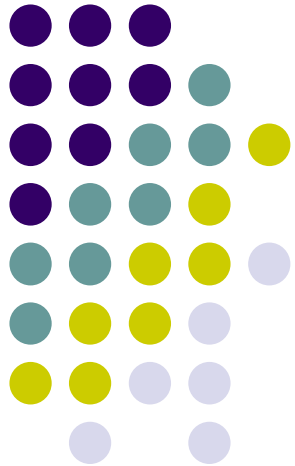
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

第8章 图形用户界面

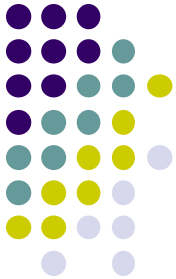
Graphic User Interface



申丽萍

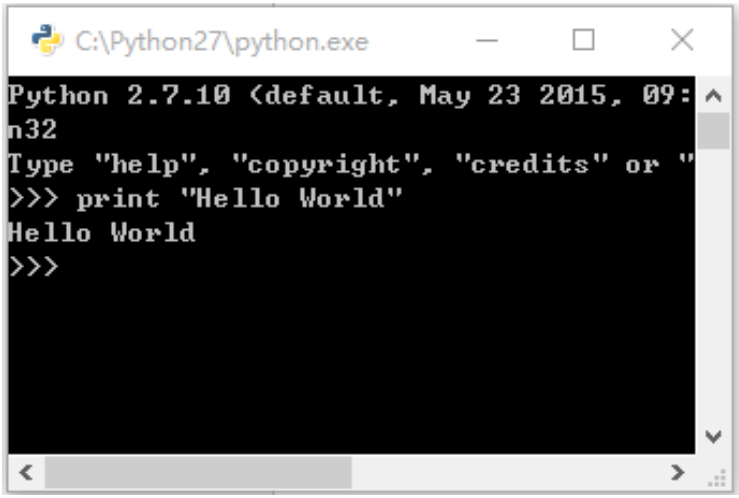
lpshen@sjtu.edu.cn

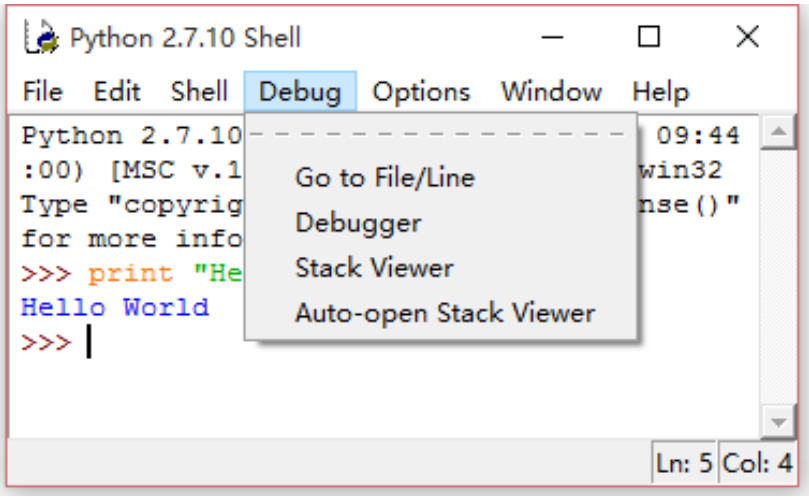
第8章 图形用户界面GUI



- GUI基本概念
- Tkinter编程
- 事件驱动编程
- MVC设计方法

两种用户界面



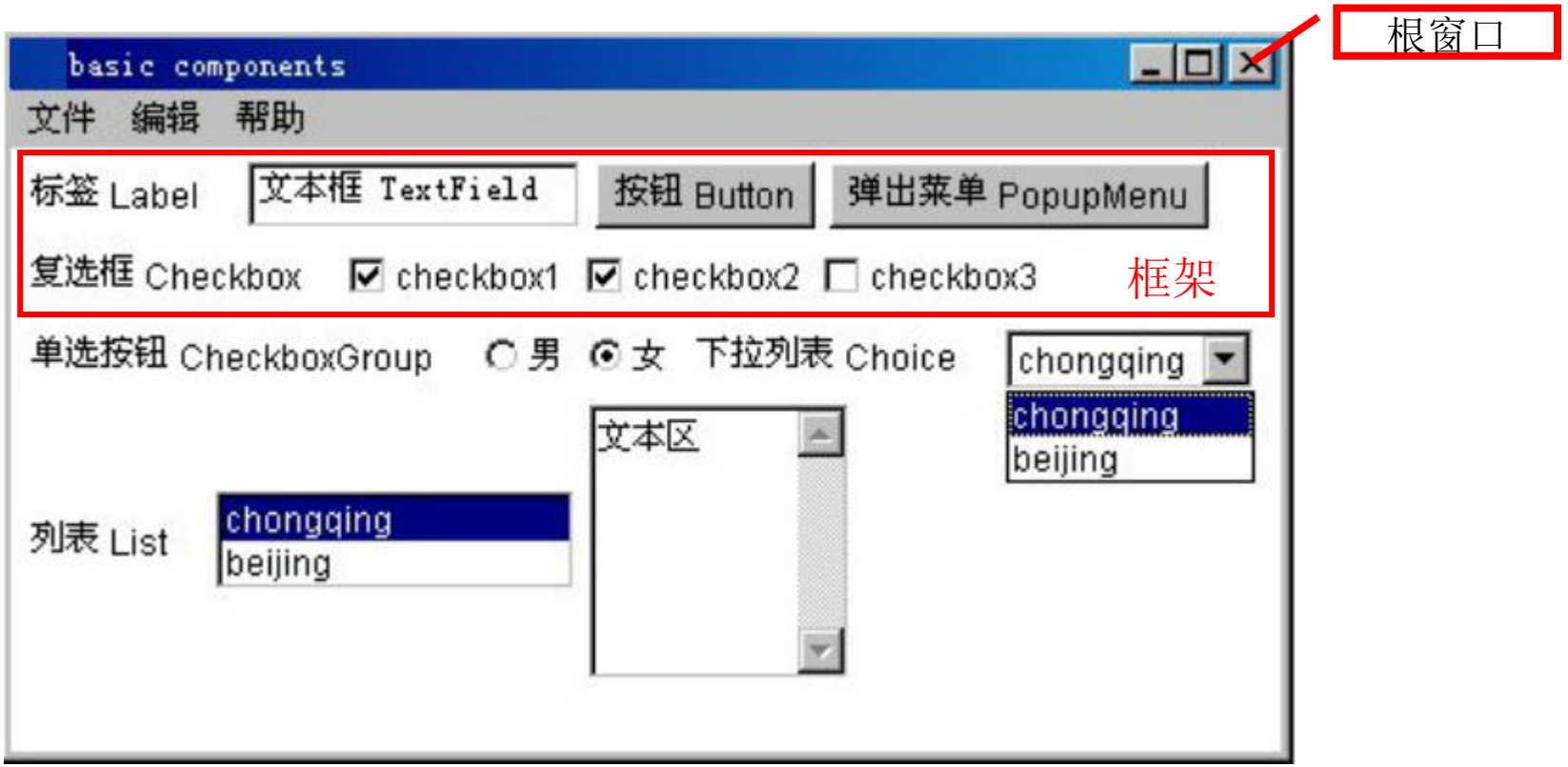


命令行界面（Command Line Interface)
---输入命令，输出文本

图形用户界面（Graphic User Interface)
---菜单/控件输入，输出文本/图形

GUI组成

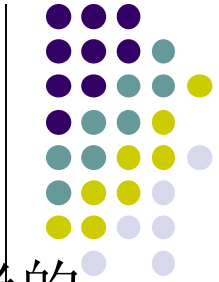
- 图形用户界面GUI由多个构件（widget, control) 组成。
- 多个构件可以布局在一个容器内。
- 容器可以是窗口（window)或框架(frame)。
- 构件的位置编排由布局管理器(layout manager)进行管理。



第8章 图形用户界面GUI

- GUI基本概念
- Tkinter编程
- 事件驱动编程
- MVC设计方法

Tkinter模块



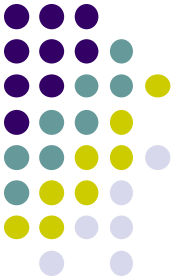
- Python的标准图形工具包是Tkinter,它基于一种比较老的跨平台工具包Tk
- 为使用Tkinter,只需导入该模块:

  `import Tkinter` 或 `from Tkinter import *`

- 15种控件 （对象）
  - Button 按钮
  - Canvas 画布，可在上面绘制直线，矩形等图形，也可含位图
  - Checkbutton 核对按钮
  - Radiobutton 无线按钮
  - Entry单行文本框，文字可输入和修改
  - Text多行文本框，文字可输入和修改
  - Label标签
  - Listbox 列表框，用户可从中选择
