

Python 语言程序设计

陈 峦 副教授

13880209111, chluan@uestc.edu.cn

研究院大楼316#

第十二章 图形绘制

- **Python的图形库：**
- **（1）Python自带的标准图形库，如tkinter模块中的画布绘图、在Tkinter图形库基础上建立的graphics模块。**
- **（2）第三方图形库，如wxPython、PyGTK、PyQt、PySide等。**
- **（3）Python内置的turtle绘图模块。**

12.1 Tkinter图形库概述

- **Tkinter**（**Tk interface**，**Tk接口**）图形库是**Tk**图形用户界面工具包的**Python**接口。
- **Tk**是一种流行的跨平台图形用户界面（**Graphical User Interface**，**GUI**）开发工具。
- **TKinter**图形库通过定义一些类和函数，实现了一个在**Python**中使用**Tk**的编程接口。
- **Tkinter**图形库就是**Python**版的**Tk**。

12.1.1 tkinter模块

- Tkinter图形库由若干模块组成：
- （1）_tkinter： 是二进制扩展模块。
- （2）tkinter： 是主模块。
- （3）tkinter.constants： 该模块定义了许多常量。

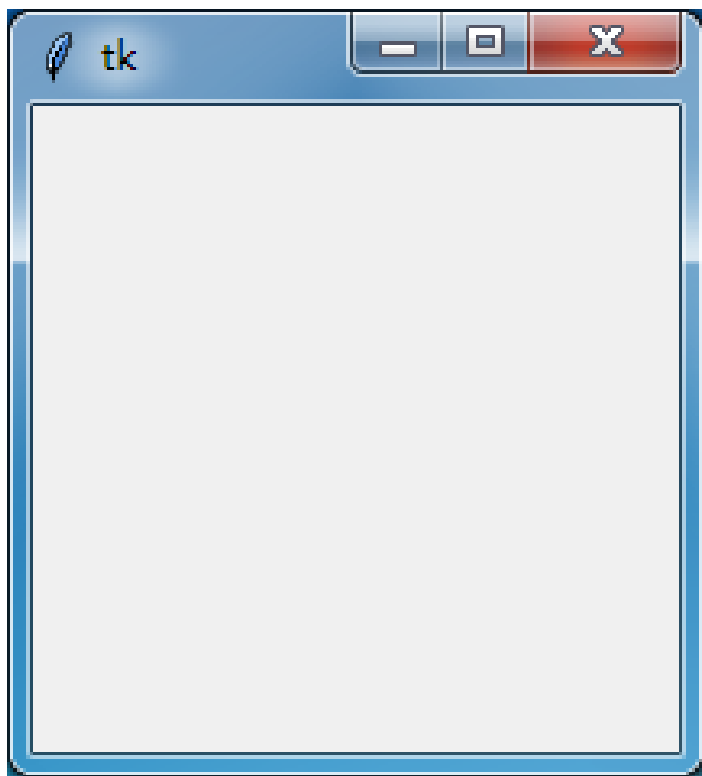
- **_tkinter**模块提供了对Tk的低级接口。低级接口并不会被应用级程序员直接使用，通常是一个共享库或DLL，但是在某些情况下，它也可被Python解释器静态链接。
- **tkinter**是最重要的模块，导入**tkinter**模块时，会自动导入**tkinter.constants**模块。

- 图形处理首先需要做的是导入**tkinter**模块。
- 导入**tkinter**模块的方法：
- **>>>import tkinter**
- **>>>from tkinter import ***

12.1.2 主窗口的创建

- 主窗口：也称为根窗口，是一个顶层窗口，所有图形都是在这个窗口中绘制的。
- 在导入tkinter模块之后，接下来就要使用Tk类的无参构造方法Tk()创建主窗口。
- 主窗口是一个对象，其创建格式为：
- 窗口对象名= Tk()

- 例：创建主窗口w。
- `>>> from tkinter import *`
- `>>> w=Tk()`



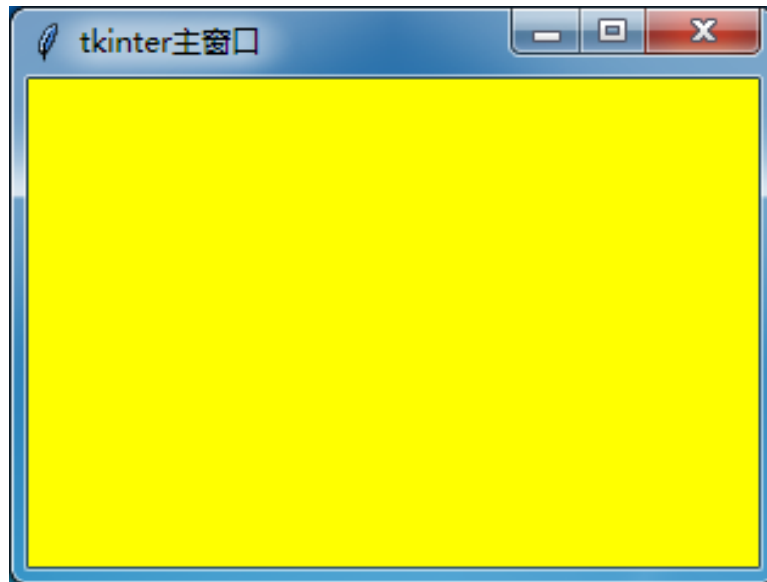
主窗口的属性：宽度

（**width**）、高度（**height**）、背景颜色（**bg**或**background**）等。主窗口的默认宽度和高度都为**200**像素、背景颜色为浅灰色。

主窗口的方法：设置窗口属性等。

- 例：设置主窗口的宽度、高度和背景颜色属性。
- **>>> from tkinter import ***
- **>>> w=Tk()**
- **>>> w['width']=300**
- **>>> w['height']=200**
- **>>> w['bg']='yellow'**

- 主窗口默认的窗口标题是tk。
- 可以通过调用主窗口对象的**title()**方法来设置窗口标题。
- 例：设置主窗口的标题为 “ **tkinter主窗口** ”。
- **>>> w.title('tkinter主窗口')**



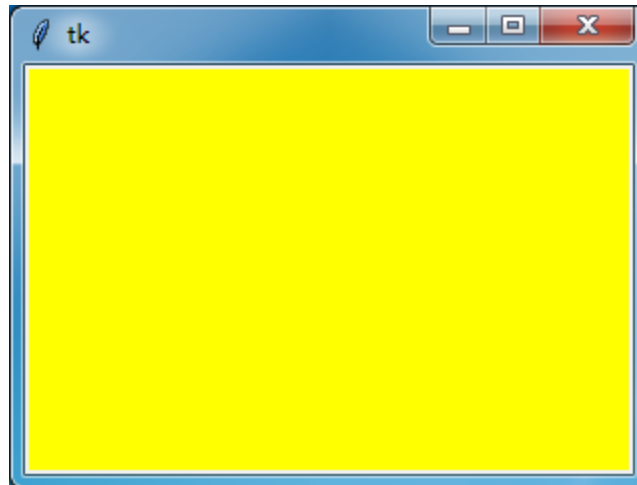
- 可以通过主窗口对象的**resizable()**方法来设置窗口的长度、宽度是否可以变化。
- 例：设置主窗口的宽度不可变，而高度可变。
- **>>> w=Tk()**
- **>>> w.resizable(width=False,height=True)**
- 其中，**width**和**height**的默认值均为**True**，即长度和宽度均是可变的。

12.1.3 画布对象的创建

- 画布（**canvas**）就是用来进行绘图的区域，**tkinter** 模块的绘图操作都是通过画布进行的。
- 画布是一个对象，可以在画布上绘制各种图形、标注文本。
- 创建画布对象语句的格式：
- 画布对象名=Canvas(窗口对象名,属性名=属性值,.....)

- 画布对象的属性：如画布的宽度（**width**）、高度（**height**）、背景颜色（**bg**或**background**）等。**bg**的默认值为浅灰色。
- 画布的方法：如在画布上创建图形、删除或移动图形等。

- 例:
- `>>> w=Tk()`
- `>>> c=Canvas(w,width=300,height=200,bg='green')`
- `>>> c.pack()`
- `>>> c['bg']='yellow'`



12.1.4 画布对象的坐标系

- 画布坐标系以画布左上角为原点，从原点水平向右为x轴正方向，从原点垂直向下为y轴正方向。
- 画布坐标以整数给出，则度量单位是像素。
- 也支持以字符串形式给出其他坐标度量单位的长度值，例如5c表示5厘米、50m表示50毫米、2i表示2英寸等。



12.1.5 画布中的图形对象

- 画布中创建的每个图形都是一个对象，称为图形对象。
- 例：矩形、椭圆、圆弧、线条、多边形、文本、图像等。
- 每个图形对象有自己的属性和方法。

- **tkinter**模块没有采用为每种图形提供单独的类来创建图形对象的实现方式，而是采用画布对象的方法来实现。
- 例：画布对象的**create_rectangle()**方法可以创建一个矩形对象。

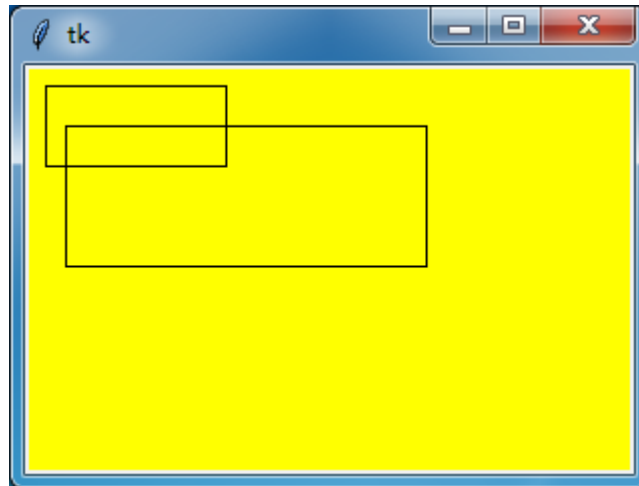
- 1. 图形对象的标识

- 画布中的图形对象需要采用某种方法来标识和引用，以便对该图形对象进行处理，具体采用标识号和标签（tag）两种标识方法。
- 标识号是创建图形对象时自动为图形对象赋予的唯一的整数编号。

- 标签相当于给图形对象命名。
- 一个图形对象可以与多个标签相关联，而同一个标签可以与多个图形对象相关联。
- 即一个图形对象可以有多个名字，而且不同图形对象可以有相同的名字。

- 给图形对象指定标签有三种方法：
- （1）在创建图形时利用**tags**属性来指定标签，可以将**tags**属性设置为单个字符串，即单个名字，也可以设置为一个字符串元组，即多个名字。
- （2）在创建图形之后，可以利用画布的**itemconfig()**方法对**tags**属性进行设置。
- （3）利用画布的**addtag_withtag()**方法来为图形对象添加新标签。

- 例:
- `>>> w=Tk()`
- `>>> c=Canvas(w,width=300,height=200,bg='yellow')`
- `>>> c=Canvas(w,width=300,height=200,bg='green')`
- `>>> id1=c.create_rectangle(10,10,100,50,tags="No1")`
- `>>> id2=c.create_rectangle(20,30,200,100,tags=("myRect","No2"))`
- `>>> c.itemconfig(id1,tags=("myRect","Rect1"))`
- `>>> c.addtag_withtag("ourRect","Rect1")`



- 说明：
- **c.itemconfig(id1,tags=("myRect","Rect1"))**
- 该语句将id1的标签重新设置为myRect和Rect1，此时原标签No1即告失效。
- **c.addtag_withtag("ourRect","Rect1")**
- 该语句为具有标签Rect1的图形对象（即第一个矩形）添加一个新标签ourRect。
- 结果：第一个矩形具有三个标签，即myRect、Rect1和ourRect。标签myRect同时引用两个矩形。

- 画布还预定义了**ALL**或**all**标签，该标签与画布上的所有图形对象相关联。

- **2. 图形对象的共性操作**

- **(1) `gettags()`方法：**用于获取给定图形对象的所有标签。
- **(2) `find_withtag()`方法：**用于获取与给定标签相关联的所有图形对象。返回结果为各图形对象的标识号所构成的元组。

- 例：（接续前例）
- `>>> print(c.gettags(id1))`
- `('myRect', 'Rect1', 'ourRect')`
- `>>> print(c.find_withtag("Rect1"))`
- `(1,)`
- `>>> print(c.find_withtag("all"))`
- `(1, 2)`
- #显示画布中所有的图形对象

- (3) **delete()**方法：用于从画布上删除指定的图形对象。
- 例： `>>>c.delete(id1)`
- (4) **move()**方法：用于在画布上移动指定图形。
- 例：
- `>>>c.move(id2,10,20)`
- `>>>c.move(id2,-10,-20)`
- `>>>c.move(id2,'10m','20m')`

12.2 画布绘图

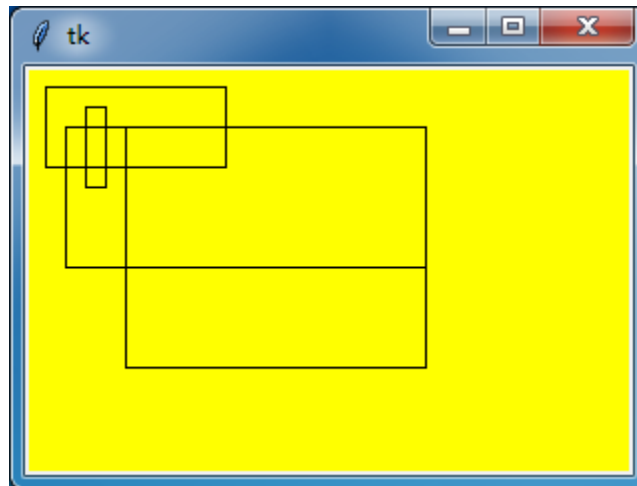
- 绘制图形前，先要导入**tkinter**模块、创建主窗口、创建画布并使画布可见。
- 例：
- **from tkinter import ***
- **w=Tk()**
- **c=Canvas(w,width=300,height=200,bg='yellow')**
- **c.pack()**

12.2.1 绘制矩形

- 1. `create_rectangle()`方法
- 画布对象提供`create_rectangle()`方法，用于在画布上创建矩形，其调用格式：
- 画布对象名.`create_rectangle(x0,y0,x1,y1,属性设置.....)`
- 其中，`(x0,y0)`是矩形左上角的坐标，`(x1,y1)`是矩形右下角的坐标。

- 例:
- `>>>c.create_rectangle(50,30,200,150)`
- **1**
- 语句返回的**1**是矩形的标识号，表示这个矩形是画布上的**1**号图形对象。

- 为了将来在程序中引用图形，一般用变量来保存 `create_rectangle()` 方法返回的图形标识号，或者将图形与某个标签相关联。
- 例：
- `>>> x=c.create_rectangle(30,20,40,60,tags="R2")`
- `>>> x`
- 4



- 2. 矩形对象的常用属性

- (1) 矩形边框属性

- **outline**属性

- 矩形边框可以用**outline**属性来设置颜色，其默认值为黑色。
- 如果将**outline**设置为空串，则不显示边框，即透明的边框。

- 在Python中，颜色用字符串表示。
- 例：red（红色）、yellow（黄色）、green（绿色）、blue（蓝色）、gray（灰色）、cyan（青色）、magenta（品红色）、white（白色）、black（黑色）等。
- 颜色还具有不同的深浅。
- 例：red1、red2、red3、red4表示红色逐渐加深。

