk()
b1 = Button(root, text = "xinguimeng")
b1['width'] = 10
b1['height'] = 8
b1.pack()
b2 = Button(root, text = "xinxing")
b2.configure(width = 10, height = 8)
b2.pack()
root.mainloop()
```



运行效果：

- 当然还可以用 `bg` 和 `fg` 来设置背景色和前景色，还可以用 `bd` 来设置边框的像素数。

- 还可以设置按钮的状态，比如按钮有时候可以“灰化”，即变得不可点击，更有甚者直接消失了。那么可以设置它的 `state` 属性，通常可以取值为 `normal`, `active`, `disabled`。比如 `b2` 为一个按钮，那么把它改变成不可用状态则使用 `b2[ 'state' ] = “disabled”` 即可，注意要加上引号奥。
- 还可以修改按钮上的文字，使用其 `text` 属性，比如 `b` 是一个按钮，则使用 `b[ 'text' ] = “xinxing”` 则把 `b` 的文字改成了“xinxing”。

### 第3章 输入框

由于输入框也是如此的重要，因为它的存在，才使得我们的程序能够接受大量的用户想要表达的信息，如果我们的用户只能够点点鼠标，那么信息量是及其有限的，连一个最简单的用户登录都无法实现。

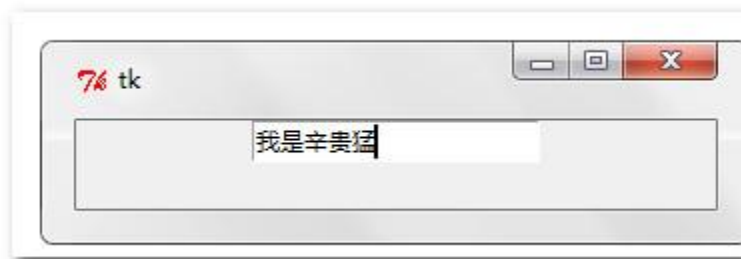
\*\*\*\*\*最简单的输入框\*\*\*\*\*

我们用 `Entry` 来表示输入框，代码如下：



```
entry - C:/Python32/新建文件夹/entry
File Edit Format Run Options Windows Help
from tkinter import *
root = Tk()
Entry(root, text = "input your name").pack()
root.mainloop()
|
```

运行效果（比如我在文本框里输入我的姓名）



\*\*\*\*\*使用输入框的数据\*\*\*\*\*

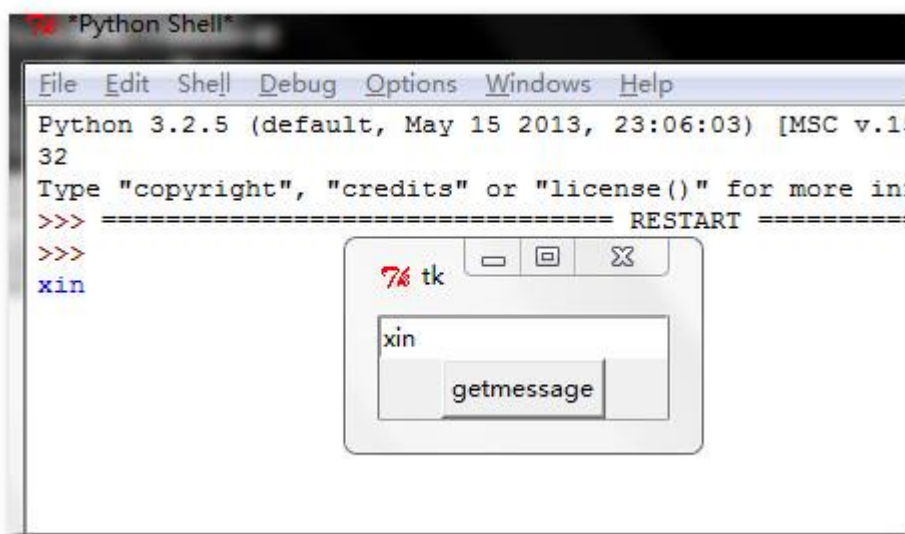
输入框的数据只有被使用才有价值，否则就没必要输入了。这里我们编写一个稍微复杂一点点的程序，我们单击按钮，来输出输入框的内容。代码如下：