  - Messagebox 消息框，类似于Label,但可以多行
  - Scale 进度条，可设置起始值和终了值
  - Scrollbar 滚动条
  - Menu菜单，点下菜单后弹出选项列表，用户可从中选择
  - Menubutton 菜单按钮，下拉和层叠菜单组件

Tkinter图形编程(1):

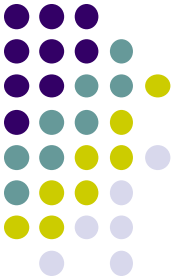
一个简单例子



```
from Tkinter import *
root = Tk()
w = Label(root,text="Hello world!")
w.pack()
root.mainloop()
```

- 第二行:创建Tk根构件,一个普通窗口.
- 第三行:创建一个Label构件,它是根窗口的子构件.
- 第四行:对Label构件进行pack布局
- 第五行:进入事件循环.

Tkinter图形编程(2)



- 创建画布

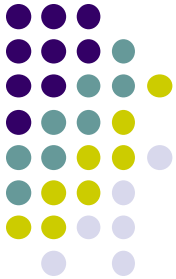
  c = Canvas(<窗口>,<选项>=<值>,...)

  - Canvas是画布类,利用它可以创建画布对象.
  - 在<窗口>中创建画布
  - <选项>=<值>用来设置画布对象的数据,如高度,宽度,背景色等

- 例如:在root中创建300x200的白色画布

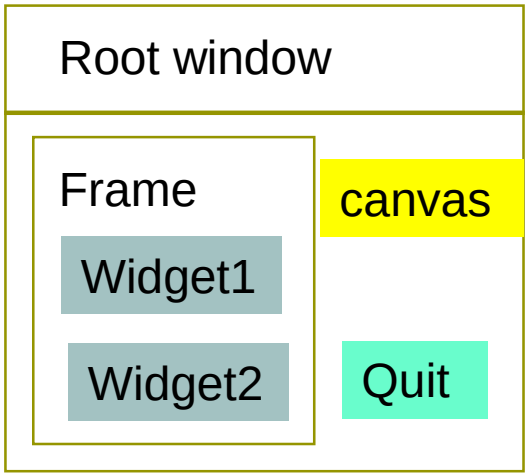
```
c = Canvas(root,width=300,
           height=200,bg='white')
```

Tkinter图形编程(3)



- 布置画布
  - 已经创建了画布c,但在窗口中看不见,因为还需要将画布"布置"到窗口中.
  - `c.pack(side)`以紧凑方式布局
  - `C.grid(row,column)`以二维表格方式布局
  - `C.place(x,y)`指定位置坐标

Tkinter图形编程(4)



- 窗口:构件容器的一种.

  root = Tk()
- 画布:

  cv = Canvas(root,bg='white')

  cv.pack()

  cv.create_rectangle(10,10,100,100)
- 框架(frame):构件容器之一.多用于窗口布局.

  f = Frame(height=..., width=..., bg=...)