- 计算机中通常用三原色模型表示颜色，将红（**R**）、绿（**G**）、蓝（**B**）以不同的值叠加，产生各种颜色。
- 三原色模型又称为**RGB**颜色模型。
- 用三元组来表示**RGB**颜色，有三种字符串表示形式：**#rgb**、**#rrggbb**、**#rrrrggggbbbb**。
- 例：
- **#f00**表示红色、**#00ff00**表示绿色、**#000000ffff**表示蓝色。

- **width属性**: 边框的宽度, 默认值为1像素。
- **dash属性**: 画成虚线形式边框。该属性的值是整数元组。最常用的是二元组(a,b), 其中a指定要画多少个像素, b指定要跳过多少个像素, 如此重复, 直至边框画完。若a、b相等, 可以简记为(a,)或a。

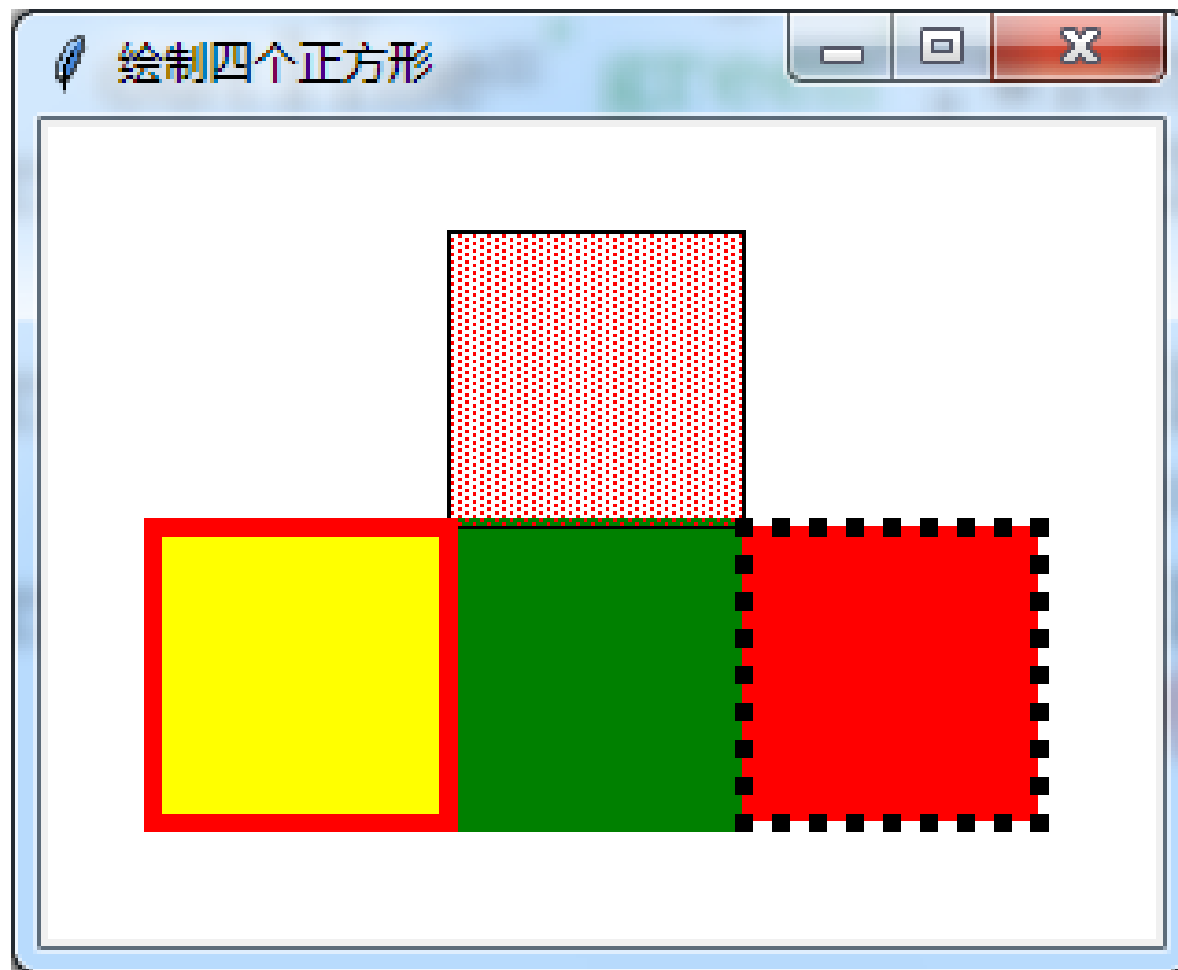
- **（2）矩形内部填充属性**
- **fill属性**：设置矩形内部区域填充颜色，默认值是空串，效果是内部透明。
- **stipple属性**：在填充颜色时设置填充画刷，即填充的点刻效果，可以取**gray12**、**gray25**、**gray50**、**gray75**等值。

- **state属性**：设置图形的显示状态。
- 默认值是**NORMAL**或**normal**，即正常显示。
- 另一个属性值是**HIDDEN**或**hidden**，它使矩形在画布上不可见。
- 使一个图形在**NORMAL**和**HIDDEN**两个状态之间交替变化，即形成闪烁的效果。
- 注意：属性值用大写字母形式时，不要加引号，而用小写字母形式时，一定要加引号。

- 例:
- `>>> c.itemconfig(1,fill="blue")`
- `>>> c.itemconfig(2,fill="grey",outline="white",width=5)`

- 在画布上创建的矩形是覆盖在先创建的矩形之上的，并且未涂色时矩形内部是透明的，即能看到被覆盖的矩形的边框。
- 画布上的所有图形对象都是按创建次序堆叠起来的，第一个创建的图形对象处于画布底部（最靠近背景），最后创建的图形对象处于画布顶部（最靠近前景）。
- 图形的位置如果有重叠，上面的图形会遮挡下面的图形。

- 例：绘制四个正方形。

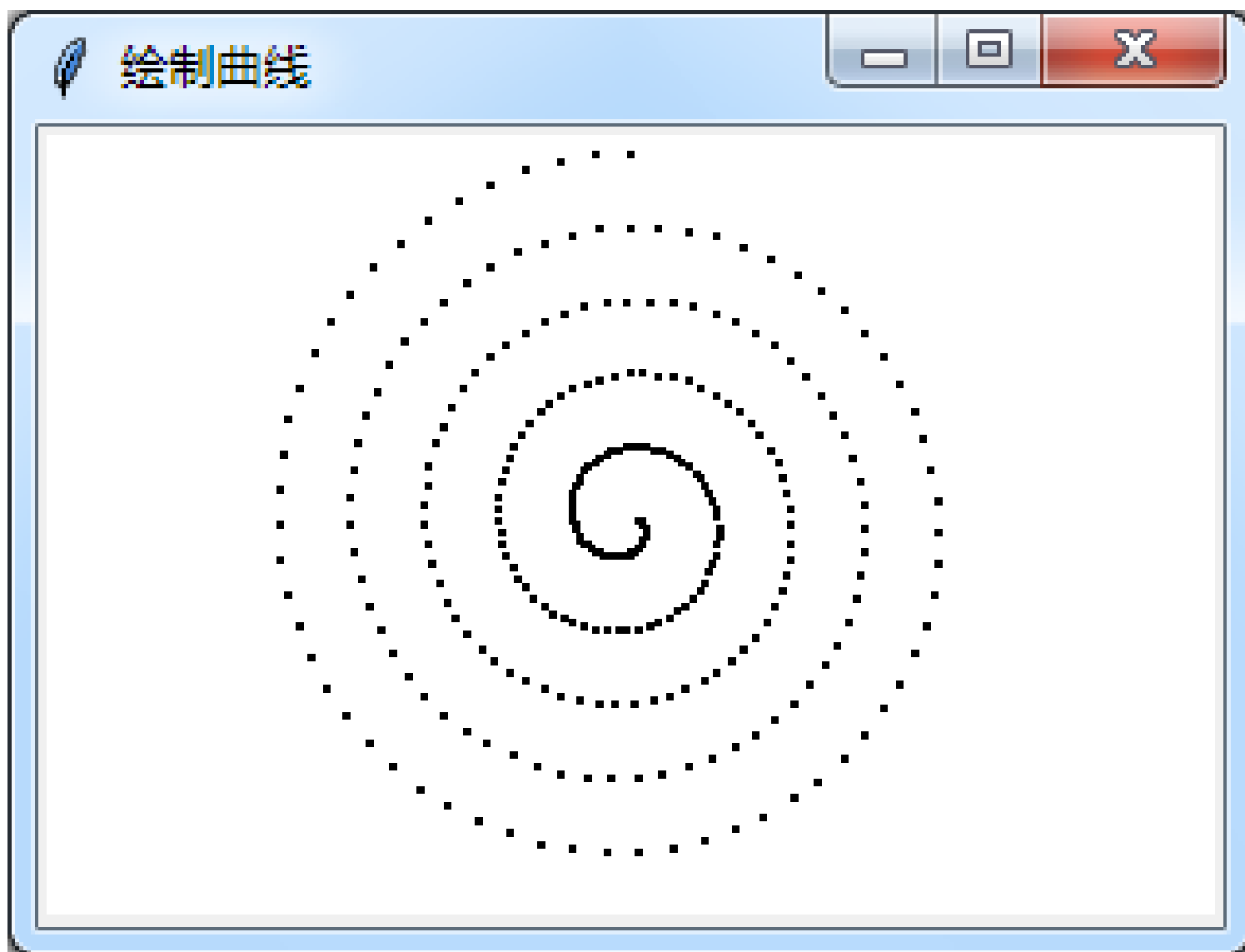


- `from tkinter import *`
- `w=Tk()` `#创建主窗口`
- `w.title('绘制四个正方形')`
- `c=Canvas(w,width=300,height=220,bg='white')` `#创建画布对象`
- `c.pack()` `#使画布可见`
- `c.create_rectangle(110,110,190,190,fill='green',\`
- `outline='green',width=5)` `#绘制无边框绿色正方形`
- `c.create_rectangle(110,30,190,110,fill='#ff0000',\`
- `stipple='gray25')` `#绘制红色点画正方形`
- `c.create_rectangle(30,110,110,190,fill='yellow',\`
- `outline='red',width=5)` `#绘制红色边框黄色正方形`
- `c.create_rectangle(190,110,270,190,dash=10,width=5,\`
- `fill='red')` `#绘制虚线边框红色正方形`

例：绘制曲线 $\begin{cases} x = 3(\cos t + t \sin t) \\ y = 3(\sin t - t \cos t) \end{cases}, t \in [0, 10\pi]$ 。

- 分析：
- 绘制函数曲线可采用计算出函数曲线的各个点的坐标，将各点画出来，如果这些点足够密，绘出的曲线会比较光滑。
- 画布对象没有提供画“点”的方法，但可以画一个很小的矩形来作为点。

- `from math import *`
- `from tkinter import *`
- `w=Tk()`
- `w.title('绘制曲线')`
- `c=Canvas(w,width=300,height=200,bg='white')`
- `c.pack()`
- `#绘制函数曲线`
- `t=0`
- `while t<=10*pi:`
- `x=3*(cos(t)+t*sin(t))`
- `y=3*(sin(t)-t*cos(t))`
- `x+=150` `#移动坐标`
- `y+=100`
- `c.create_rectangle(x,y,x+0.5,y+0.5)`
- `t+=0.1`



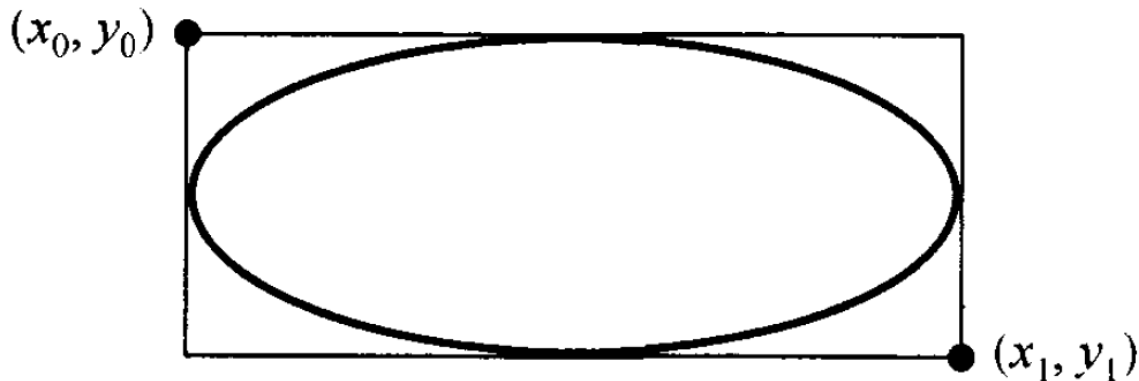
- **3. 矩形对象的坐标表示**
- **create_rectangle()**方法中坐标参数的形式是很灵活的。
- 既可以直接提供坐标值，也可以先将坐标数据存入变量，然后将该变量传给该方法；
- 既可以将所有坐标数据构成一个元组，也可以将它们组成多个元组。

- 例:
- >>> p1=(10,10)
- >>> p2=(150,100)
- >>> c.create_rectangle(p1,p2)
- >>> p3=(50,60,100,150)
- >>> c.create_rectangle(p3)

- 推荐将坐标存储在变量中，这更便于在绘制多个图形时计算它们之间的相对位置。
- 坐标表示方法对所有图形对象（只要用到坐标参数）都是适用的。

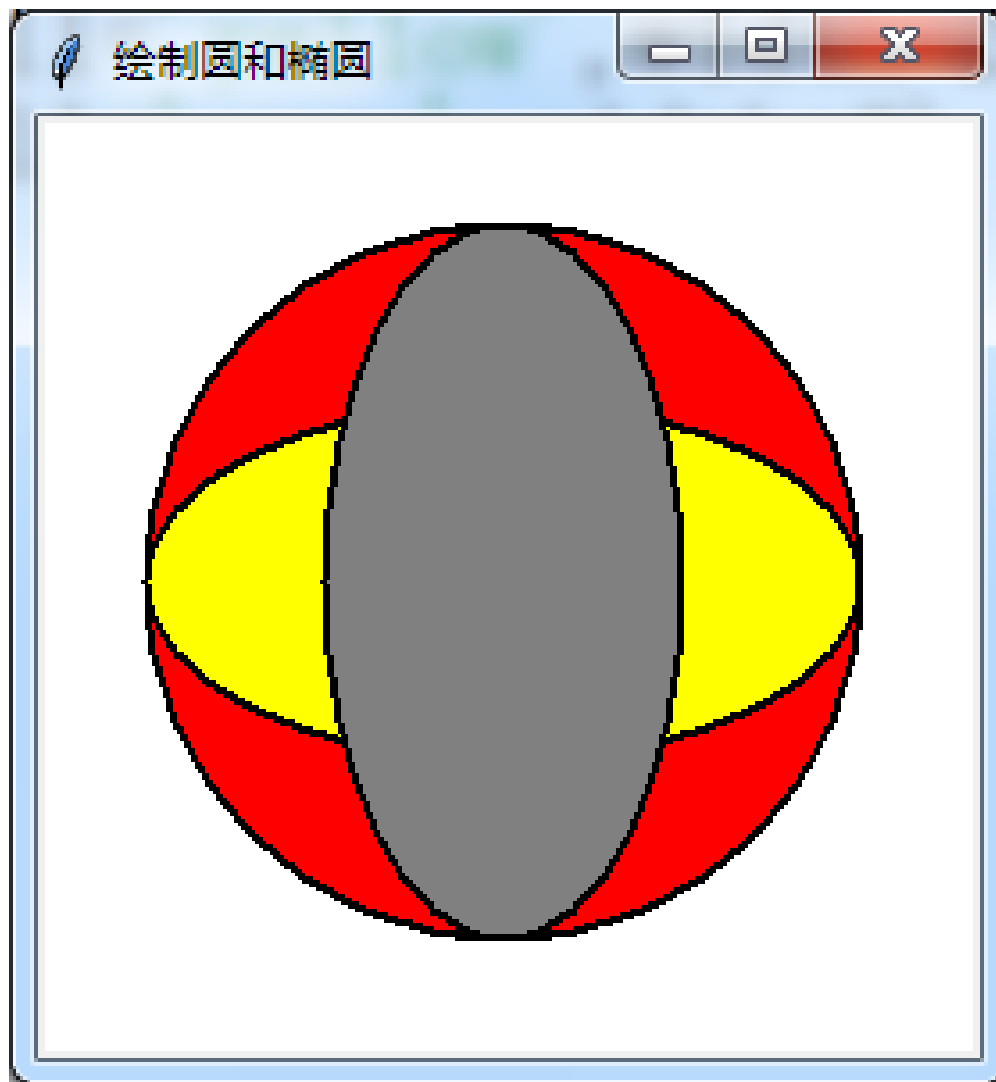
12.2.2 绘制椭圆与圆弧

- `create_oval()`方法用于在画布上画一个椭圆，其特例是圆。
- 椭圆的位置和尺寸由其外接矩形决定，而外接矩形由左上角坐标 (x_0, y_0) 和右下角坐标 (x_1, y_1) 定义。