```
entry - C:/Python32/新建文件夹/entry
File Edit Format Run Options Windows Help
from tkinter import *
def get():
    x = e.get()
    print(x)
root = Tk()
e = Entry(root)
e.pack()

b = Button(root, text = "getmessage", command = get)
b.pack()

root.mainloop()
```

运行效果(注意后台的那个 xin 的输出)。



\*\*\*\*\*作为密码框来用\*\*\*\*\*

密码框和普通的文本输入框最大的不同在于密码框的显示很单一，我们通过屏幕得不到有用的信息。

其实只需要修改一个 show 属性即可，如 en 是一个输入框，则可以写如下代码：`en[ 'show' ] = “*”`

\*\*\*\*\*其他函数\*\*\*\*\*

关于输入框，我们有时候会判断用户的输入的一些细节，比如 QQ 号的位数不能过短或过长，邮箱一定要有一个@符号等。

系统也自带了一些有用的函数，当然加上正则表达式，功能就更加强大了。

## 第4章 菜单

菜单的重要性不言而喻，菜单的作用通常就是左键点击之后可以执行相应的命令，当然还有弹出式菜单、子菜单、菜单的快捷键、菜单的主动更新、菜单的状态、菜单分隔符等等若干内容。



不过我不计划写那么全面，因为在拖拽式编程里面建一个菜单实在是太方便了，但是在 tkinter 里却比较原始，大家也不要嫌麻烦，实在有点落后。

#### \*\*\*\*\*主菜单\*\*\*\*\*

首先就是一个窗体，或者说一个窗口是有菜单这个属性的，我们只需要设置一个窗体的 menu 属性为一个菜单就可以了，新建一个菜单的实例是 Menu 方法，为主菜单中添加菜单用 add\_command 方法，而且菜单也往往需要关联相应的函数，而每一个菜单项的 command 属性就是该函数的函数名。

下面是实例代码，相信结合如下备注，大家可以看得比较清楚：

A screenshot of a Python IDE window titled 'menu.py - C:/Python32/新建文件夹/menu.py'. The window contains the following Python code:

```
from tkinter import *
root = Tk()
def xin():
    print("I am XinGuiMeng")

xmenu = Menu(root) #需要制定 该菜单是属于谁的

for item in ['java','cpp','c','php']:
    xmenu.add_command(label = item,command = xin)
root["menu"] = xmenu
root.mainloop()
```



效果如图：

每当我们点击其中的一个菜单，则会在命令行打印一行“xinguimeng”这个字符串。下面是我任意点击了几个按钮之后的截图：

```
>>>
>>> ===== RESTART =====
>>>
I am XinGuiMeng
I am XinGuiMeng
I am XinGuiMeng
I am XinGuiMeng
I am XinGuiMeng
|
```

```
>>>
>>> ===== RESTART =====
>>>
I am XinGuiMeng
I am XinGuiMeng
I am XinGuiMeng
I am XinGuiMeng
I am XinGuiMeng
|
```

\*\*\*\*\*下拉菜单\*\*\*\*\*



下拉菜单也非常的常见，如果一个菜单没有下拉菜单，那么它和按钮的区别就很小了，正是因为有了下拉菜单，才使得菜单的信息量如此之大。

子菜单的代码示例：

```
*cascademenu.py - C:/Python32/新建文件夹/cascademenu.py*
File Edit Format Run Options Windows Help
from tkinter import *
def xin():
    print("I am XinGuiMeng")
root = Tk()
menubar = Menu(root) #把这一组菜单变成下拉菜单的项
xmenu = Menu(menubar, tearoff = 0)
for item in ['xin', 'gui', 'meng']:
    menubar.add_command(label = item, command = xin)
for item in ['java', 'cpp', 'c', 'php']:
    xmenu.add_command(label = item, command = xin)
menubar.add_cascade(label = "progame", menu = xmenu) #把xmenu变成progame的子菜单
root["menu"] = menubar
root.mainloop()
```

子菜单的运行效果：



\*\*\*\*\*弹出菜单\*\*\*\*\*


弹出菜单一般也称之为上下文菜单，即点击右键之后弹出的菜单，通常也称为右键菜单。弹出菜单可能会因

为右键所在的位置的不同而弹出不同的菜单，因此弹出菜单和鼠标的位置息息相关。

弹出式菜单的一般思路就是我们先建立一组菜单，然后响应鼠标右键的消息，每当用户点击右键的时候，我们就让这一组菜单显示出来。

这里又用到了消息机制，可能会导致些许的迷惑，但是代码应该大致能读懂。

具体用到的函数和代码示例如下：