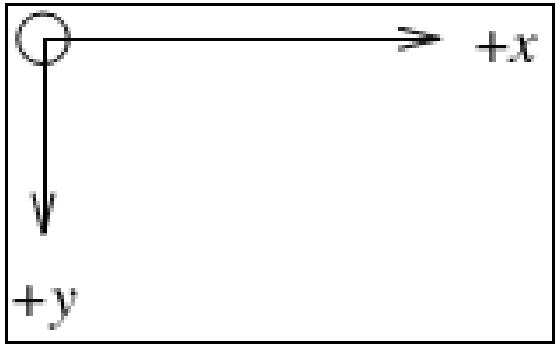
  Label(f, text='hello from frame')
- 顶层(top-level)窗口:类似Frame,但有窗口特征.

  t = Toplevel()

  Label(t, text='hello from Toplevel')

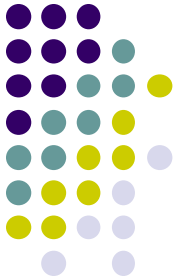
Tkinter图形编程(5)

- 画布上的坐标系



  - 坐标单位是像素;也可用厘米,英寸等.
  - c.pack()表示请画布c执行"pack布局"操作
- 画布上的图形
  - 都有标识号,用于区分同一画布上的多个图形
  - 也可以为图形命名(标签)

画布对象的方法(1)



- 删除画布上的图形

  `c.delete(id)`

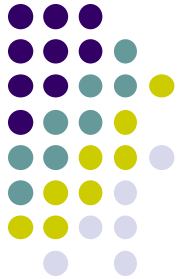
  - id是画布上图形的标识号
- 移动画布上的图形

  `c.move(id,dx,dy)`

  - `id`是图形标识号,`dx`和`dy`是轴向移动距离
- 设置画布上图形的选项

  `c.itemconfig(id,<选项>=<值>...)`

画布对象的方法(2)



- 画矩形

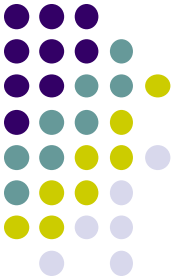
```
c.create_rectangle(x0,y0,x1,y1,<选项>...)
```

或

```
r = c.create_rectangle(...)
```

  - 返回所画矩形的标识号
  - 常用选项
    - **outline = 颜色**
    - **fill = 颜色**
    - **state=NORMAL/HIDDEN**
    - **dash=(线段长,间隔)**

例:画矩形



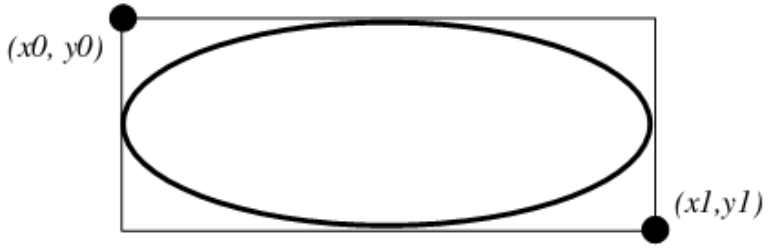
```
>>> c.create_rectangle(50,50,200,100)
1
>>> r2 = c.create_rectangle(80,70,240,150,tags="rect#2")
>>> print r2
2
>>> c.itemconfig(1,fill="black")
>>> c.itemconfig(r2,fill="grey",outline="white",width=6)
>>> c.delete(r2)
>>> c.move(1,50,50)
>>> c.create_rectangle(50,50,51,51) # Tkinter画点
>>> p1 = (10,10)
>>> p2 = (50,80)
>>> c.create_rectangle(p1,p2,tags="#3")
>>> xy = (100,110,200,220)
>>> c.create_rectangle(xy)
```

画布对象的方法(3)

- 画椭圆

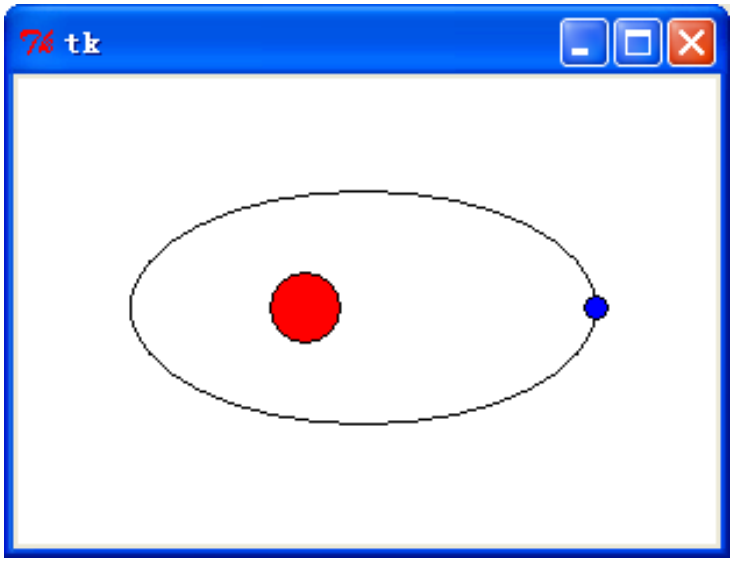
`c.create_oval(x0,y0,x1,y1,<选项>...)`

  - 返回所画椭圆的标识号
  - 常用选项
    - **outline = 颜色**
    - **fill = 颜色**
    - **state=NORMAL/HIDDEN**
    - **dash=(线段长,间隔)**