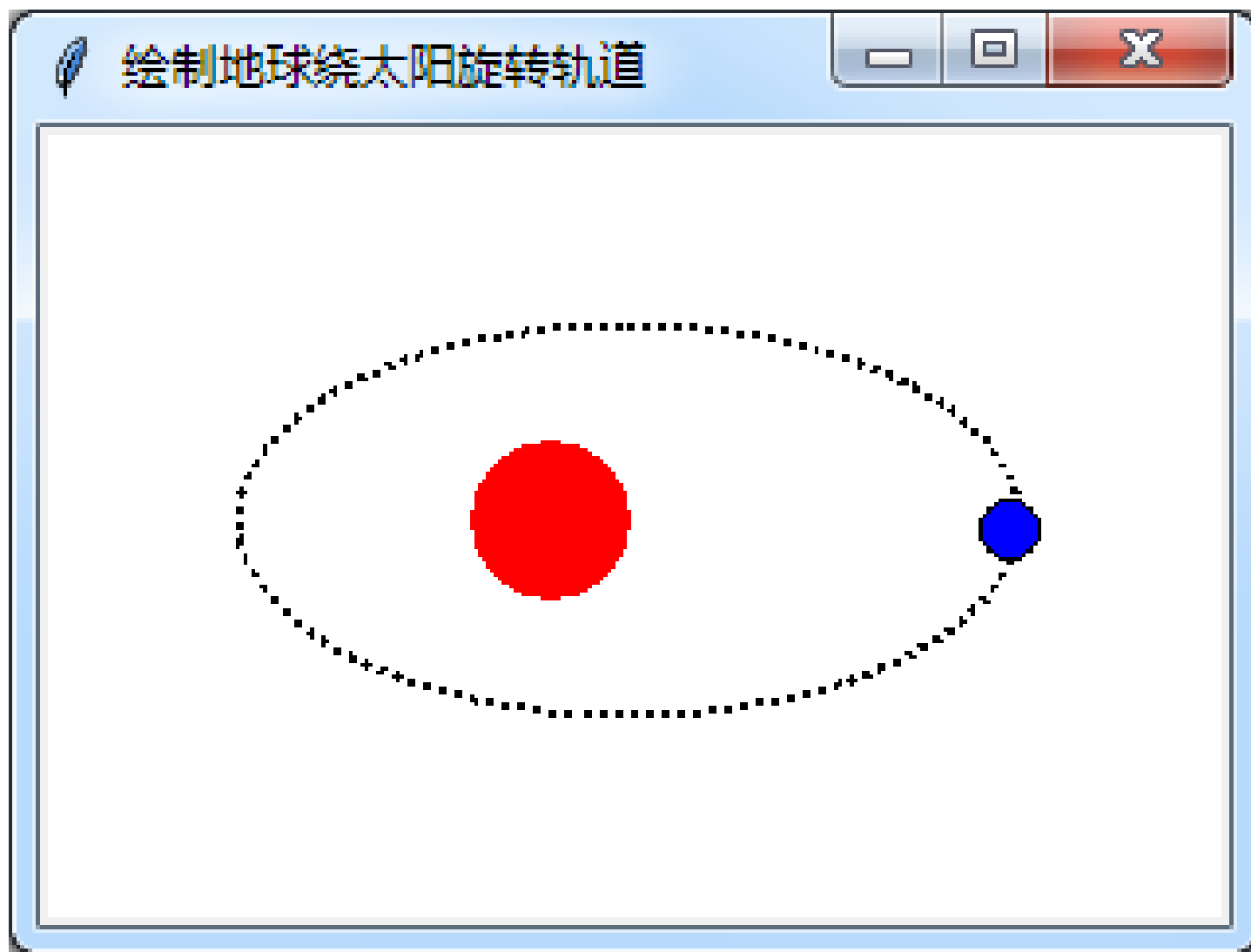
- **create_oval()**方法的调用格式:
- 画布对象名.**create_oval(x0,y0,x1,y1,属性设置.....)**
- 其返回值是所创建椭圆的标识号，可以将标识号存入变量。
- 与矩形类似，椭圆的常用属性包括**outline**、**width**、**dash**、**fill**、**state**和**tags**等。
- 画布对象的**itemconfig()**方法、**delete()**方法和**move()**方法同样可用于椭圆的属性设置、删除和移动。

- 例：创建圆和椭圆。



- `from tkinter import *`
- `w=Tk()`
- `w.title('绘制圆和椭圆')`
- `c=Canvas(w,width=260,height=260,bg='white')` #创建画布对象
- `c.pack()`
- `c.create_oval(30,30,230,230,fill='red',width=2)` #绘制红色圆
- `c.create_oval(30,80,230,180,fill='yellow',width=2)` #绘制黄色椭圆
- `c.create_oval(80,30,180,230,fill='gray',width=2)` #绘制灰色椭圆

- 例：描绘地球绕太阳旋转的轨道。



- `from tkinter import *`
- `w=Tk()`
- `w.title('绘制地球绕太阳旋转轨道')`
- `c=Canvas(w,width=300,height=200,bg="white")` #创建画布对象
- `c.pack()`
- `c.create_oval(50,50,250,150,dash=(4,2),width=2)` #绘制椭圆轨道
- `c.create_oval(110,80,150,120,fill="red",outline="red")` #绘制太阳
- `c.create_oval(240,95,255,110,fill="blue")` #绘制地球

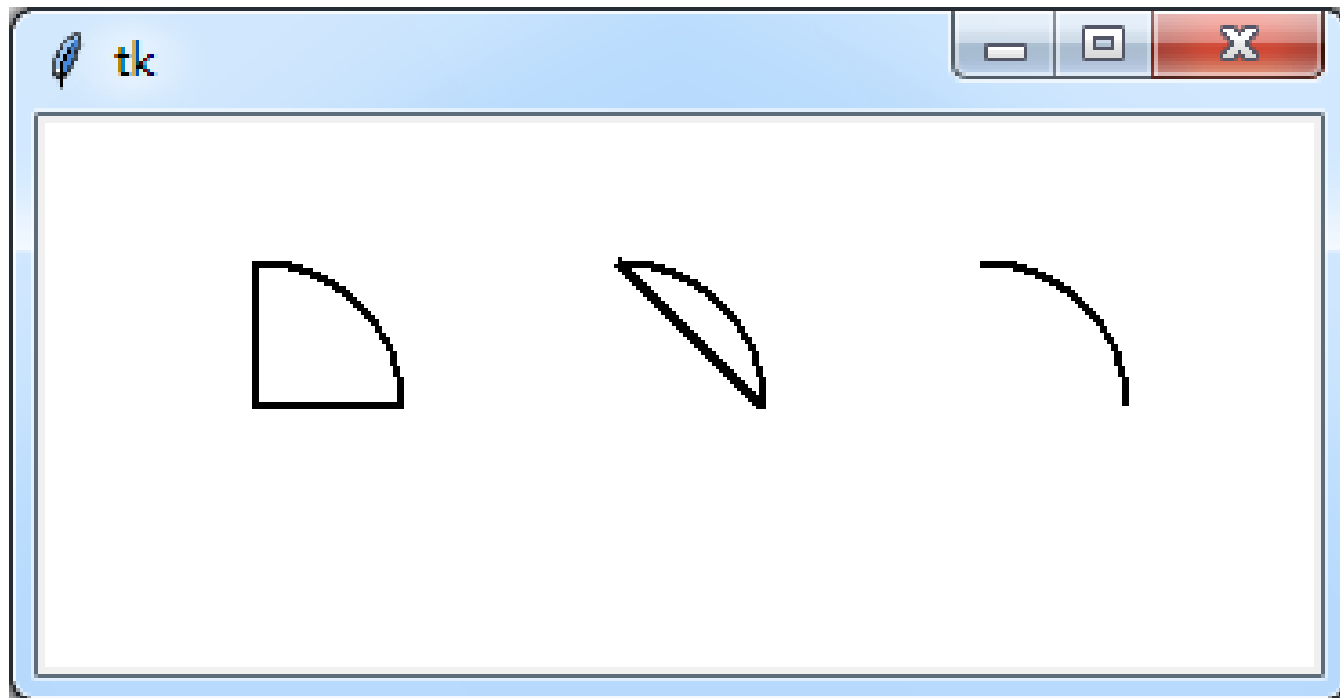
● 2. 绘制圆弧

- `create_arc()`方法用于在画布上创建一个弧形。
- 与椭圆类似，`creat_arc()`的参数是用来定义一个矩形的左上角和右下角的坐标，该矩形确定了一个内接椭圆(特例是圆)，弧形是该椭圆的一段。
- `create_arc()`方法的调用格式：
- 画布对象名.`create_arc(x0,y0,x1,y1,属性设置.....)`
- 其返回值是所创建圆弧的标识号。

- 弧形的开始位置由属性**start**定义，其值为一个角度（**x**轴正方向为 0° ）；
- 弧形的结束位置由属性**extent**定义，其值表示从开始位置逆时针旋转的角度。
- **start**属性的默认值为 0° ，**extent**属性的默认值为 90° 。
- 例：**start**设置为 0° ，**extent**设置为 360° ，则可画出一个完整的椭圆（与**create_oval()**方法相同）。

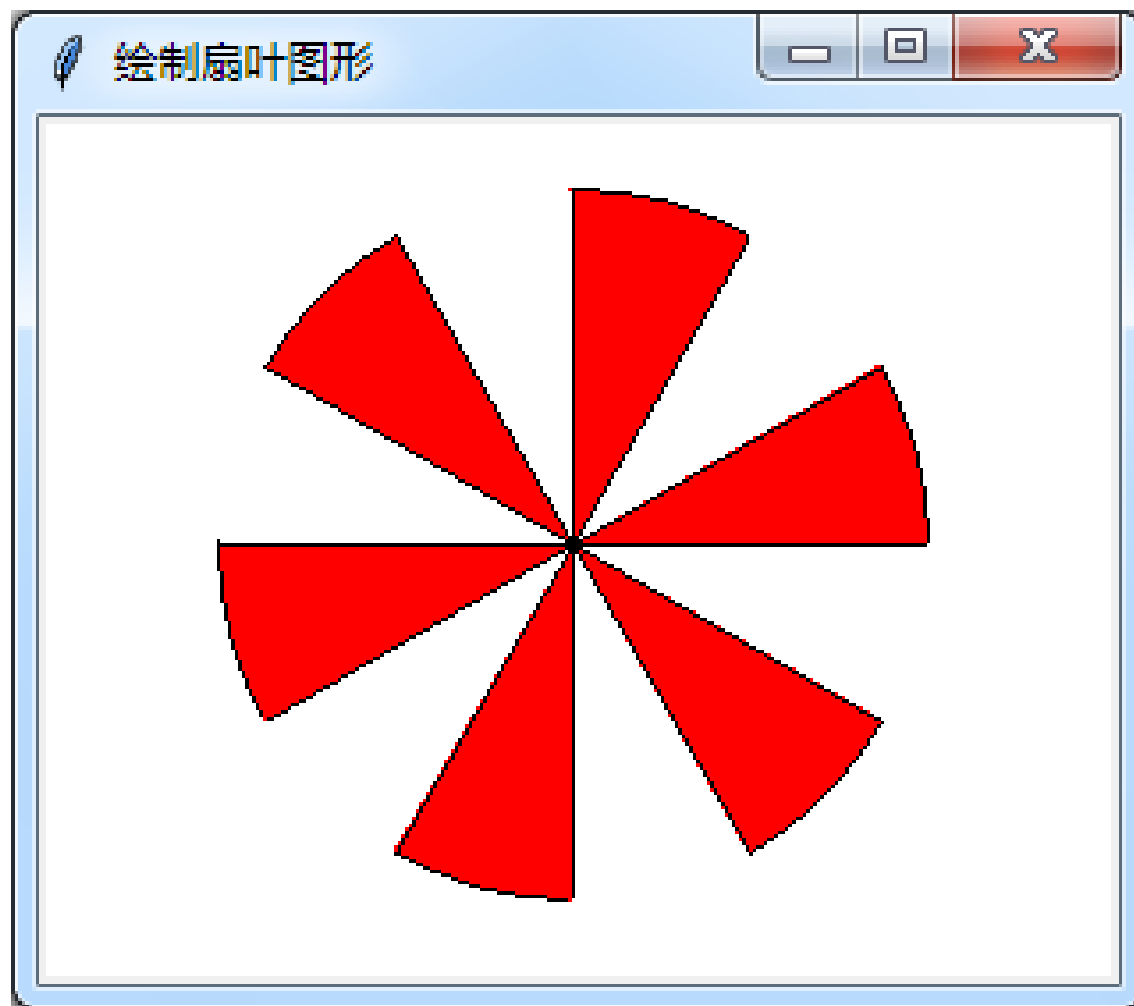
- 属性**style**用于规定圆弧的样式，可以取三种值：
- **PIESLICE**是扇形，即圆弧两端与圆心相连；
- **ARC**是弧，即圆周上的一段；
- **CHORD**是弓形，即弧加连接弧两端的弦。
- **Style**的默认值是**PIESLICE**。

- 例:
- `from tkinter import *`
- `w=Tk()`
- `c=Canvas(w,width=350,height=150,bg="white")` #创建画布对象
- `c.pack()`
- `c.create_arc(20,40,100,120,width=2)` ##默认样式是PIESLICE
- `c.create_arc(120,40,200,120,style=CHORD,width=2)`
- `c.create_arc(220,40,300,120,style=ARC,width=2)`



- 弧形的其他常用属性**outline**、**width**、**dash**、**fill**、**state**和**tags**的意义和默认值都和矩形类似。
- 注意：
- 只有**PIESLICE**和**CHORD**形状才可填充颜色。
- 画布对象的**itemconfig()**、**delete()**和**move()**方法同样可用于弧形的属性设置、删除和移动。

- 例：创建扇叶图形。



- **from tkinter import ***
- **w=Tk()**
- **w.title('绘制扇叶图形')**
- **c=Canvas(w,width=300,height=240,bg='white')**
- **c.pack()**
- **for i in range(0,360,60):**
- **c.create_arc(50,20,250,220,fill='red',start=i,extent=30)**