```
7% popmenu - C:/Python32/新建文件夹/popmenu
File Edit Format Run Options Windows Help
from tkinter import *
root = Tk()
menubar = Menu(root)

def xin():
    print("I am XinGuiMeng")

for x in ['python','c','java','php']:
    menubar.add_command(label = x,command = xin)

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

root.bind("<Button-3>",pop)
root.mainloop()
```

弹出菜单效果图：



#### \*\*\*\*\*其他操作\*\*\*\*\*

说起来关于菜单的操作，很多都很重要，比如菜单的动态创建，菜单的删除和插入，而且菜单的操作也及其的灵活。

这里我不是很想写下去的原因就是其他编程语言在这一点上做的实在是太出色了，大家也可以去参考该类的具体函数，应该很好理解。

## 第5章 消息篇

写了这么久，终于可以写消息了。在任何一个界面编程里面，都不可能不讲消息机制，当然，比如 Qt 的界面里面会有自己的“信号槽”技术，MFC 有自己的消息映射机制，对于消息的处理，是至关重要的。

#### \*\*\*\*\*消息的应用\*\*\*\*\*

Windows 操作系统是基于消息的，一个消息就是一个结构体，当然不同的消息类型也会包含不一样的数据，如果是鼠标的按键消息，则会记录按键的时间，按键的时候的坐标等等。

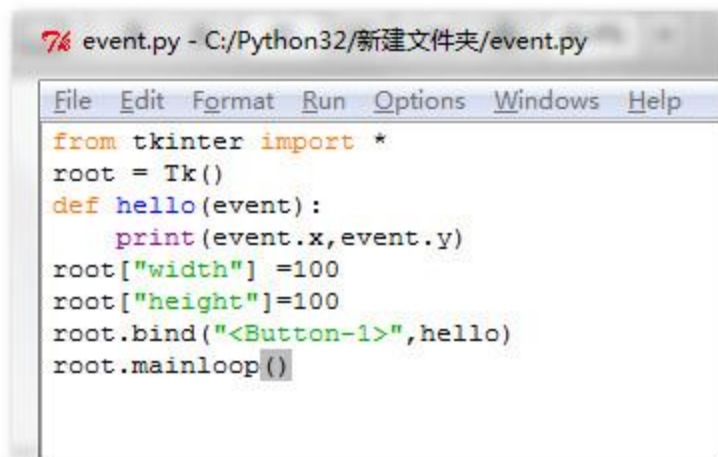
消息是由操作系统来捕获的，然后操作系统会自动把消息发送给我们，我们只需要处理它们就可以了。当然，tkinter 和 windows 都会帮助我们处理一些基本的消息，比如窗口的拖动，缩放这些最基本的操作，不用我们去做。

我们做的应用程序的大致流程就是：用户产生某些动作（比如单击鼠标）→操作系统捕获并且传递给我们的应用程序→我们的应用程序去处理它并且继续等待其他消息。

Windows 的系统消息有很多，大约有二百多个，但是我们经常关注的并没有那么多。更多的是围绕着鼠标和键盘消息。

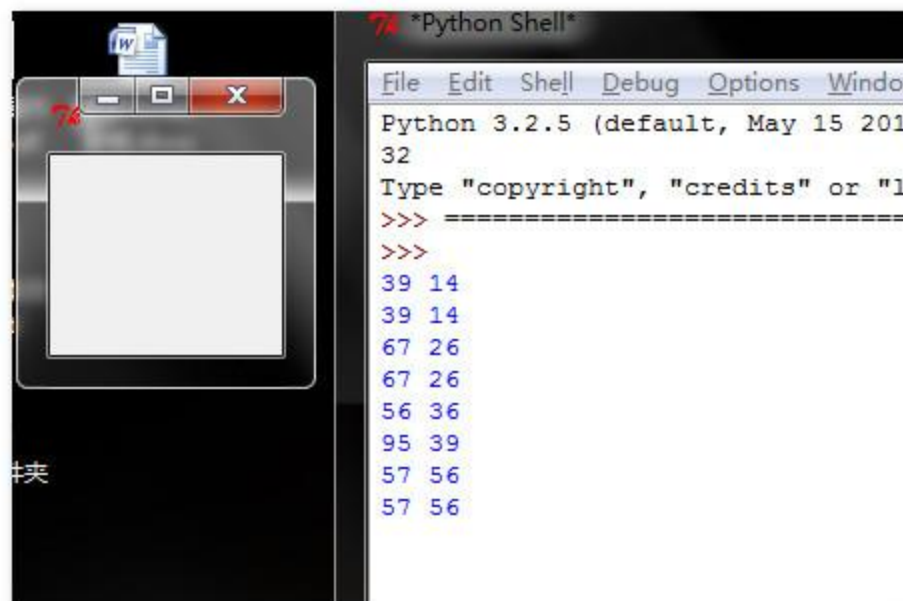
我们看一下在 python 中自己的消息机制，对于任一个窗口部件 widget，都可以为它绑定一个函数或者方法：widget.bind(event, handler) 这里的 event 即相应的事件类型，handler 即相应的消息处理函数，它的调用时自动的，我们无需操心，即只要 event 事件发生的时候，handler 函数会被自动调用，即一个回调函数。

这里列举一个当鼠标左键按下的时候，我们使用 hello 函数来记录该事件的 x 坐标和 y 坐标。

A screenshot of a Python IDE window titled 'event.py - C:/Python32/新建文件夹/event.py'. The window contains a Tkinter script that creates a root window, sets its width and height to 100, binds a left mouse button click event to a function named 'hello', and starts the main loop. The 'hello' function prints the event's x and y coordinates.