例:画椭圆

```
>>> o1 = c.create_oval(50,50,250,150)
>>> o2 = c.create_oval(110,85,140,115,fill='red')
>>> o3 = c.create_oval(245,95,255,105,fill='blue')
```

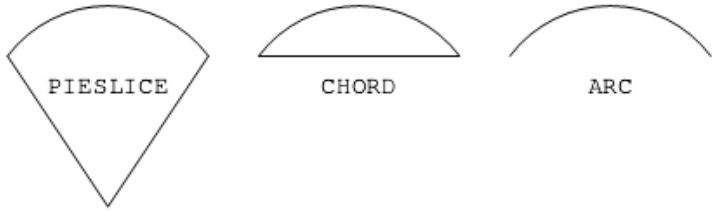


画布对象的方法(4)

- 画弧形

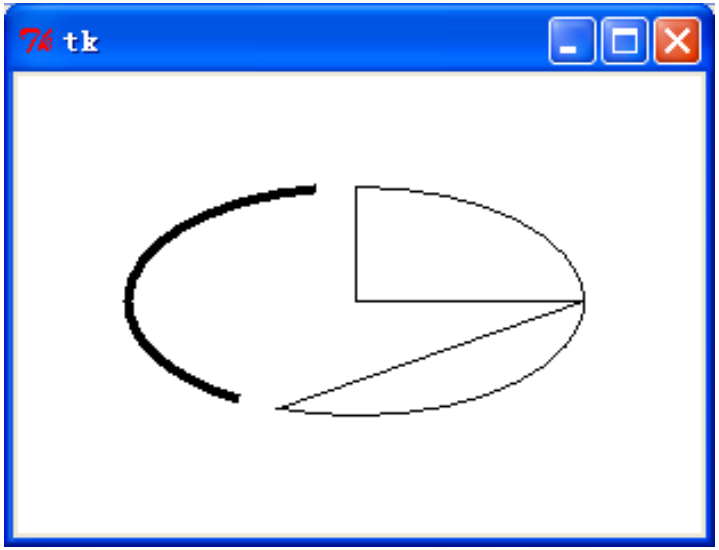
`c.create_arc(x0,y0,x1,y1,<选项>...)`

  - 返回所画弧形的标识号
  - 常用选项
    - **start=开始位置(角度)**
    - **extent=逆时针旋转的角度**
    - **style=PIESLICE/ARC/CHORD**
    - **outline,fill,state, dash等**



例:画弧形

```
>>> bbox = (50,50,250,150)
>>> c.create_arc(bbox)
>>>
   c.create_arc(bbox,start=100,extent=140,style="arc",width=4)
>>> c.create_arc(bbox,start=250,extent=110,style="chord")
```

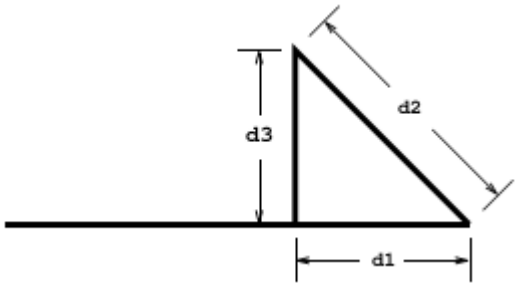


画布对象的方法(5)

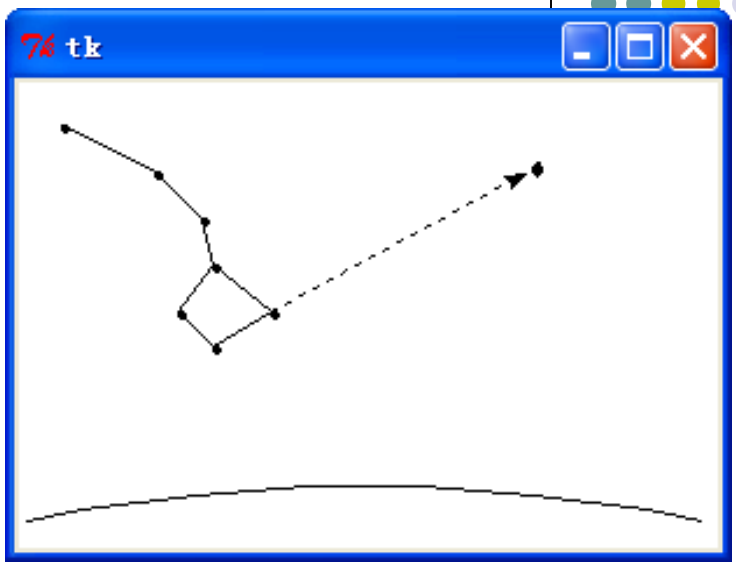
- 画线条

`c.create_line(x0,y0,...xn,yn,<选项>...)`

  - 返回所画线条的标识号
  - 常用选项
    - **smooth=0:折线/非0:平滑曲线**
    - **arrow=NONE/FIRST/LAST/BOTH**
    - **arrowshape=(d1,d2,d3)**
    - **fill,state, dash等**