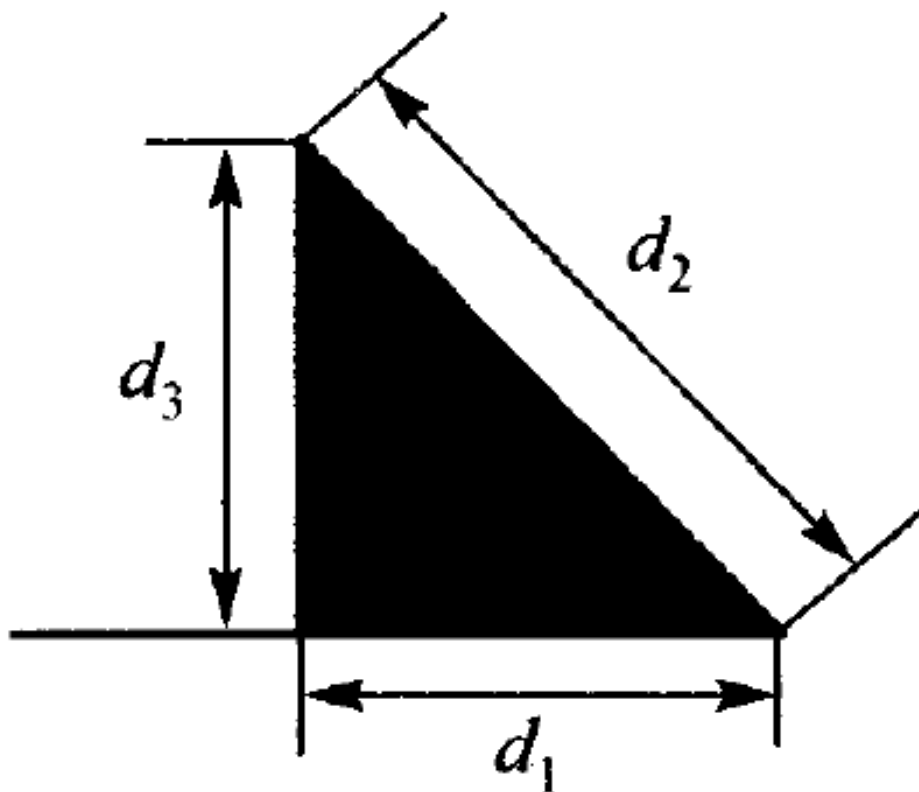
12.2.3 绘制线条与多边形

- 1. 绘制线条
- 画布对象提供`create_line()`方法，用于在画布上创建连接多个点的线段序列，其调用格式：
- 画布对象名.`create_line(x0,y0,x1,y1,属性设置.....)`
- `create_line()`方法将各点 (x_0,y_0) ， (x_1,y_1) ，.....， (x_n,y_n) 按顺序用线条连接起来，其返回值是所创建的线条的标识号。

- 若没有特别说明，则相邻两点间用直线连接，即图形整体上是一条折线。
- 但如果将属性**smooth**设置成非0值，则各点被解释成B样条曲线的顶点，图形整体是一条平滑的曲线。
- 线条不能形成边框和内部区域两个部分，因此没有**outline**属性，只有**fill**属性，表示线条的颜色，其默认值为黑色。

- 可以通过属性**arrow**来设置线条箭头，该属性的默认值是**NONE**（无箭头）。
- 如果将**arrow**设置为**FIRST**，则箭头在**(x0,y0)**端；
- 设置为**LAST**，则箭头在**(xn,yn)**端；
- 设置为**BOTH**，则两端都有箭头。

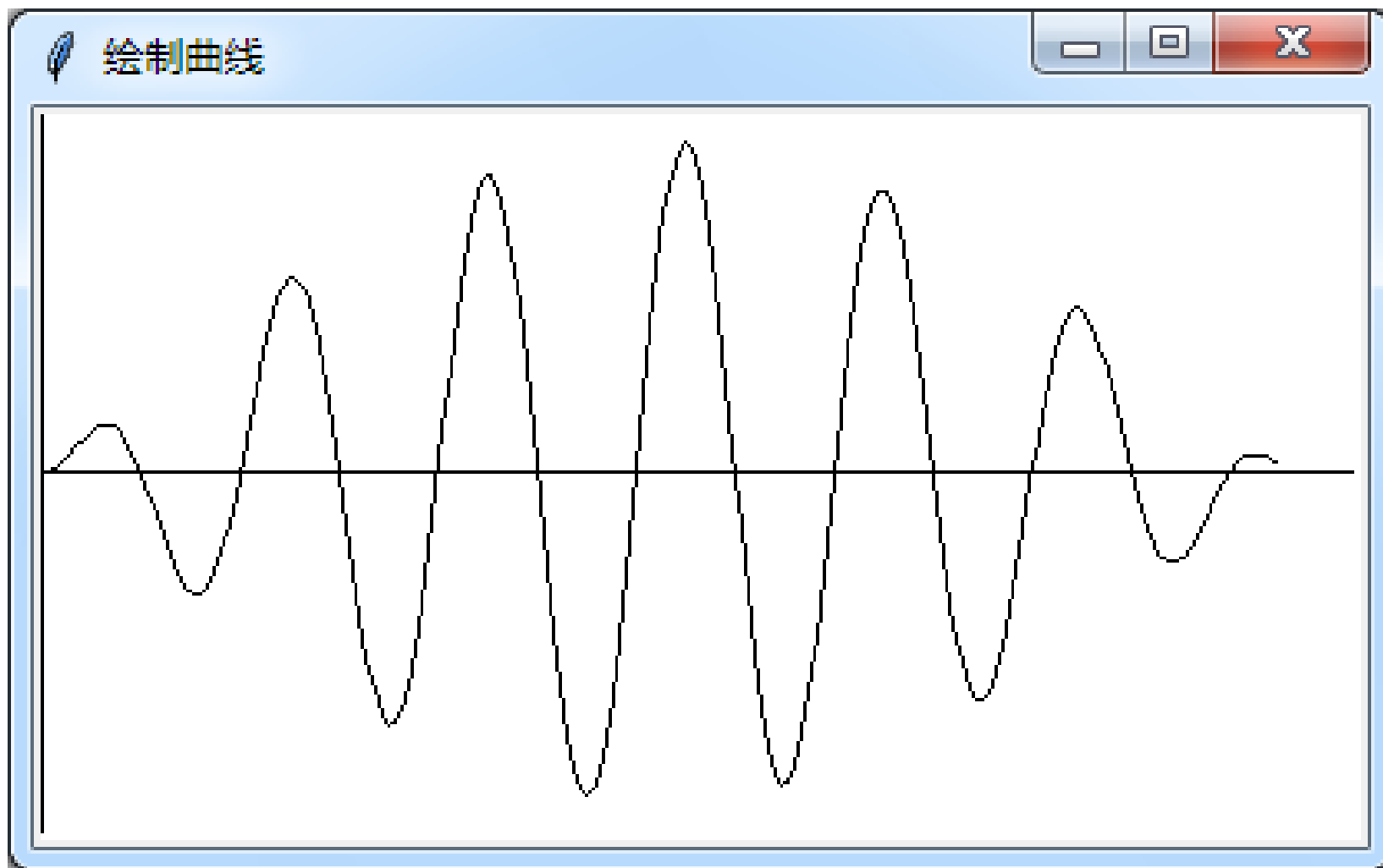
- 属性arrowshape用于描述箭头形状，其值为三元组(d_1, d_2, d_3)。默认值为(8,10,3)。



arrowshape 属性取值

- 线条还具有**width**、**dash**、**state**、**tags**等属性。
- 画布对象的**itemconfig()**方法、**delete()**方法和**move()**方法同样可用于线条的属性设置、删除和移动。

- 例：绘制 $y=\sin x \sin(4\pi x)$ 曲线。



- `from math import *`
- `from tkinter import *`
- `w=Tk()`
- `w.title('绘制曲线')`
- `W=400;H=220` #画布宽度、高度
- `O_X=2;O_Y=H/2` #x, y圆点, 窗口左边中心
- `S_X=120;S_Y=100` #x, y轴缩放倍数
- `x0=y0=0;` #坐标初始值
- `c=Canvas(w,width=W,height=H,bg='white')` #创建画布对象
- `c.pack()`
- `c.create_line(0,O_Y,W,O_Y)` #绘制x轴
- `c.create_line(O_X,0,O_X,H)` #绘制y轴
- `for i in range(0,180,1):`
 - `arc=pi*i/180`
 - `x=O_X+arc*S_X`
 - `y=O_Y-sin(arc)*sin(4*pi*arc)*S_Y`
 - `c.create_line(x0,y0,x,y)`
 - `x0=x;y0=y`

● 2. 绘制多边形

- **create_polygon()**方法，用于在画布上创建一个多边形。
- 与线条类似，多边形是用一系列顶点(至少三个)的坐标定义的，系统将把这些顶点按次序连接起来。
- 与线条不同的是，最后一个顶点需要与第一个顶点连接，从而形成封闭的形状。

- **create_polygon()**方法的调用格式:
- 画布对象名.**create_polygon(x0,y0,x1,y1,.....,属性设置.....)**
- 其返回值是创建多边形的标识号。

- 与矩形类似，**outline**和**fill**分别设置多边形的边框颜色和内部填充色；但
- 与矩形不同的是，多边形的**outline**属性默认值为空串，即边框不可见，而**fill**属性的默认值为黑色。
- 与线条类似，一般用直线连接顶点，但如果将属性**smooth**设置成非0值，则表示用B样条曲线连接顶点，这样绘制的是由平滑曲线围成的图形。

- 多边形的属性**width**、**dash**、**state**和**tags**的用法与矩形类似。
- 画布对象的**itemconfig()**方法、**delete()**方法和**move()**方法同样可用于多边形的属性设置、删除和移动。

- 例：用红、黄、绿三种颜色填充矩形。



- `from tkinter import *`
- `w=Tk()`
- `w.title('三种颜色填充矩形')`
- `c=Canvas(w,width=340,height=200,bg='white')`
- `c.pack()`
- `c.create_rectangle(50,50,290,150,width=5)` #绘制矩形
- `c.create_polygon(50,50,50,150,130,150,fill="red")` #绘制红色三角形
- `c.create_polygon(50,50,130,150,290,150,210,50,\`
- `fill="yellow")` #绘制黄色平行四边形
- `c.create_polygon(210,50,290,150,290,50,fill="green")` #绘制绿色三角形

12.2.4 显示文本与图像

- 1. 显示文本
- `create_text()`方法用于在画布上显示一行或多行文本。
- 与普通的字符串不同，这里的文本被作为图形对象。

- `create_text()`方法的调用格式:
- 画布对象名.`create_text(x,y,属性设置.....)`
- 其中, `(x,y)`指定文本显示的参考位置。
- 其返回值是所创造的文本的标识号。

- 文本内容由text属性设置，其值就是要显示的字符串。
- 字符串中可以使用换行字符“\n”，从而实现多行文本的显示。
- anchor属性用于指定文本的哪个锚点与显示位置(x, y)对齐。

- 文本有一个边界框，**tkinter**模块为边界框定义了若干个锚点，锚点用**E**（东）、**S**（南）、**W**（西）、**N**（北）、**CENTER**（中）、**SE**（东南）、**SW**（西南）、**NW**（西北）、**NE**（东北）等方位常量表示。
- 通过锚点可以控制文本的相对位置。

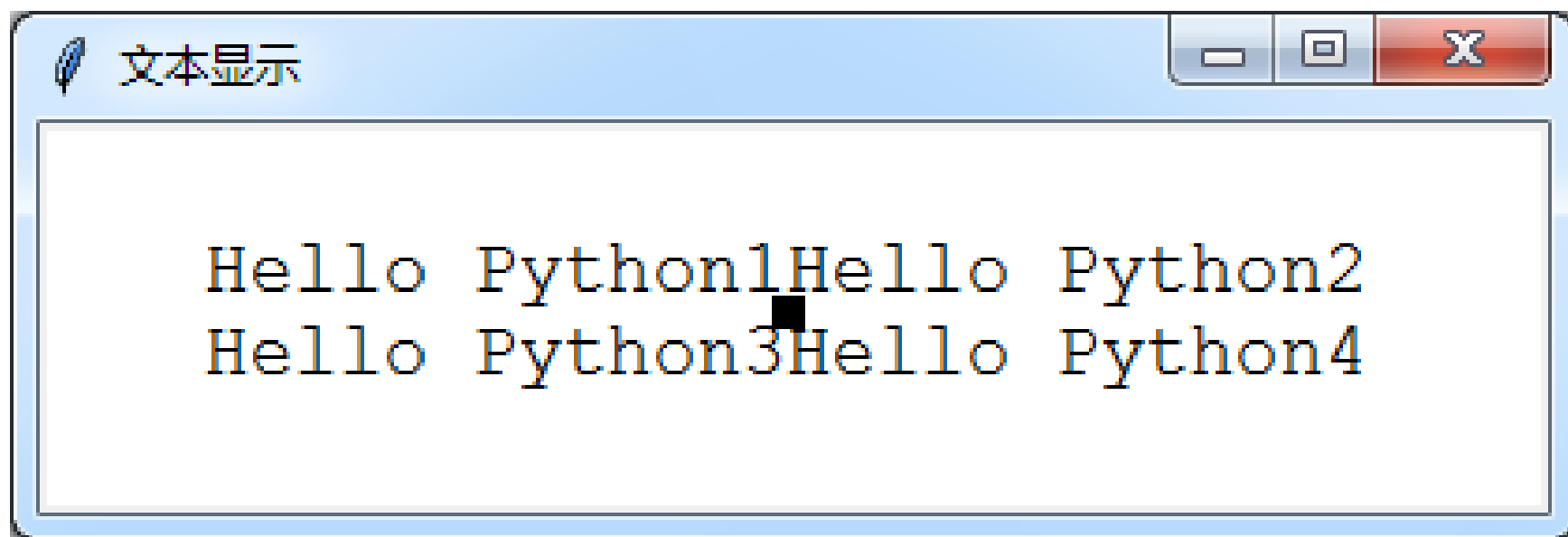
- 例：若将**anchor**设置为**N**，则将文本边界框的顶边中点置于参考点(x,y)；
- 若将**anchor**设置为**SW**，则将文本边界框的左下角置于参考点(x,y)。
- **anchor**的默认值为**CENTER**，表示将文本的中心置于参考点(x,y)。

- **fill**属性用于设置文本的颜色，默认值为黑色。如果设置为空串，则文本不可见。
- **justify**属性用于控制多行文本的对齐方式，其值为**LEFT**、**CENTER**或**RIGHT**，默认值为**LEFT**。
- **width**属性用于控制文本的宽度，超出宽度就要换行。

- **font**属性指定文本字体。
- 字体描述使用一个三元组，包含字体名称、大小和字形名称。
- 例：
- **("Times New Roman",10,"bold")**表示10号加黑新罗马字；
- **("宋体",12,"italic")**表示12号斜体宋体。

- **state**属性、**tags**属性的意义与其他图形对象相同。
- 画布对象的**itemcget()**和**itemconfig()**方法可用于读取或修改文本的内容。
- 画布对象的**delete()**方法、**move()**方法可用于文本的删除和移动。

- 例：画布文本显示示例（用锚点控制文本显示位置）。



- `from tkinter import *`
- `w=Tk()`
- `w.title('文本显示')`
- `c=Canvas(w,width=400,height=100,bg="white")`
- `c.pack()`
- `c.create_rectangle(200,50,201,51,width=8)` #显示文本参考位置
- `c.create_text(200,50,text="Hello Python1",\`
- `font=("Courier New",15,"normal"),anchor=SE)` #右下对齐
- `c.create_text(200,50,text="Hello Python2",\`
- `font=("Courier New",15,"normal"),anchor=SW)` #左下对齐
- `c.create_text(200,50,text="Hello Python3",\`
- `font=("Courier New",15,"normal"),anchor=NE)` #右上对齐
- `c.create_text(200,50,text="Hello Python4",\`
- `font=("Courier New",15,"normal"),anchor=NW)` #左上对齐