```
from tkinter import *
root = Tk()
def hello(event):
    print(event.x, event.y)
root["width"] = 100
root["height"] = 100
root.bind("<Button-1>", hello)
root.mainloop()
```

运行效果如下：



### \*\*\*\*\*常用的事件\*\*\*\*\*

常见的事件如下：

- <Button-1>事件：它是鼠标左键在某窗口部件上左键按下。
- <Button-2>事件：它表示鼠标中键按下。
- <Button-3>事件：它表示鼠标右键按下。
- <Button-Motion>事件：它表示鼠标左键被按住并且移动鼠标，即“拖动鼠标”。如果 1 改成 2 则代表按住的为中间键，1 改成 3 则表示按住的为右键。
- <ButtonRelease-1>事件：鼠标左键释放。当 1 改成 2 的时候表示中键被释放，1 改成 3 表示鼠标右键被释放。
- <Double-Button-1>事件：鼠标左键双击。数字改成 2 或者 3 表示中键和右键。

- `<Enter>`事件。鼠标指针进入某个窗口部件的范围，它只是当鼠标进入该范围即可，用户按下回车键并不起作用。注意与下面的`<Return>`区分。
- `<Leave>`事件。鼠标指针从该窗口部分离开。
- `<Return>`事件。用于按下 Enter 键。
- `<Key>`事件，我们可以从 event 对象中的 char 成员属性中得到按下的是哪个键。
- `<a>`事件：即字母 a 输入，大部分字母都可以替换该字母，比如 bcdefg...z 等等。

其他事件不一一列举了。

#### \*\*\*\*\*事件的属性\*\*\*\*\*

当我们写一个消息处理函数的时候，通常用一个 event 来作为参数，接下来可以用 event.x\_root 表示鼠标相对于屏幕左上方的位置，event.y\_root 表示鼠标相对于屏幕左上方的位置，event.x 表示鼠标指针相对于本窗口的横坐标，event.y 表示鼠标相对于本窗口的纵坐标，event.type 表示事件类型。

当事件类型为键盘消息的时候，还可以使用 char 来得到按键字符，用 keycode 来得到键的代码。

到此，我们为我们的界面应用程序加上消息处理，与用户的交互性进一步增强。

## 第6章 完整的窗口应用程序

考虑一个完整的窗口应用程序都需要哪些内容，那么会发现我们学习的内容已经差不多了。当然，我们还

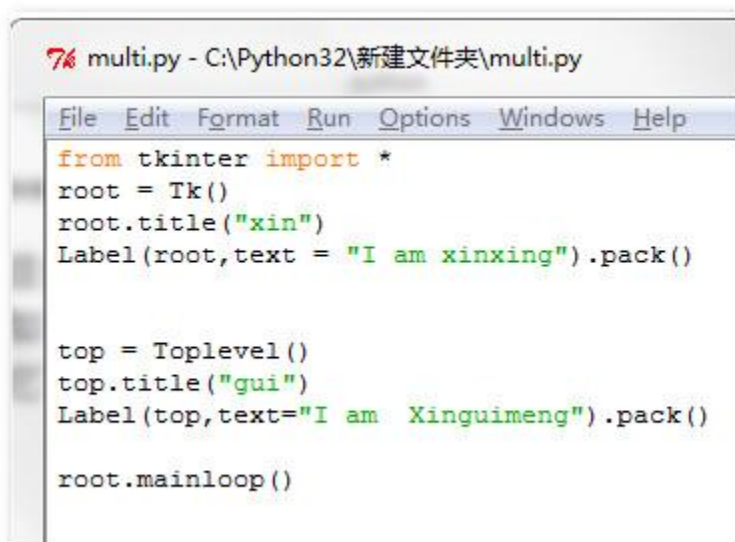
有很多控件没有学习，我并不计划在第一版中详细介绍那些啰嗦的控件。

#### \*\*\*\*\*多窗口应用程序\*\*\*\*\*

我们目前的窗口应用程序只有一个窗口，显然这是不合适的，如果要创建多个窗口，可以使用 `Toplevel()` 来创建，而且它不需要用 `pack()` 来显示。

下面的示例的 `root` 与 `top` 的区别在于，`root` 是根窗口，只有这一个，而 `top` 是顶层窗口，可以有很多这样的顶层窗口，但是根窗口只能有一个。

代码示例如下：