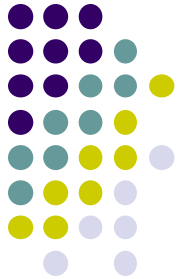


例:画线条




```
>>> s1 = (20,20)
>>> s2 = (60,40)
>>> s3 = (80,60)
>>> s4 = (85,80)
>>> s5 = (70,100)
>>> s6 = (85,115)
>>> s7 = (110,100)
>>> polaris = (220,40)
>>> c.create_oval(s1,(23,23),fill='black')
>>> c.create_oval(s2,(63,43),fill='black')
...
>>> c.create_oval(s7,(113,103),fill='black')
>>> c.create_oval((222,36),(226,42),fill='black')
>>> c.create_line(s1,s2,s3,s4,s5,s6,s7,s4)
>>> c.create_line(s7,polaris,dash=(4,),arrow=LAST)
>>> c.create_line(5,190,150,160,295,190,smooth=1)
```

画布对象的方法(6)



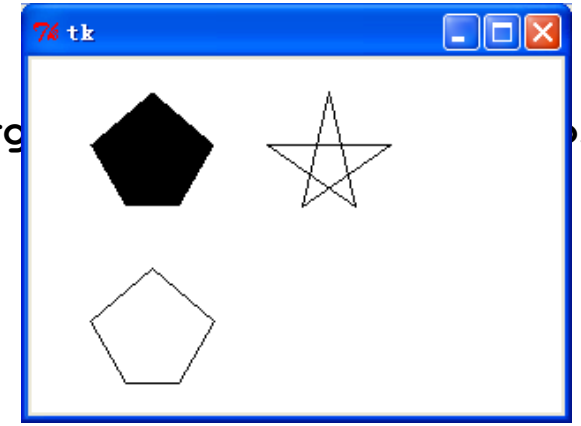
- 画多边形

`c.create_polyfon(x0,y0,...xn,yn,<选项>...)`

  - 返回所画多边形的标识号
  - 常用选项
    - **smooth=0:折线/非0:平滑曲线**
    - **outline(缺省值为空)**
    - **fill(缺省值为黑色)**
    - **state, dash等**

例:画弧形

```
>>> p11,p21,p31 = (70,20),(70+100,20),(70,20+100)
>>> p12,p22,p32 = (35,50),(35+100,50),(35,50+100)
>>> p13,p23,p33 = (55,85),(55+100,85),(55,85+100)
>>> p14,p24,p34 = (85,85),(85+100,85),(85,85+100)
>>> p15,p25,p35 = (105,50),(105+100,50),(105,50+100)
>>> c.create_polygon(p11,p12,p13,p14,p15)
>>>
   c.create_polygon(p21,p23,p25,p22,p24,outline="black",fill="
   ")
>>>
   c.create_polyg                                  35,outline="black",fill="
   ")
```

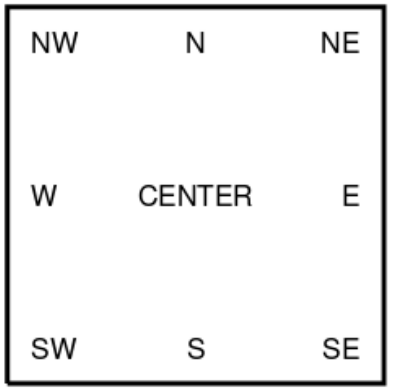


画布对象的方法(7)

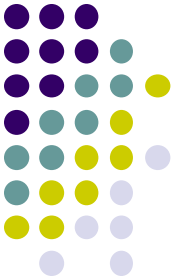
- 创建文本

`c.create_text(x,y,<选项>...)`

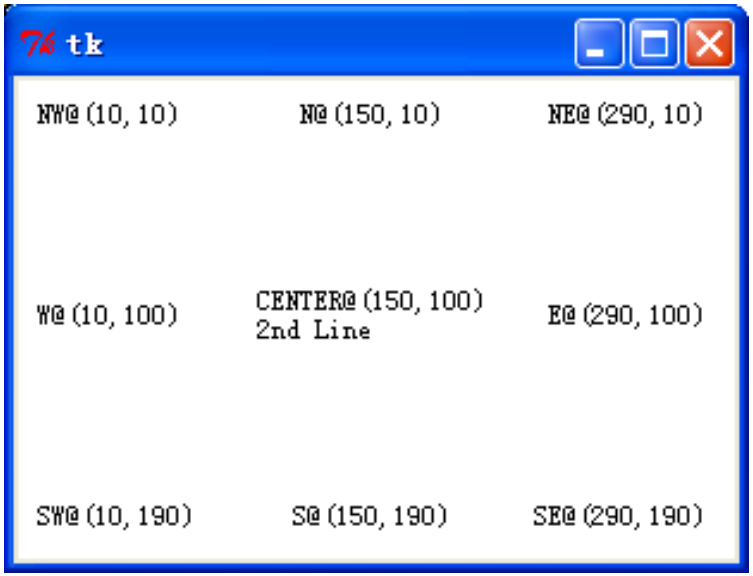
  - 返回所创建文本的标识号
  - 常用选项
    - **text="文本内容"**
    - **anchor=E/S/W/N/CENTER/SE/NE/...**
    - **justify=LEFT/CENTER/RIGHT**
    - **fill,state, dash等**