- **2. 显示图像**

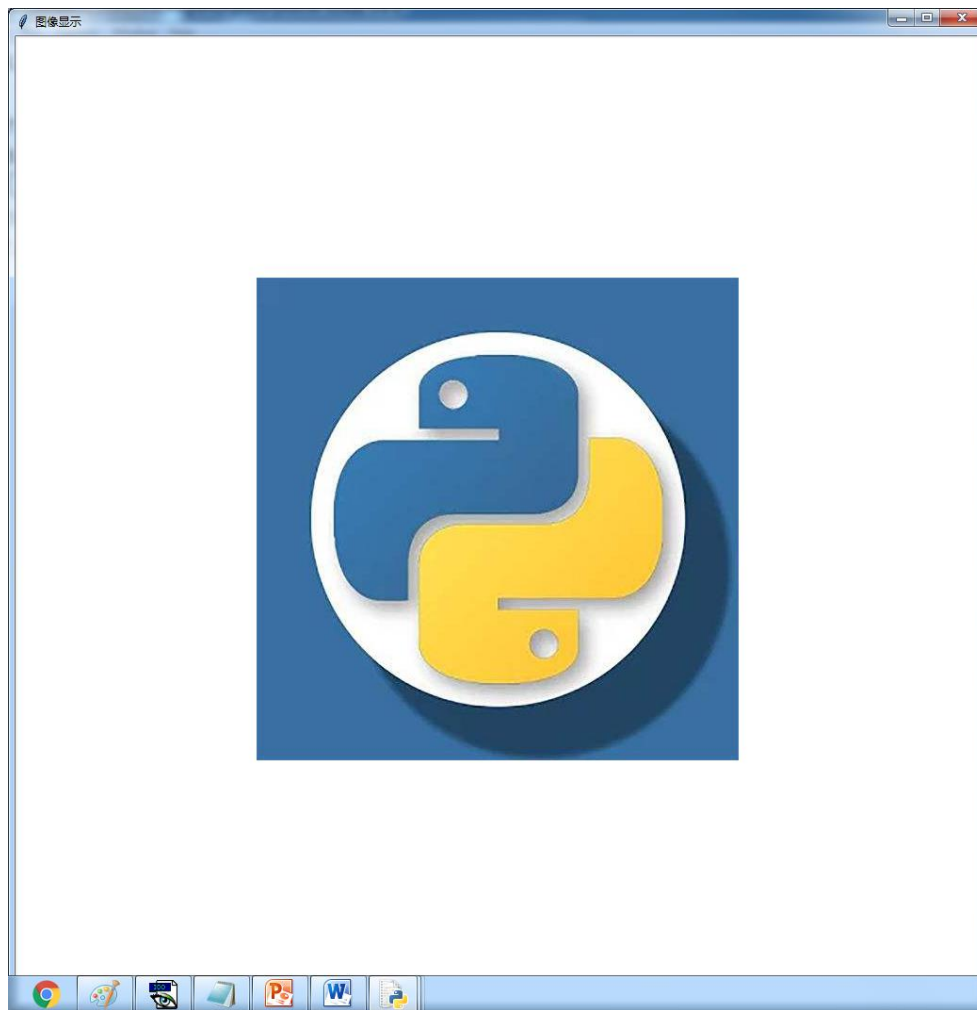
- 除了在画布上绘图外，还可以将现成的图像显示在画布上。
- **tkinter**模块针对不同格式的图像文件有不同的显示方法。
- 显示gif格式图像的具体步骤：

- （1）利用tkinter模块提供的PhotoImage类来创建图像对象。
- 语句格式：
- 图形对象=PhotoImage(file=图像文件名)
- 其中，属性file用于指定图像文件（支持gif、png、bmp、pgm、ppm等格式），PhotoImage()返回值是一个图像对象。

- (2) 通过画布对象提供的create_image()方法在画布上显示图像。
- 方法的调用格式:
- 画布对象名.create_image(x,y,image=图像对象,属性设置.....)
- 其中, (x,y)是决定图像显示位置的参考点。
- 其返回值是所创建的图像在画布上的标识号。

- 图像在画布上的位置由参考点(x,y)和anchor属性决定，具体设置与文本相同。
- 可以为图像设置属性state、tags，意义与其他图形对象相同。
- 画布对象的delete()方法、move()方法同样可用于图像的删除和移动。

- 例：假设有图像文件e:\a.png，请将该图像显示在画布中。



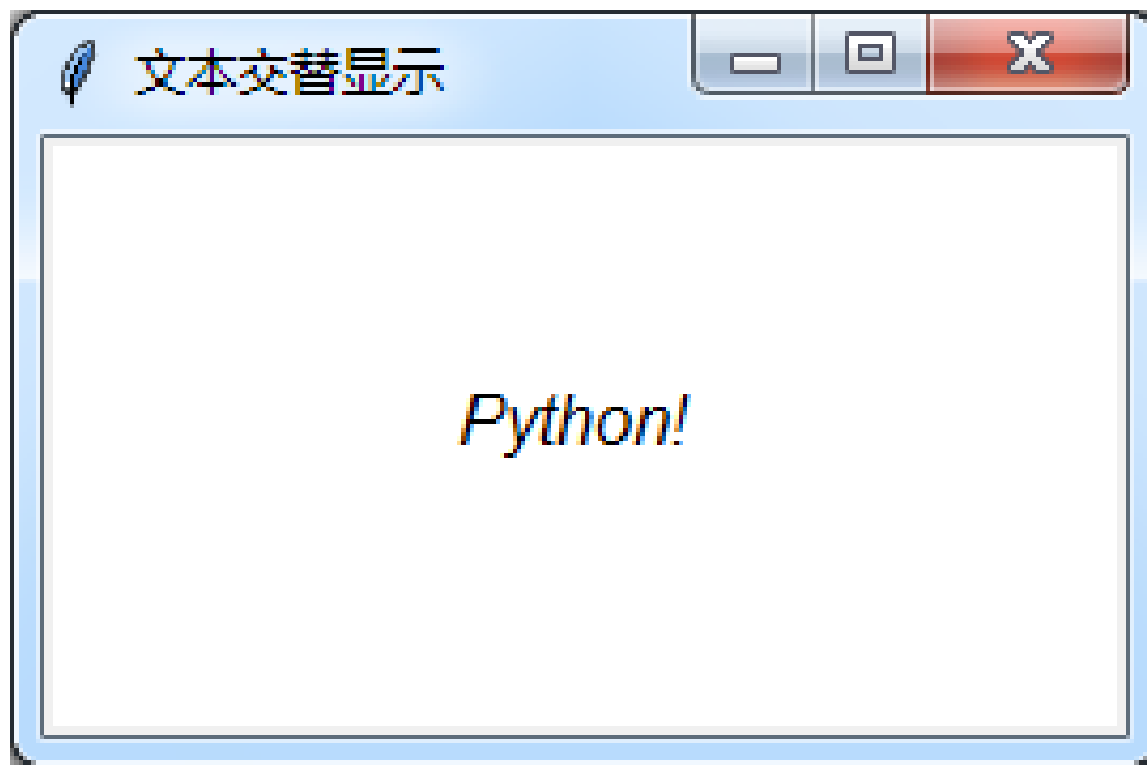
- **from tkinter import ***
- **w=Tk()**
- **w.title('图像显示')**
- **c=Canvas(w,width=300,height=150,bg='white')**
- **c.pack()**
- **pic=PhotoImage(file="e:\\mypython\\python.png")**
- **c.create_image(150,75,image=pic)**

12.3 图形的事件处理

- 所谓事件（**event**）是指在程序执行过程中发生的操作，例如单击了鼠标左键、按下了键盘上的某个键等。
- 某个对象可以与特定事件绑定在一起，这样当特定事件发生时，可以调用特定的函数来处理这个事件。

- 画布及画布上的图形都是对象，都可以与操作事件绑定，这样用户可以利用键盘、鼠标来操作、控制画布和图形。

- 例：在画布轮换交替显示两行文本，鼠标左键单击文本时替换一次，右键单击文本时隐藏文本，鼠标指针指向文本时使文本随机移动。



- **def canvasF(event):**
- **if c.itemcget(t,"text")== "Python!":**
- **c.itemconfig(t,text="Programming!")**
- **else:**
- **c.itemconfig(t,text="Python!")**
- **def textF1(event):**
- **c.move(t,randint(-10,10),randint(-10,10))**
- **def textF2(event):**
- **if c.itemcget(t,"fill")!= "white":**
- **c.itemconfig(t,fill="white")**
- **else:**
- **c.itemconfig(t,fill="black")**

- **from tkinter import ***
- **from random import ***
- **w=Tk()**
- **w.title('文本交替显示')**
- **c=Canvas(w,width=250,height=150,bg="white")**
- **c.pack()**
- **t=c.create_text(125,75,text="Python!",font=("Arial",12,"italic"))**
- **c.bind("<Button-1>",canvasF)**
- **c.tag_bind(t,"<Enter>",textF1)**
- **c.tag_bind(t,"<Button-3>",textF2)**
- **w.mainloop()**

- 说明:
- `c.bind("<Button-1>",canvasF)`
- 利用画布的**bind()**方法将画布对象与鼠标左键单击事件**<Button-1>**进行绑定，当用户在画布对象上单击鼠标左键时，就去执行函数**canvasF()**。
- `c.tag_bind(t,"<Enter>",textF1)`
- 利用画布的**tag_bind()**方法将画布上的文本对象**t**与鼠标指针进入事件**<Enter>**进行绑定，当鼠标指针指向文本**t**时，就去执行函数**textF1()**。

- 说明:
- `c.tag_bind(t,"<Button-3>",textF2)`
- 利用画布的`tag_bind()`方法将画布对象上的文本对象`t`与鼠标右键单击事件`<Button-3>`进行绑定，当用户在文本`t`上单击鼠标右键时，就去执行函数`textF2()`。

- 说明：（三个事件处理函数的功能）
- **canvasF()**的功能是改变文本**t**的内容。
- **textF1()**的功能是随机移动文本，移动的距离用一个随机函数来产生。
- **textF2()**的功能是改变文本的颜色。

- 说明：
- `w.mainloop()`
- 该语句的作用是进入主窗口的事件循环。
- 系统将自行监控在主窗口上发生的各种事件，并触发相应的处理函数。

12.4 turtle绘图与Graphics图形库

- Python的图形库有很多，除了Tkinter图形库之外，常用的还有turtle绘图与Graphics图形库。

12.4.1 turtle绘图

- 利用turtle模块绘图通常称为海龟绘图。
- 绘图时有一个箭头（比作小海龟），按照命令一笔一笔地画出图形，就像小海龟在屏幕上爬行，并能留下爬行的足迹，于是就形成了图形。
- 海龟是绘图的画笔，屏幕是用来绘图的纸张。

- **1. 导入turtle模块**
- 使用**turtle**绘图,首先需要导入**turtle**模块,有以下两种方法:
- **>>> import turtle**
- **>>> from turtle import ***

● 2. turtle绘图属性

- turtle绘图有三个要素，分别是位置、方向和画笔。
- （1）位置是指箭头在Turtle图形窗口中的位置。
- turtle图形窗口的坐标系采用笛卡儿坐标系，即以窗口中心点为原点，向右为x轴正方向，向上为y轴正方向。
- 在turtle模块中，可用reset()函数让箭头回到坐标原点。

- (2) 方向是指箭头的指向，使用`left(degree)`、`right(degree)`函数使得箭头分别向左、向右旋转degree度。
- (3) 画笔是指绘制的线条的颜色和宽度，有关画笔控制函数如下：
- `down()`：放下画笔，移动时绘制图形。这也是默认的状态。
- `up()`：提起画笔，移动时不绘制图形。

- **pensize(w)**或**width(w)**: 绘制图形时画笔的宽度, **w**为一个正数。
- **pencolor(s)**或**color(s)**: 绘制图形时画笔的颜色, **s**是一个字符串, 例: 'red'、'blue'、'green'分别表示红色、蓝色、绿色。
- **fillcolor(s)**: 绘制图形的填充颜色。