```
74 multi.py - C:\Python32\新建文件夹\multi.py
File Edit Format Run Options Windows Help
from tkinter import *
root = Tk()
root.title("xin")
Label(root, text = "I am xinxing").pack()

top = Toplevel()
top.title("gui")
Label(top, text="I am Xinguimeng").pack()

root.mainloop()
```

效果图如下：





### \*\*\*\*\*窗体\*\*\*\*\*

在学习其他的界面编程的时候，frame 这个单词也会经常碰到。简单来说，一个 frame 可以理解为一个矩形区域，是一个用来作为容器来布局窗体，即可以理解为一个“框架”，一个“窗体”。

接下来的示例给大家演示一下创建了三个 Frame 的代码示例：



```
76 frame.py - C:/Python32/新建文件夹/frame.py
File Edit Format Run Options Windows Help
from tkinter import *
root = Tk()
for x in ['red', 'blue', 'yellow']:
    Frame(height = 20,width = 20,bg = x).pack()
root.mainloop()
|
```

矩形效果：



### \*\*\*\*\*其他控件\*\*\*\*\*

控件还是非常重要的，但是无论是讲解控件还是学习控件的使用，都是一个非常令人头疼的话题，学习就像背字典，讲解也像是在编字典。

作为一个字典，我们在需要的时候查阅就可以了，不必要把它们都记住。但是，随着使用次数的增加，我们是可以把它们记住的，并且，会很熟悉。



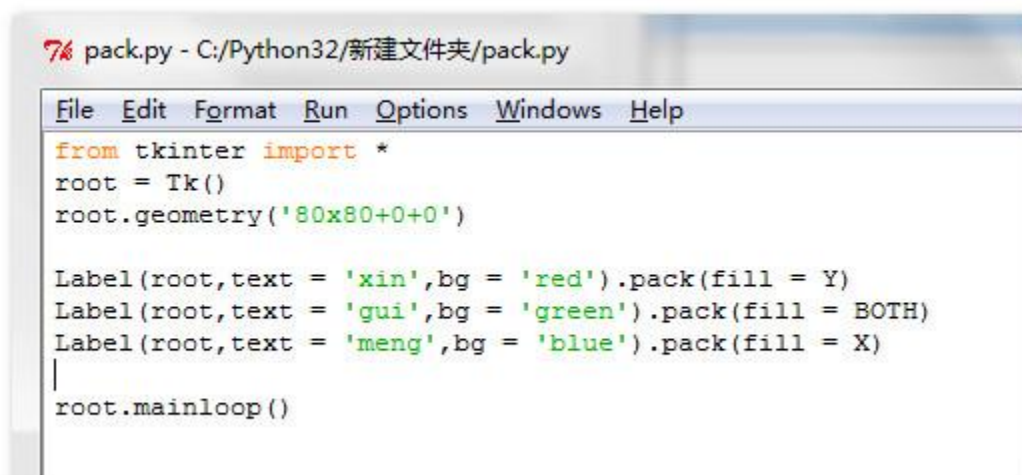
## 第7章 布局管理

界面的布局是很重要的，即使我们目前接触到的界面的元素并不多，但是了解界面的布局依然是非常重要的一点，一个清爽简介的界面没有一个良好的布局是难以想象的。

\*\*\*\*\*pack\*\*\*\*\*

前面我们使用过不少 pack 函数了，我们可以把它理解为一个弹性的容器。我们可以使用 pack 函数按照系统默认的方式把我们的东西加到界面上去。

使用默认的设置 pack 会依次向下添加组件，而且 pack 可以使用 fill 来说明填充的方向。比如看如下示例：