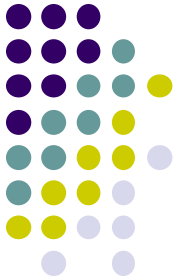
例:创建文本



```
>>> t1 = c.create_text(10,10,text="NW@(10,10)",anchor=NW)
>>> c.create_text(150,10,text="N@(150,10)",anchor=N)
>>> c.create_text(290,10,text="NE@(290,10)",anchor=NE)
>>> c.create_text(10,100,text="W@(10,100)",anchor=W)
>>> c.create_text(150,100,text="CENTER@(150,100)\n2nd Line")
```



画布对象的方法(8)



- 创建图像
  - 先创建图像对象

```
img = PhotoImage(file = <gif图像文件名>)
```

  - 在画布上显示图像对象

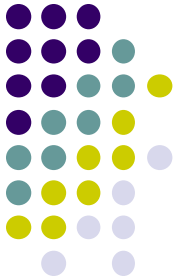
```
c.create_image(x,y,image=img,<选项>...)
```

  - 返回所创建图像的标识号
  - 常用选项
    - **anchor=E/S/W/N/CENTER/SE/NE/...**
    - **state, tags等**

例:创建图像

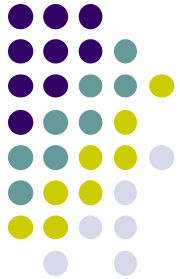
```
>>> pic = PhotoImage(file="C:\WINDOWS\Web\exclam.gif")
>>> c.create_image(150,100,image=pic)
```

第8章 图形用户界面GUI



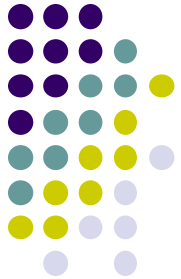
- GUI基本概念
- Tkinter编程
- 事件驱动编程
- MVC设计方法

事件绑定函数



- GUI应用程序一般都有一个事件循环(通过mainloop方法进入).
- 事件来源有多种,包括用户的按键和鼠标操作等.
- Tkinter提供了强大机制使用户能自己处理事件:对每种构件,可将Python函数或方法绑定到事件上.
- 事件：
  - 鼠标事件:
    - <Button-1>, <B1-Motion>, <Enter>, <Leave>,...
  - 键盘事件:
    - <Return>, <F5>, <‘a’>,<‘8’>,...
    - 总是发送到当前拥有键盘焦点的构件.可用<widget>.focus_set()将焦点移到某个构件

常用事件



<Enter>: 鼠标进入构件
<Leave>: 鼠标离开构件
<Button-1>：按下鼠标左键
<Button-2>：按下鼠标中键
<Button-3>：按下鼠标右键
<B1-Motion>:按下鼠标左键并移动
<Double-Button-1>：双击鼠标左键
<Key>：按下任意键
<Key-a>: 按下a键
<Return>：按下回车键
……

事件绑定

1. 实例绑定：仅对该实例有效

< widget >.bind(<event>, <callback>)

事件处理程序也叫回调(callback)函数

2. 类绑定：对所有该类的实例有效

root.bind_class(<class>, <event>,<callback>)

3. 窗口绑定：对窗口中所有构件有效

root. bind (<event>, <callback>

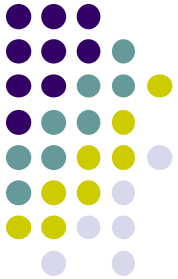
4. 应用程序绑定：对应用中的所有构件有效

root.bind_all(<event>, <callback>)

5. 画布构件绑定：画布中的指定tagID有效

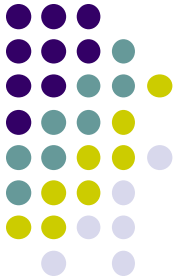
Canvas.tag_bind(<tagID>, <event>,<callback>)