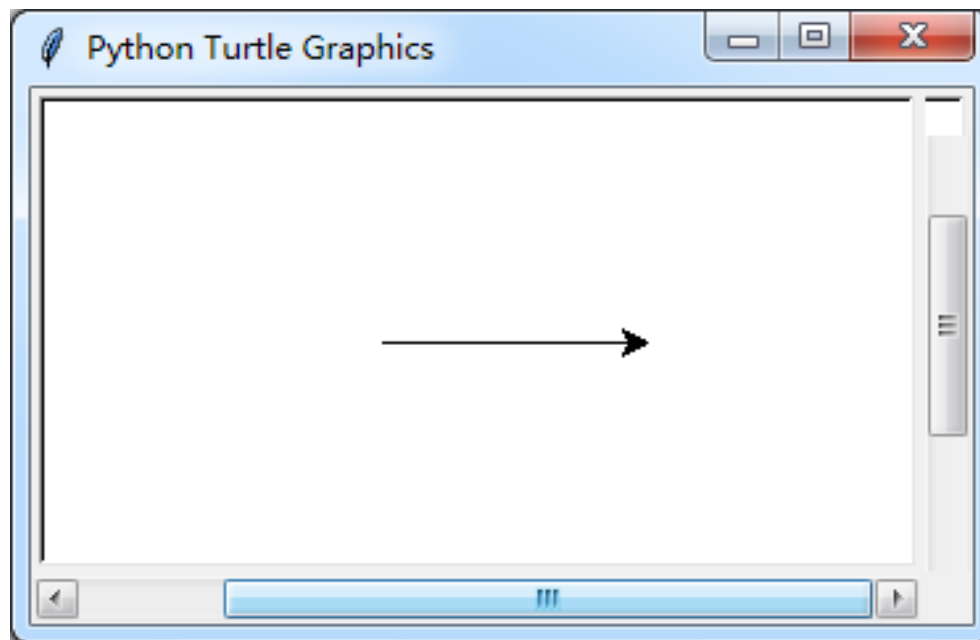
- **3. turtle绘图命令**

- **turtle**绘图有许多控制箭头运动的命令。

- **goto(x,y)**: 将箭头从当前位置径直移动到坐标为(x,y)的位置, 这时当前方向不起作用, 移动后方向也不改变。

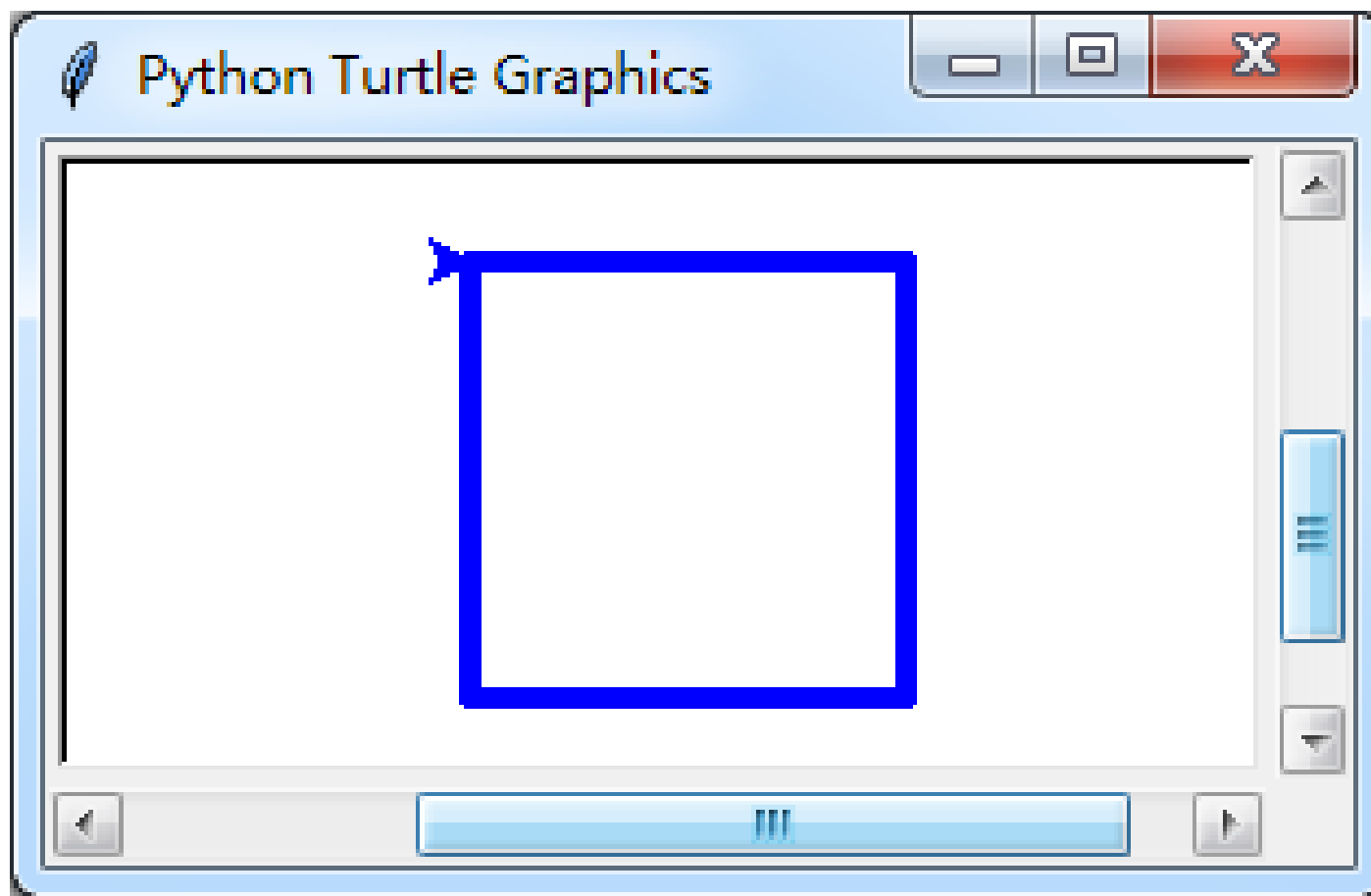
- 如果想要移动箭头到(x,y)处, 但不绘制图形, 可以使用**up()**函数。

- 例：绘制一根水平直线。
- `from turtle import *`
- `reset()`#将整个绘图窗口清空并将箭头置于原点
(窗口的中心)
- `goto(100,0)`#从当前位置(0,0)移动到(100,0)位置



- **forward(d)**: 控制箭头向前移动，其中d代表移动的距离。在移动前，需要设置箭头的位置、方向和画笔三个属性。
- **backward(d)**: 与**forward(d)**函数相反，控制箭头向后移动，其中d代表移动的距离。
- **speed(v)**: 控制箭头移动的速度，v取[0,10]范围的整数，数字越大，速度越快。
- 也可以使用'slow'、'fast'来控制速度。

- 例：绘制一个正方形。



- **from turtle import ***
- **color("blue")** #定义绘制时画笔的颜色
- **pensize(5)** #定义绘制时画笔的线条宽度
- **speed(10)** #定义绘图的速度
- **for i in range(4):** #绘出正方形的四条边
 - **forward(100)**
 - **right(90)**

- **turtle**模块还有一些内置函数，例如画圆的函数 **circle(r)**，该函数以箭头当前位置为圆的底部坐标，以r为半径画圆。

- 例：绘制三个同心圆。
- **from turtle import ***
- **for i in range(3):**
- **up()** **#提起画笔**
- **goto(0,-50-i*50)** **#确定画圆的起点**
- **down()** **#放下画笔**
- **circle(50+i*50)** **#画圆**


```
from turtle import *  
pencolor("blue") #定义画笔的颜色  
pensize(2)       #定义画笔的线条宽度  
speed(10)        #定义绘图的速度  
for i in range(10):  
    up()          #提起画笔  
    goto(0,-10-i*10) #确定画笔的起点  
    down()        #放下画笔  
    circle(10+i*10) #画圆
```

```
from turtle import *  
  
pencolor("blue") #定义画笔的颜色  
  
pensize(2)       #定义画笔的线条宽度  
  
speed(10)        #定义绘图的速度  
  
for i in range(50): #绘出正n边形的n条边  
    forward(4+i*4)  
    right(360//4)
```

```
from turtle import *
pencolor("blue") #定义画笔的颜色
pensize(2)       #定义画笔的线条宽度
speed(10)        #定义绘图的速度
n=4
for i in range(20): #绘出正n边形的n条边
    up()           #提起画笔
    goto(-i*n*2,i*n*2) #确定画笔的起点
    down()         #放下画笔
    for j in range(n):
        forward(4+i*n*4)
        right(360//n)
```

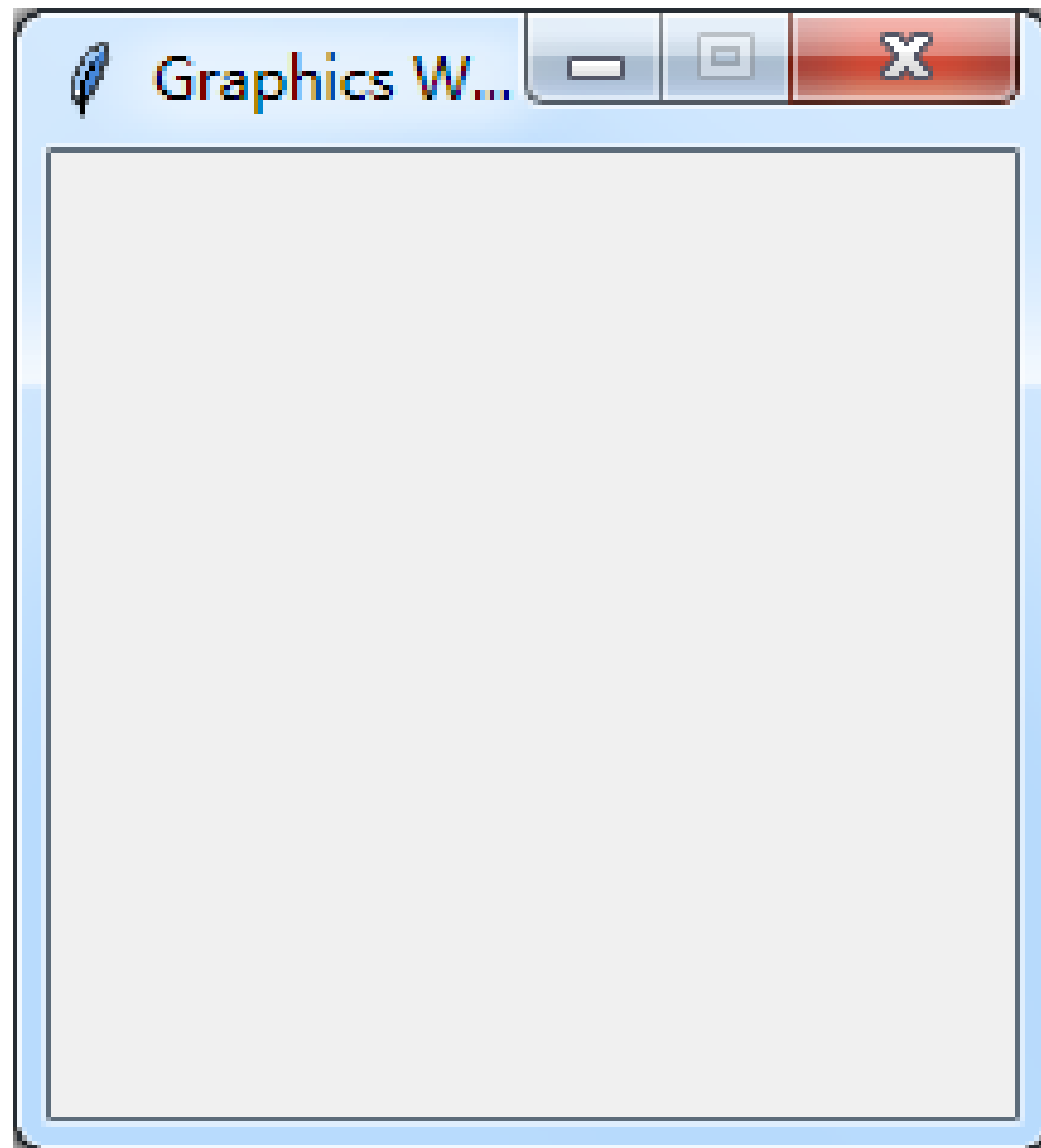
12.4.2 Graphics图形库

- **Graphics**图形库是在**Tkinter**图形库基础上建立的，由**graphics**模块组成。
- **graphics**模块的所有功能都是依赖于**tkinter**模块功能实现的。
- **graphics**模块将**tkinter**模块的绘图功能以面向对象的方式重新包装，更容易学习和应用。

- **1. 模块导入与图形窗口**
- **graphics**模块文件（**graphics.py**）可以从网站 <http://mcsp.wartburg.edu/zelle/python> 下载，下载后将**graphics.py**文件与用户自己的图形程序放在一个目录中，或者放在**Python**安装目录中即可。

- 使用**graphics**绘图，首先要导入**graphics**模块。
- 语句格式有两种：
- **>>> import graphics**
- **>>> from graphics import ***

- 使用graphics提供的**GraphWin()**函数创建一个图形窗口。
- 在图形窗口中，设有标题栏，以及“最小化”、“最大化”、“关闭”等按钮。
- 例：
- **>>> win=GraphWin()**



- **GraphWin()**函数在屏幕上创建了一个图形窗口，默认窗口标题是“**Graphics Window**”，默认宽度和高度都是**200**像素。
- 图形窗口的坐标系与画布对象的坐标系相同，仍以窗口左上角为原点，**x**轴向右为正方向，**y**轴向下为正方向。
- **graphics**图形窗口也有各种属性，在调用**GraphWin()**函数时可以提供各种参数。

- 例：
- **>>>win=GraphWin("My Graphics Window",300,200)**
- 窗口标题为 “My Graphics Window”， 宽度为300像素， 高度为200像素。

- 通过**GraphWin**类创建图形窗口的界面实际上是对**tkinter**模块中创建画布对象界面的重新包装，即，当利用**graphics**模块创建图形窗口时，系统会把这个请求向下传递给**tkinter**模块，而**tkinter**模块就创建一个画布对象并返回给上层的**graphics**模块。
- 这样做可以使得图形处理更加容易和理解。

- 为了对图形窗口进行操作，可以建立一个窗口对象win，以后就可以通过窗口对象win对图形窗口进行操作。
- 例：窗口操作结束后应该关闭图形窗口，关闭窗口的函数调用方法为：
- **>>> win.close()**

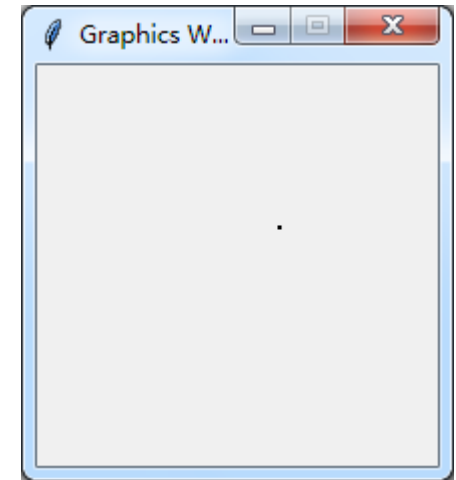
● 2. 图形对象

- 在tkinter模块中，只为画布提供了Canvas类，而画布上绘制的各种图形并没有对应的类。
- 画布是对象，而画布的图形并不是对象，不是按面向对象的风格构造的。
- graphics模块就是为了改进这一点而设计的。

- 在graphics模块中，提供了**GraphWin**（图形窗口）、**Point**（点）、**Line**（直线）、**Circle**（圆）、**Oval**（椭圆）、**Rectangle**（矩形）、**Polygon**（多边形）、**Text**（文本）等类，利用类可以创建相应的图形对象。
- 每个对象都是相应的类的实例，对象都具有自己的属性和方法（操作）。

- (1) 点
- graphics模块提供了point类，用于在窗口中画点。
- 创建点对象的语句格式为：
- `p=Point(x坐标,y坐标)`

- 例:
- `>>>from graphics import *`
- `>>>win=GraphWin()`
- `>>>p=Point(100,50)`
- `>>>p.draw(win)`
- `>>>print(p.getX(),p.getY())`
- 100 50
- `>>>p.move(20,30)`
- `>>>print(p.getX(),p.getY())`
- 120,80



100.0 50.0
120.0 80.0

- **Point**对象的其他方法：
- **p.setFill()**: 设置点**p**的颜色。
- **p.setOutline()**: 设置边框的颜色。对**Point**对象来说，与**setFill()**方法没有区别。
- **p.undraw()**: 隐藏对象**p**，即在图形窗口中，对象**p**变成不可见。注意，隐藏并非删除，对象**p**仍然存在，随时可以重新执行**draw()**。
- **p.clone()**: 复制一个与**p**一模一样的对象。

- 除了用字符串指定颜色之外，**graphics**模块还提供了**color_rgb(r,g,b)**函数来设置颜色，其中的**r**，**g**，**b**参数取**0~255**之间的整数，分别表示红色、绿色、蓝色的数值，**color_rgb()**函数表示的颜色就是三种颜色混合以后的颜色。
- 例：
- **color_rgb(255,0,0)**表示亮红色；
- **color_rgb(0,255,0)**表示亮绿色。

- **(2) 直线**

- 直线类**Line**用于绘制直线。创建直线对象的语句格式为:

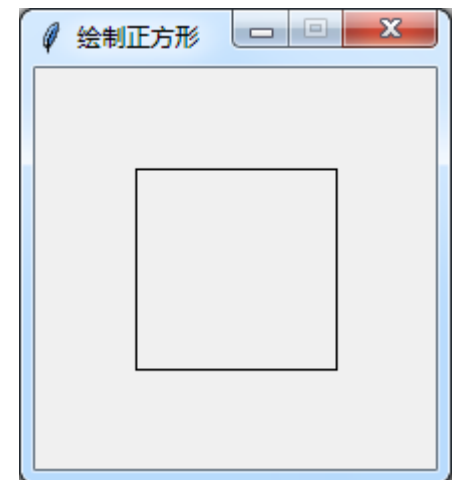
- **line=Line(端点1,端点2)**

- 其中，两个端点都是**Point**对象。

- 和**Point**对象一样，**Line**对象也支持**draw()**、**undraw()**、**move()**、**setFill()**、**setOutline**、**clone()**等方法。

- **Line**对象还支持**setArrow()**方法，用于为直线画箭头，**setWidth()**方法用于设置直线宽度。

- 例：利用直线对象绘制一个正方形。
- `from graphics import *`
- `win=GraphWin("绘制正方形")`
- `p1=Point(50,50);p2=Point(150,50)`
- `p3=Point(150,150);p4=Point(50,150)`
- `l1=Line(p1,p2);l2=Line(p2,p3)`
- `l3=Line(p3,p4);l4=Line(p4,p1)`
- `l1.draw(win);l2.draw(win)`
- `l3.draw(win);l4.draw(win)`