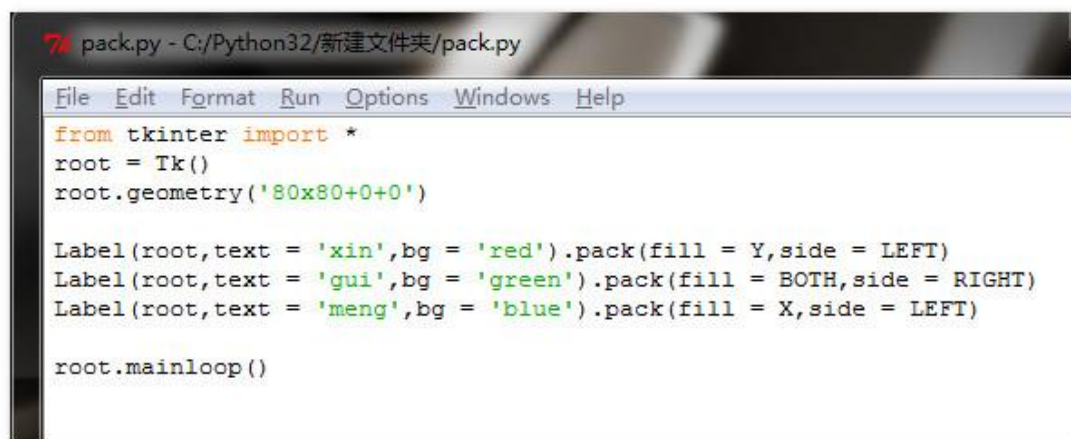
```
74 pack.py - C:/Python32/新建文件夹/pack.py
File Edit Format Run Options Windows Help
from tkinter import *
root = Tk()
root.geometry('80x80+0+0')

Label(root,text = 'xin',bg = 'red').pack(fill = Y)
Label(root,text = 'gui',bg = 'green').pack(fill = BOTH)
Label(root,text = 'meng',bg = 'blue').pack(fill = X)
|
root.mainloop()
```

下面是效果图（由于第一个使用了在 Y 方向填充，第二个使用了在 X 和 Y 方向填充，而第三个则使用在 X 方向填充，所以大家可以拖动边框来观察这三个的区别）：



pack 这种布局是非常简单的，它也可以用于左右布局，比如这里让第一部分在左边，第二部分在右边，第三部分继续在左边。比如如下的示例代码：

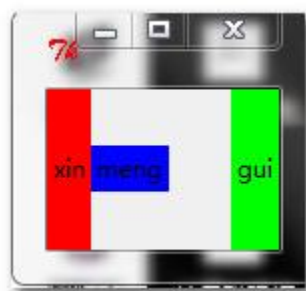


```
pack.py - C:/Python32/新建文件夹/pack.py
File Edit Format Run Options Windows Help
from tkinter import *
root = Tk()
root.geometry('80x80+0+0')

Label(root,text = 'xin',bg = 'red').pack(fill = Y,side = LEFT)
Label(root,text = 'gui',bg = 'green').pack(fill = BOTH,side = RIGHT)
Label(root,text = 'meng',bg = 'blue').pack(fill = X,side = LEFT)

root.mainloop()
```

下面是效果图：




当然关于 pack 还有其他参数，但是它的控制能力还是过于弱，所以不多介绍了。

#### \*\*\*\*\*固定布局\*\*\*\*\*

所谓固定布局就是直接把控件的坐标写到代码中去，或者写到配置文件中去读取，不管怎么说，这种布局最大的问题就是界面被缩放后可能导致不美观。

而固定布局主要通过 place 函数来实现，它还分为相对坐标和绝对坐标布局。

因为我不怎么建议使用，所以只是举一个例子，就可以了，代码和示例如下：