事件编程



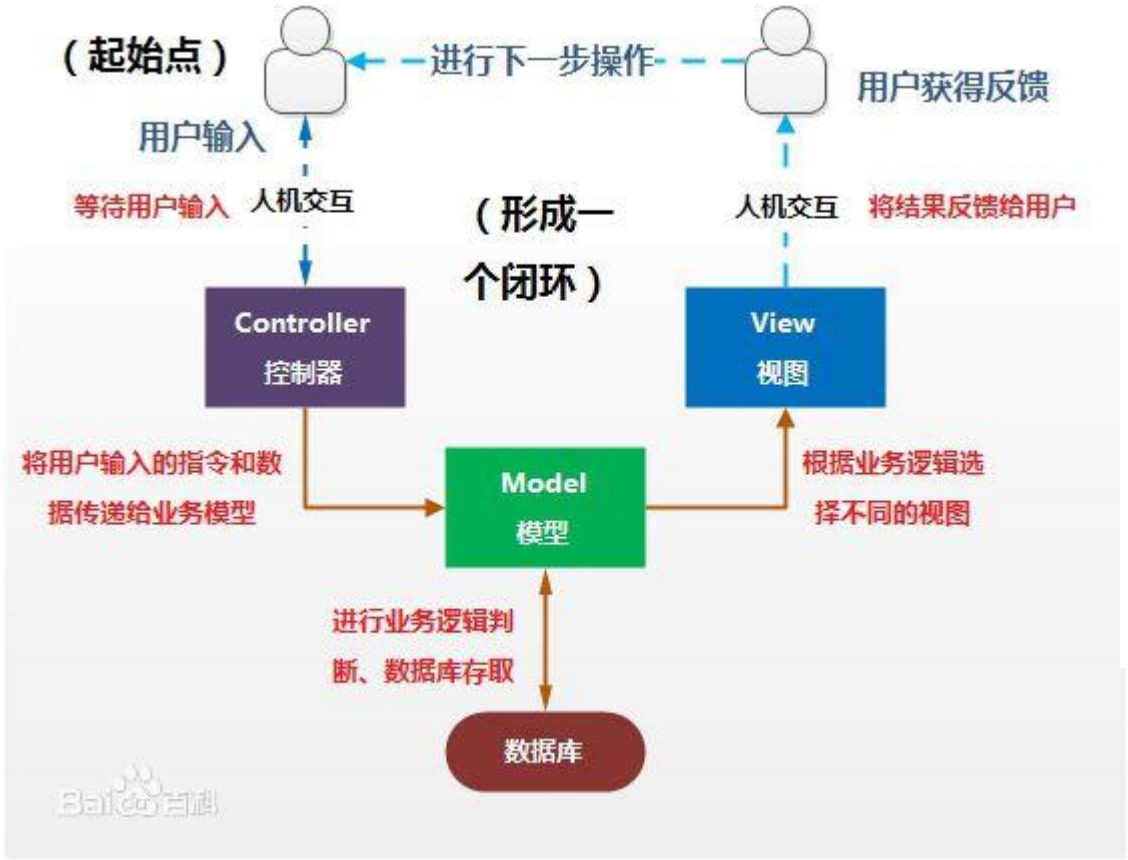
```
def canvasFunc(event):
    if c.itemcget(t,"text") == "Hello!":
        c.itemconfig(t,text="Goodbye!")
    else:
        c.itemconfig(t,text="Hello!")
def textFunc(event):
    if c.itemcget(t,"fill") != "white":
        c.itemconfig(t,fill="white")
    else:
        c.itemconfig(t,fill="black")
...
t = c.create_text(150,100,text="Hello!")
c.bind("<Button-1>",canvasFunc)       #画布与左键绑定
c.tag_bind(t,"<Button-3>",textFunc)  #画布上文本与右键绑定
root.mainloop()
```

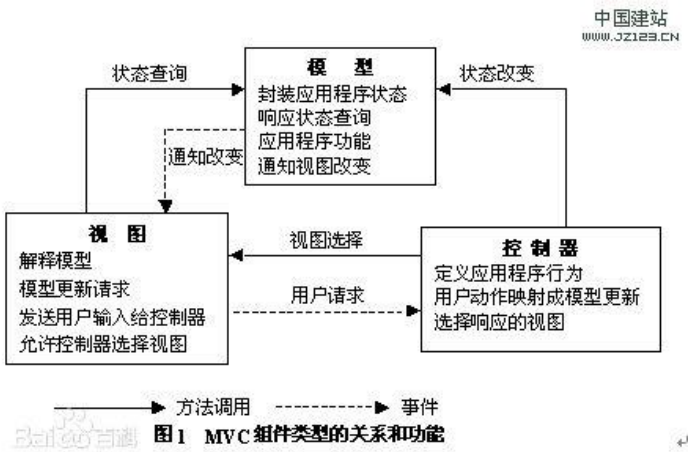
第8章 图形用户界面GUI



- GUI基本概念
- Tkinter编程
- 事件驱动编程
- MVC设计方法

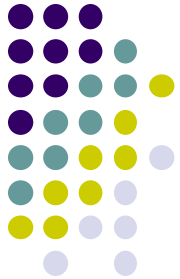
MVC框架





- MVC是一个框架模式，它强制性的使应用程序的输入、处理和输出分开。
- MVC 分层同时也简化了分组开发。不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

编程案例:统计图表(1)



- 程序规格

  输入:考试分数

  输出:以饼图表示的各分数段所占比例

- 算法

  输入分数mark,换算成a,b,c,d,f等级并累加该等级的人数;

  创建窗口和画布;

  计算各等级的比例(a/n等),并据此确定扇形起止角度(sA,eA等);

  绘制各扇形;

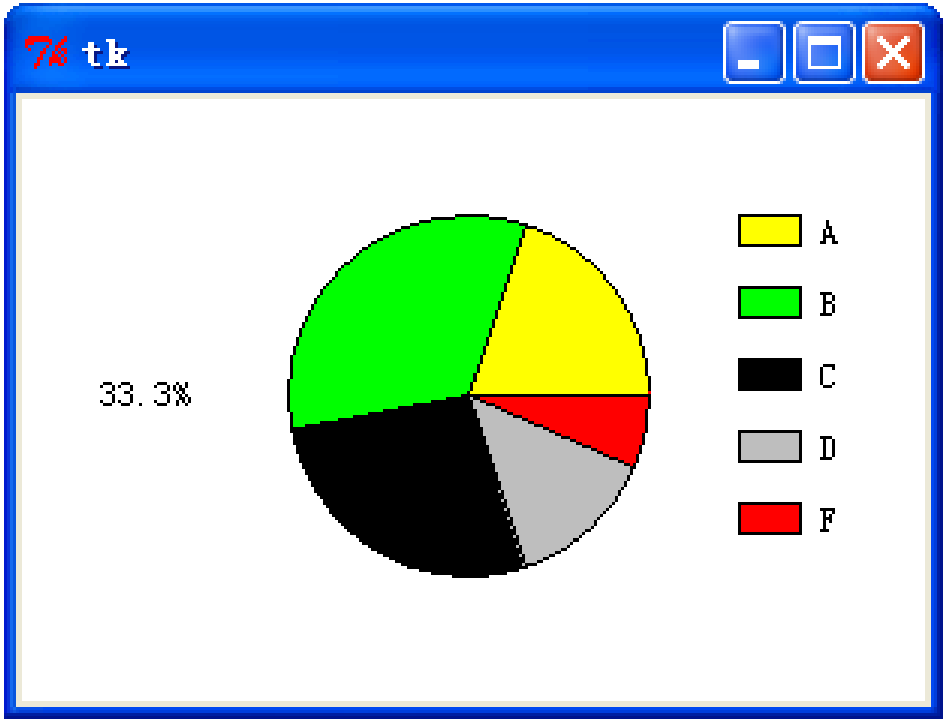
  绘制图例;