- (3) 圆

- 圆类为**Circle**，创建圆形对象的语句格式为：

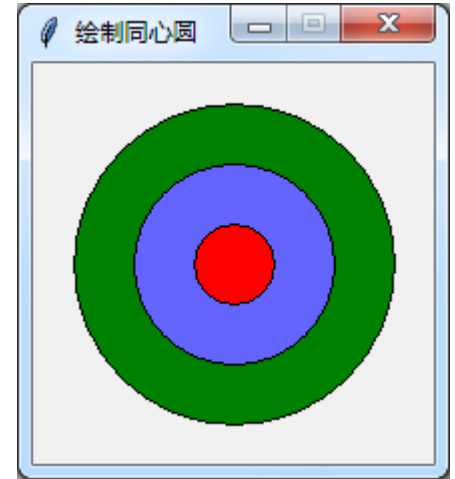
- **c=Circle(圆心,半径)**

- 其中，圆心是**Point**对象，半径是个数值。

- **Circle**对象支持**draw()**、**undraw()**、**setFill()**、**setOutline**、**clone()**、**setWidth()**等方法。

- **Circle**对象还支持**c.getRadius()**方法，用于获取圆形对象**c**的半径。

- 例：绘制三个同心圆，并且将它们填充不同颜色。
- `from graphics import *`
- `win=GraphWin("绘制同心圆")`
- `pt=Point(100,100)`
- `cir1=Circle(pt,80);`
- `cir1.draw(win);cir1.setFill('green')`
- `cir2=Circle(pt,50);`
- `cir2.draw(win);cir2.setFill(color_rgb(100,100,255))`
- `cir3=Circle(pt,20);`
- `cir3.draw(win);cir3.setFill(color_rgb(255,0,0))`



- (4) 椭圆

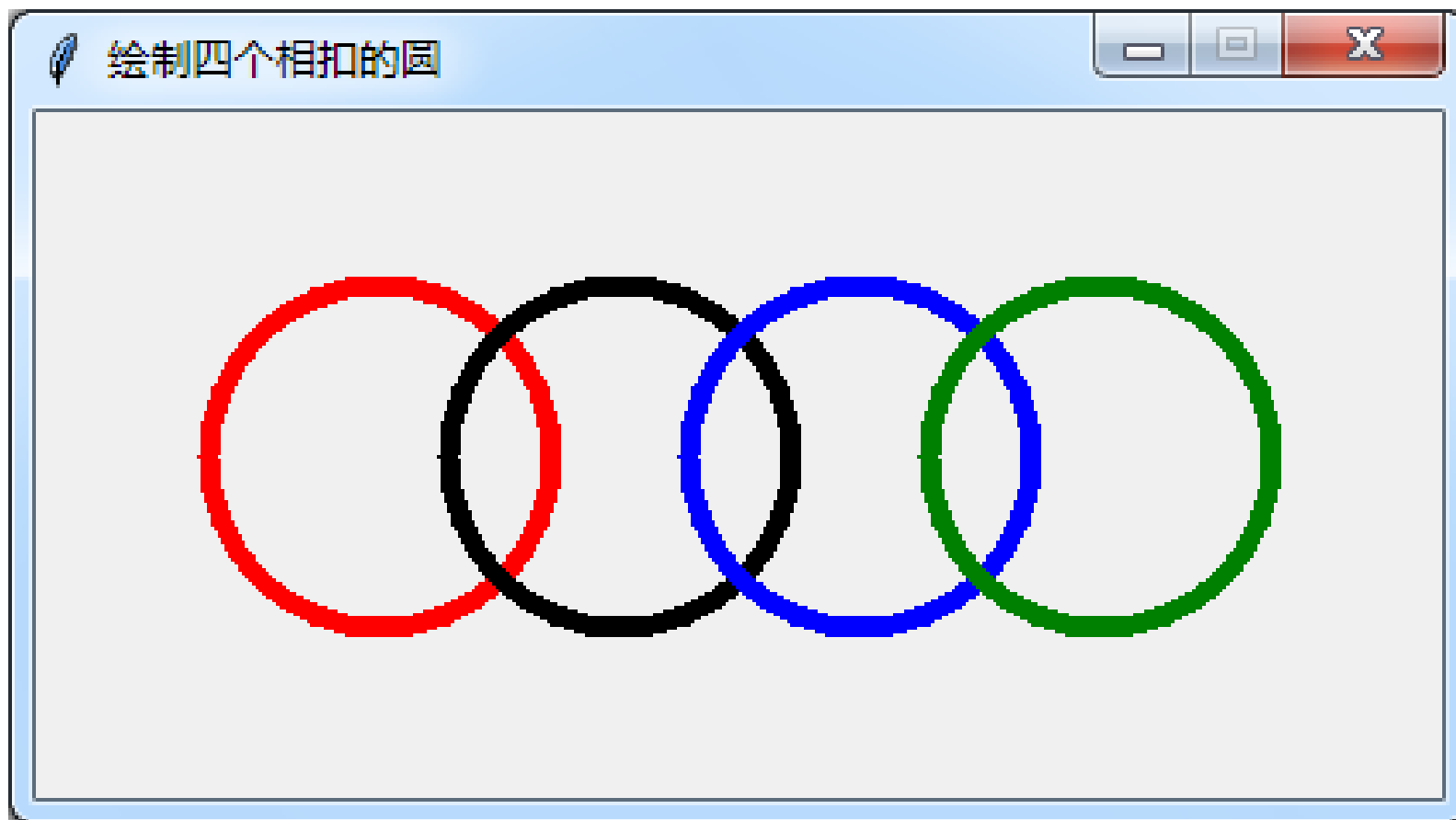
- 椭圆类为Oval，创建椭圆对象的语句格式为：

- o=Oval(左上角,右下角)

- 其中，左上角和右下角是两个Point对象，用于指定一个矩形，再由这个矩形定义一个内接椭圆。

- 椭圆对象支持draw()、undraw()、move()、setFill()、setOutline()、clone()、setWidth()等方法。

- 例：绘制四个相扣的圆，并且将它们的边线设置成不同颜色，边线宽度相同。



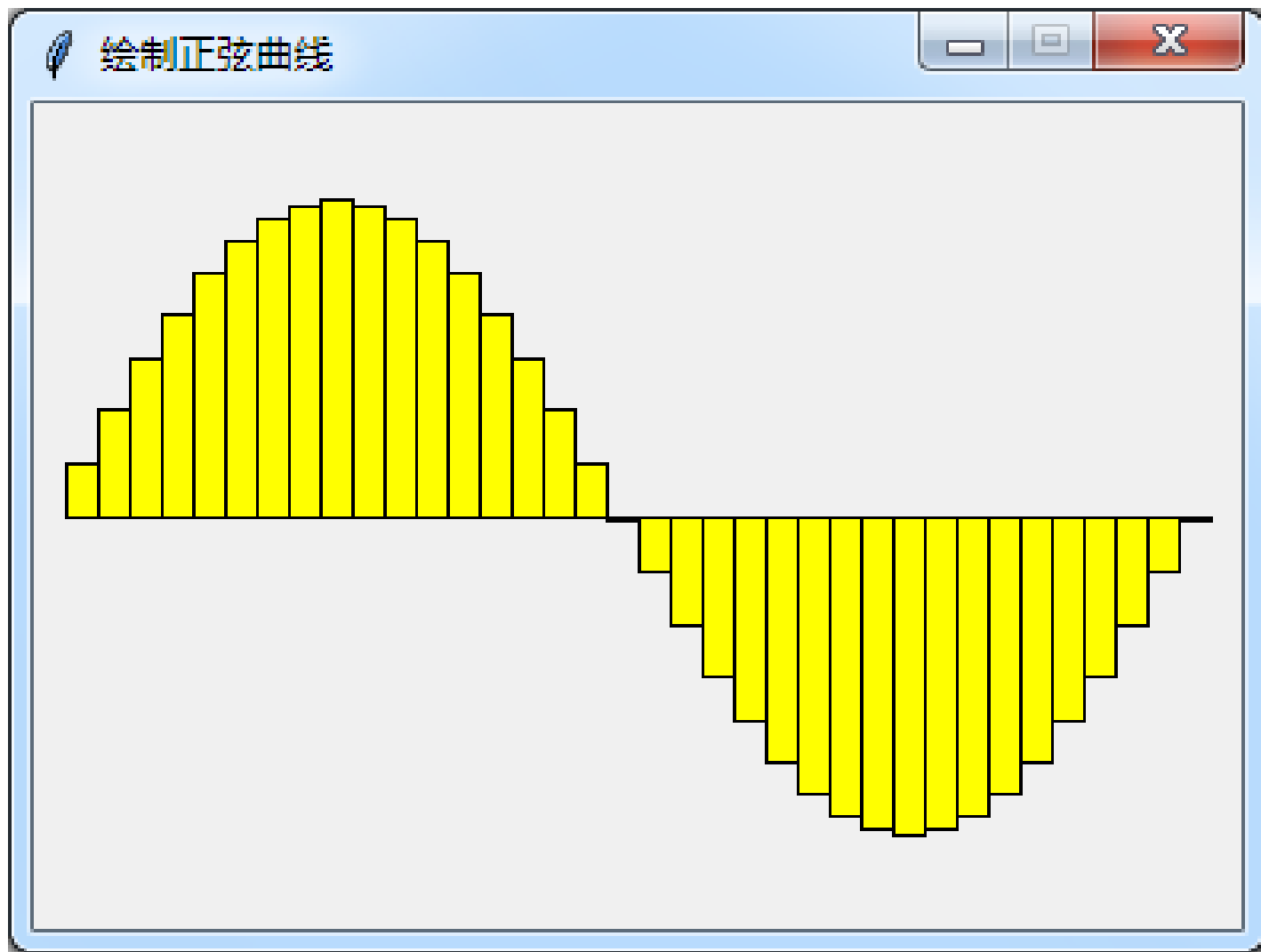
- **from graphics import ***
- **win=GraphWin('绘制四个相扣的圆',410,200)**
- **pt1=Point(50,50);pt2=Point(150,150)**
- **o1=Oval(pt1,pt2);o1.draw(win);**
- **o1.setOutline('red');o1.setWidth(6)**
- **o2=o1.clone(); #复制相同的圆对象**
- **o2.draw(win);o2.move(70,0);o2.setOutline('black');o2.setWidth(6)**
- **o3=o2.clone();**
- **o3.draw(win);o3.move(70,0);o3.setOutline('blue');o3.setWidth(6)**
- **o4=o3.clone();**
- **o4.draw(win);o4.move(70,0);o4.setOutline('green');o4.setWidth(6)**

- (5) 矩形

- 矩形类为**Rectangle**，创建矩形对象的语句格式为：
- **r=Rectangle(左上角,右下角)**
- 其中，左上角和右下角是两个**Point**对象，用于指定矩形。
- 矩形对象支持**draw()**、**undraw()**、**move()**、**setFill()**、**setOutline()**、**clone()**、**setWidth()**等方法。

- 矩形还支持的方法有**r.getP1()**、**r.getP2()**和**r.getCenter()**，分别用于获取左上角、右下角和中心坐标，返回值都是**Point**对象。

- 例：绘制如图所示的正弦曲线图形。

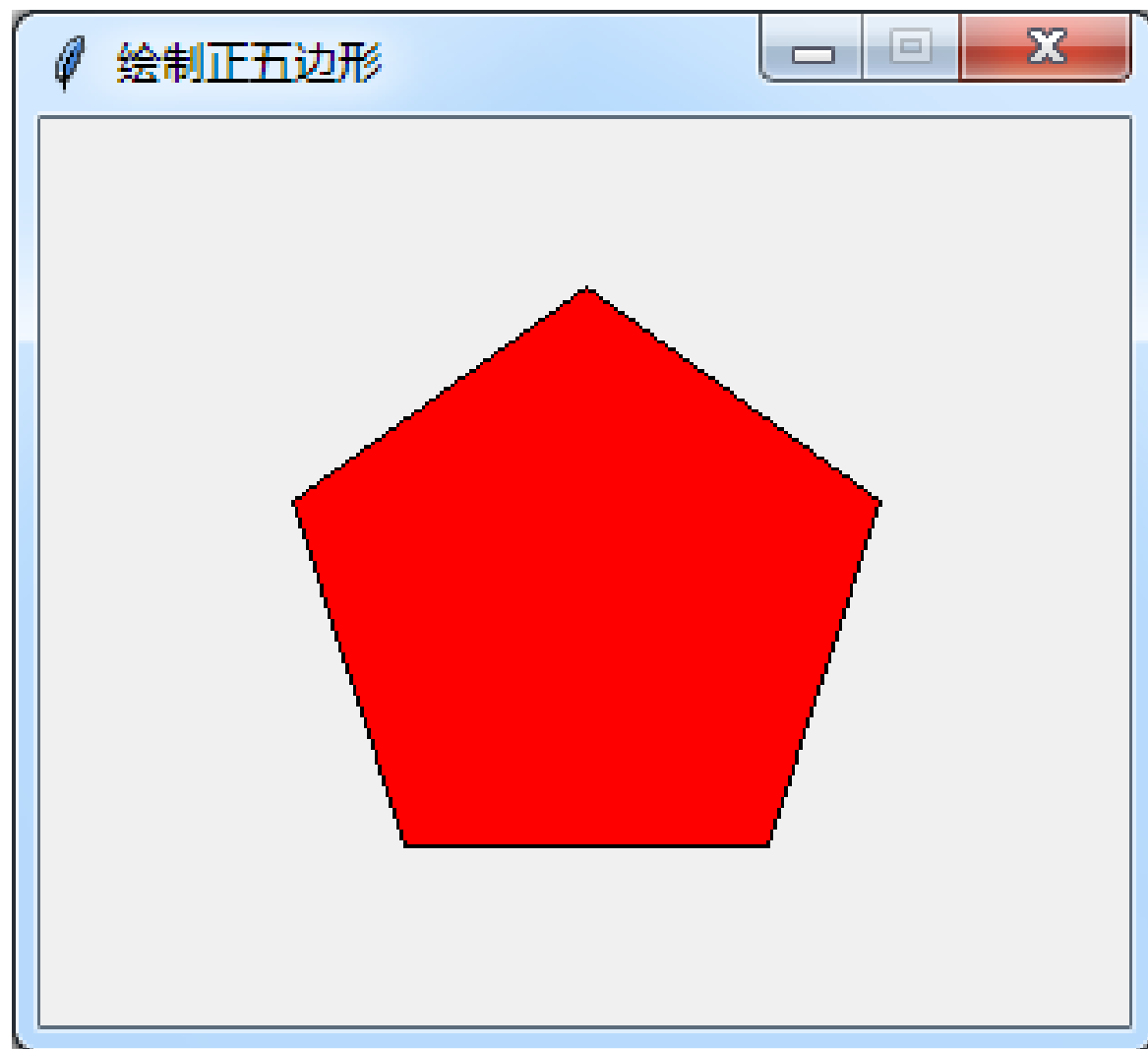


- **from graphics import ***
- **from math import ***
- **win=GraphWin('绘制正弦曲线',380,260)**
- **x=10**
- **for i in range(0,36):**
- **pt1=Point(x,-100*sin(x*pi/180)+130)**
- **pt2=Point(x+10,130)**
- **r=Rectangle(pt1,pt2)**
- **r.draw(win);r.setFill('yellow')**
- **x+=10**

- **(6) 多边形**

- 多边形类为**Polygon**，创建多边形对象的语句格式为：
- **p=Polygon(顶点1,.....,顶点n)**
- 将各顶点用直线相连，即成多边形。
- 多边形对象支持**draw()**、**undraw()**、**move()**、**setFill()**、**setOutline()**、**clone()**、**setWidth()**等方法。
- 还支持方法**poly.getPoints()**，用于获取多边形的各个顶点坐标。