```
demo - C:/Python32/新建文件夹/demo
File Edit Format Run Options Windows Help
from tkinter import *
root = Tk()
x = Label(root, text = "xinguimeng")
x.place(x = 3, y = 3, anchor = NW)
root.mainloop()
```

运行效果:

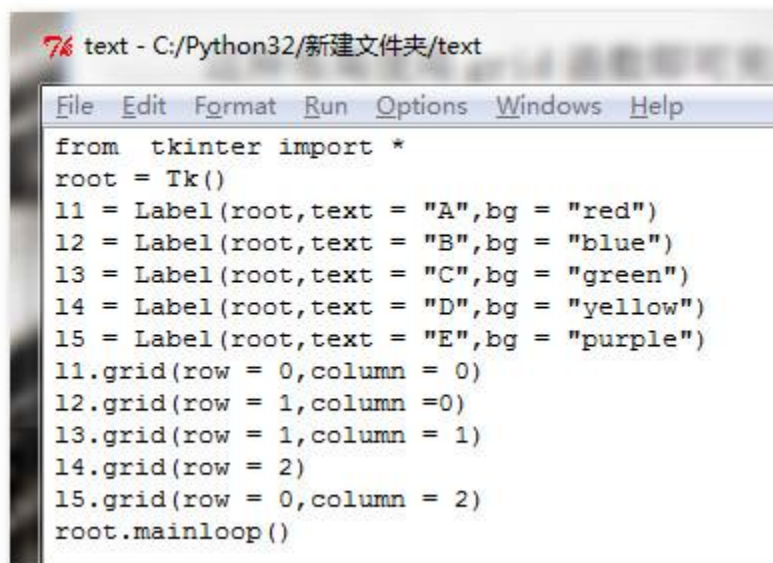


#### \*\*\*\*\*网格布局\*\*\*\*\*

Grid 布局就是我们所说的网格布局，这种布局的实现机制是把界面分割成表格，在指定的位置放置想要的 widget 就可以了。

这种布局使用 grid 函数即可完成，它有两个重要的参数，一个是 row，用于指定行数，另一个是 column，用于指定列数。如果不指定 row，则会将组件放置到第一个可用的行上，如果不指定 column，则会使用第一列，注意这里

的 grid 不需要先创建，直接用就可以。

A screenshot of a Python IDE window titled '76 text - C:/Python32/新建文件夹/text'. The window contains a Tkinter script that creates a root window and places five labels (A, B, C, D, E) using the grid layout manager. The labels are placed in a 3x2 grid: A (red) at (0,0), B (blue) at (1,0), C (green) at (1,1), D (yellow) at (2,0), and E (purple) at (0,2). The script ends with root.mainloop().

```
from tkinter import *
root = Tk()
l1 = Label(root, text = "A", bg = "red")
l2 = Label(root, text = "B", bg = "blue")
l3 = Label(root, text = "C", bg = "green")
l4 = Label(root, text = "D", bg = "yellow")
l5 = Label(root, text = "E", bg = "purple")
l1.grid(row = 0, column = 0)
l2.grid(row = 1, column = 0)
l3.grid(row = 1, column = 1)
l4.grid(row = 2, column = 0)
l5.grid(row = 0, column = 2)
root.mainloop()
```

下面是网格布局效果图：



## 第8章 总结

可能你认为看完本文之后没有达到预期的效果，但是本文却为大家简要的预览了 tkinter 的框架。

可能你会说我有太多的控件没有涉及到，很多重要的函数也没有说，但是作为 tkinter 的入门回修改了好几次才确定，它确实非常值得推荐的，也是我来下来的。

有些东西我不置可否，也不知道大家是否关心，虽然它们在某些时候很重要。比如字体，比如对话框，如果大家感觉很重要，我希望会在第二版中加入它们。

总的来说，我很欣赏 python 的语法和风格，但是用 python 去做界面真的不是它的长处，而且作为一门解释语言，在这方面速度还是比较慢的。