  为各扇形绑定“鼠标进入”事件,并定义事件处理函数(inPieA()等);

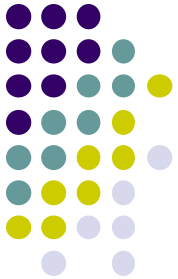
  进入主事件循环。

编程案例:统计图表(2)

- 代码实现:**piechart.py**



动画编程



- 动画:运动的图形.
  - 现实中运动是连续的.
- 计算机动画
  - 运动离散化:t, t+Δt, t+2Δt, t+3Δt,...
  - 快速交替显示一组静止图形,或者让一幅图形快速移动.
  - 关键是控制交替显示或移动的速度:24帧/秒

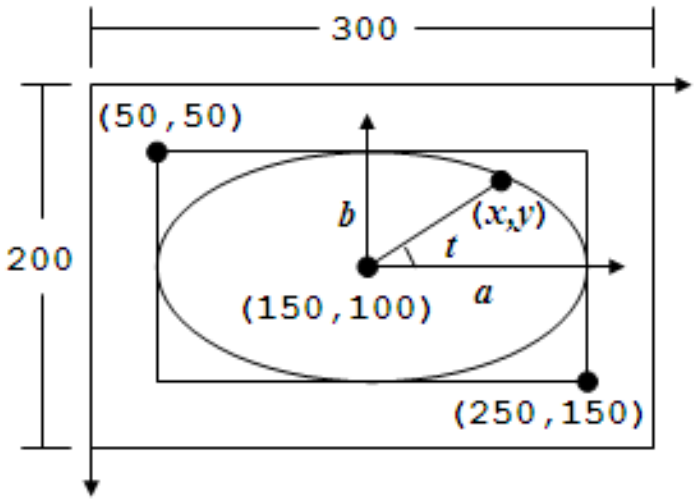
编程实例:演示天体运动(1)

- 地球在椭圆轨道上运动的计算:

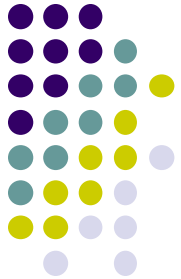
$$\begin{cases} x = a\cos t \\ y = b\sin t \end{cases} \quad \begin{cases} x' = a\cos(t+0.01\pi) \\ y' = b\sin(t+0.01\pi) \end{cases} \quad \begin{cases} dx = x' - x \\ dy = y' - y \end{cases}$$

  - 画布坐标系中:

$$\begin{cases} x = 150 + a\cos t \\ y = 100 - b\sin t \end{cases}$$



编程实例:演示天体运动(2)



- 月球一方面随地球绕太阳运动,一方面还绕地球运动.
  - 分别计算两种运动导致的坐标变化,然后求和.

编程实例:演示天体运动(3)

- 算法

  创建窗口和画布;

  在画布上绘制太阳、地球和月球,以及地球的绕日椭圆轨道;

  设置地球和月球的当前位置;

  进入动画循环:

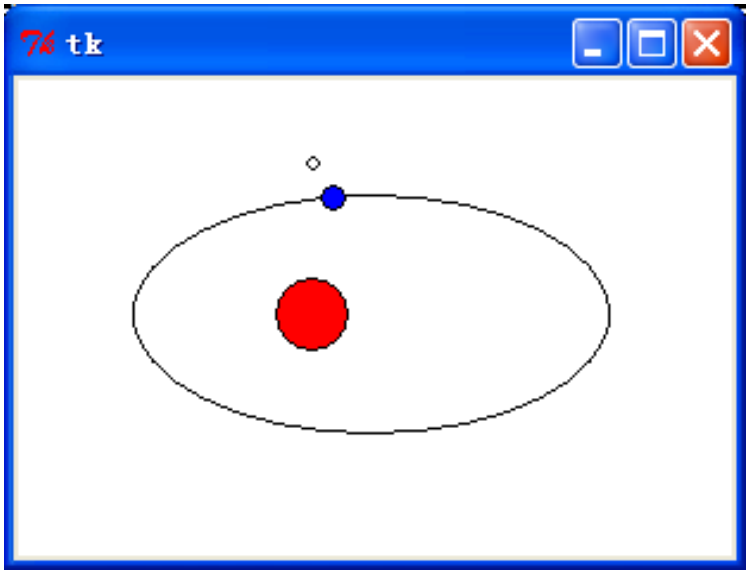
  旋转0.01π

  计算地球和月球的新位置

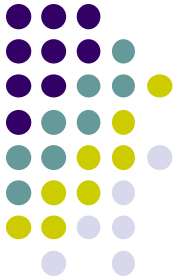
  移动地球和月球到新位置

  更新地球和月球的当前位置;

  停顿一会



- 程序:animation.py



End