- 例：绘制红色的正五边形。



- `from graphics import *`
- `from math import *`
- `win=GraphWin('绘制正五边形',300,250)`
- `p1=Point(100,200)`
- `p2=Point(200,200)`
- `p3=Point(200+100*cos(pi*72/180),200-100*sin(pi*72/180))`
- `p4=Point(100+50,200-50/sin(pi*36/180)-50/tan(pi*36/180))`
- `p5=Point(100-100*cos(pi*72/180),200-100*sin(pi*72/180))`
- `p=Polygon(p1,p2,p3,p4,p5)`
- `p.draw(win);p.setFill('red');`

- (7) 文本

- 文本类为**Text**，创建文本对象的语句格式为：

- **t=Text(中心点,字符串)**

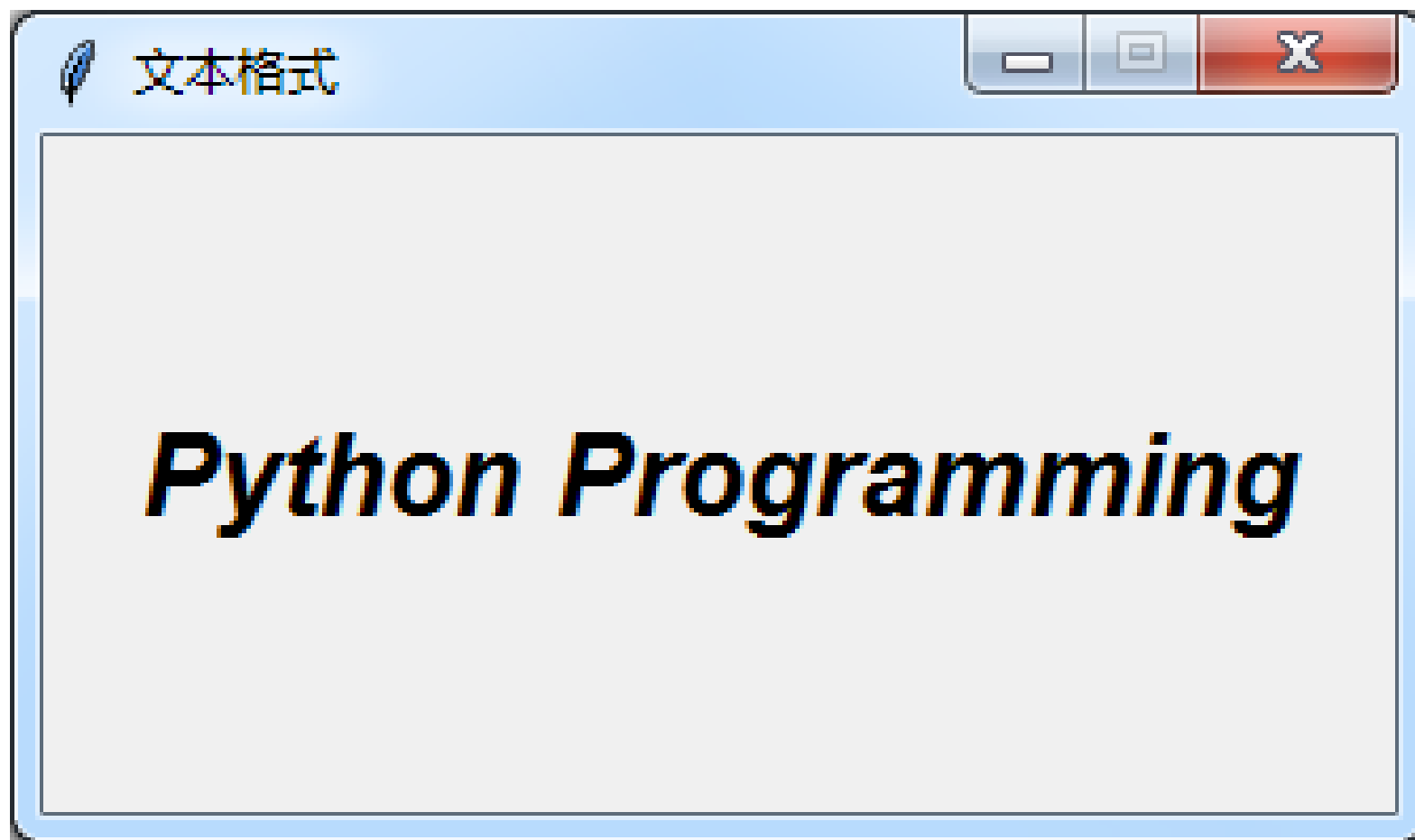
- 其中，中心点是个**Point**对象，字符串是显示的文本内容。

- 文本对象支持**draw()**、**undraw()**、**move()**、**setFill()**、**setOutline()**、**clone()**等方法，其中**setFill()**和**setoutline()**方法都是设置文本的颜色。

- 文本对象方法**t.setText(新字符串)**用于改变文本内容；
- 方法**t.getText()**用于获取文本内容；
- 方法**t.setTextColor()**用于设置文本颜色，与**setFill**效果相同
- 方法**setFace()**用于设置文本字体，可选值有 **helvetica**、**courier**、**times romm**以及**arial**；

- 方法**setSize()**用于设置字体大小、取值范围为**5-36**;
- 方法**setStyle()**用于设置字体风格, 可选值有 **normal**、**bold**、**italic**以及**bold italic**;
- 方法**getAnchor()**用于返回文本显示中间位置点（锚点）的坐标值。

- 例：文本格式示例。



- **from graphics import ***
- **from math import ***
- **win=GraphWin('文本格式',320,160)**
- **p=Point(160,80)**
- **t=Text(p,'Python Programming')**
- **t.draw(win)**
- **t.setFace('arial')**
- **t.setSize(20)**
- **t.setStyle('bold italic')**

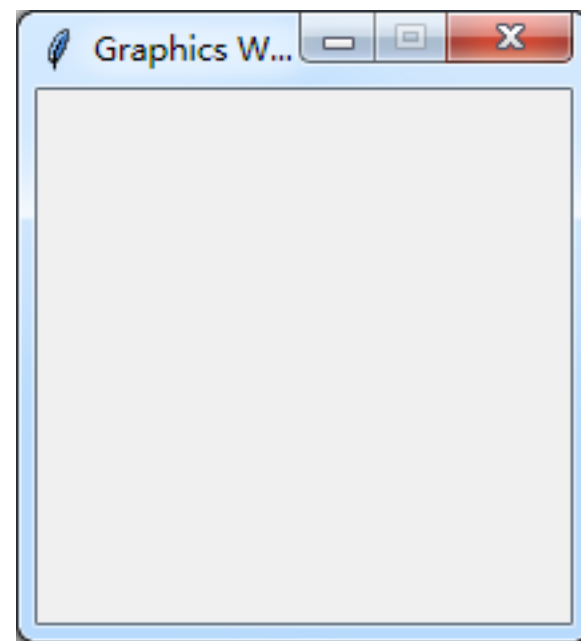
3. 交互式图形操作

- 图形用户界面可以用于程序交互式的输入和输出，用户通过单击按钮、选择菜单栏中的选项以及在屏幕文本框中输入文字等来与应用程序进行交互。
- 当用户移动鼠标、单击按钮或者从键盘输入数据时，就产生了一个事件，这个事件被发送到图形用户界面的相应对象进行处理。

- 例：单击按钮会产生一个单击事件，该事件将会传递给按钮处理代码，按钮处理代码将执行相应操作。
- **graphics**模块提供了两个简单的方法获得用户在图形界面窗口中的操作事件。

- **(1) 捕捉鼠标单击事件**
- 可以通过**GraphWin**类中的**getMouse()**方法获得用户在窗口内单击鼠标的信息。
- 当**getMouse()**方法被一个**GraphWin**对象调用时，程序将停止并等待用户在窗口内单击鼠标。
- 用户单击鼠标的位置以**Point**对象作为返回值返回给程序。

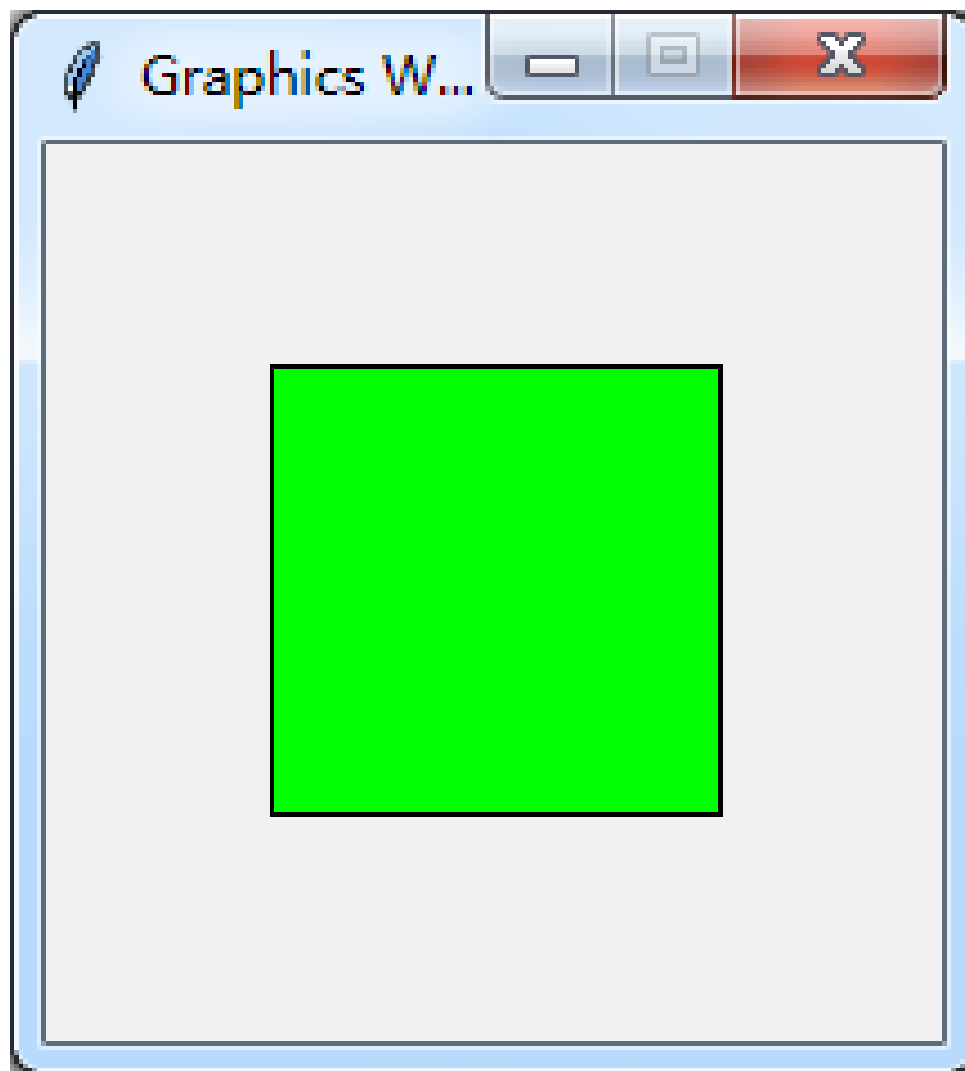
- 例:
- `from graphics import *`
- `win=GraphWin()`
- `p=win.getMouse()`
- `print(p.getX(),p.getY())`



96.0 103.0

- `getMouse()`的返回值是一个**Point**对象，使用该对象的**getX()**和**getY()**方法可以得到单击鼠标的坐标。

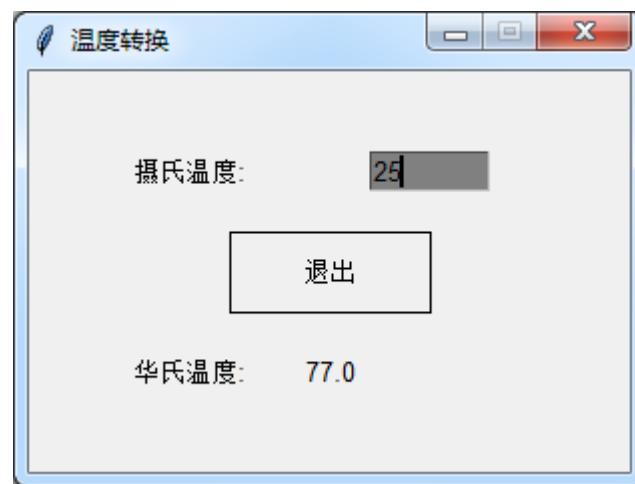
- 例：在窗口单击鼠标时绘制一个绿色的正方形。



- **from graphics import ***
- **win=GraphWin()**
- **p1=win.getMouse()**
- **p2=Point(p1.getX()+100,p1.getY()+100)**
- **r=Rectangle(p1,p2)**
- **r.draw(win)**
- **r.setFill('green1')**

- (2) 处理文本输入
- **graphics**模块还包括一个输入对象**Entry**，用于获取窗口中的键盘输入事件。
- **Entry**对象在图形窗口中创建一个文本框，它与**Text**对象类似，也使用**setText()**和**getText()**方法。
- 不同之处在于**Entry**对象的内容可以被用户修改。

- 例：创建一个图形窗口，其中有一个输入框，用于输入摄氏温度值，同时提供一个“温度转换”按钮，单击按钮时能够将摄氏温度转换为华氏温度，同时“温度转换”变为“退出”按钮，单击按钮退出图形窗口。



- `from graphics import *`
- `win=GraphWin("温度转换",300,200)`
- `t1=Text(Point(80,50),"摄氏温度:")`
- `t1.setSize(10);t1.draw(win)`
- `t2=Text(Point(80,150),"华氏温度:")`
- `t2.setSize(10);t2.draw(win)`
- `input=Entry(Point(200,50),8)`
- `input.setText("0")`
- `input.setSize(10);input.draw(win)`
- `output=Text(Point(150,150),"")`
- `output.draw(win)`
- `button=Text(Point(150,100),"温度转换")`
- `button.setSize(10);button.draw(win)`
- `Rectangle(Point(100,80),Point(200,120)).draw(win)`
- `win.getMouse()` #等待鼠标单击
- `celsius=eval(input.getText())` #输入温度值
- `fahrenheit=9/5*celsius+32`
- `output.setText(fahrenheit)` #显示输出
- `output.setSize(10);`
- `button.setText("退出")` #改变按钮提示
- `win.getMouse()` #等待响应鼠标单击，退出程序
- `win.close()`

- 说明：
- 在程序中，按钮仅起到了提示和修饰作用，实际上在窗口的任意位置单击鼠标都能够进行温度转换操作。
- 程序中并没有保存用户单击鼠标的位置，**getMouse()**方法只是用于暂停程序，使用户可以在输入框中输入温度值。

12.5 图形应用举例

- 在Python环境下绘制图形有三种方法：**tkinter**绘图、**turtle**绘图和**graphics**绘图。

12.5.1 验证Fibonacci数列的性质

设 f_n 表示数列的第 n 项，Fibonacci 数列有如下性质：

$$f_1^2 + f_2^2 + f_3^2 + \cdots + f_n^2 = f_n \times f_{n+1}$$

取 $n=8$ ，Fibonacci 数列前 8 项是 1, 1, 2, 3, 5, 8, 13, 21，

这时有 $1^2+1^2+2^2+3^2+5^2+8^2+13^2+21^2=21 \times 34$ 。

可以直观地表示成图。

图中有 8 个正方形，各个正方形的边长分别是 Fibonacci 数列前 8 项，

前 8 项的平方和即图中 8 个正方形的面积和，

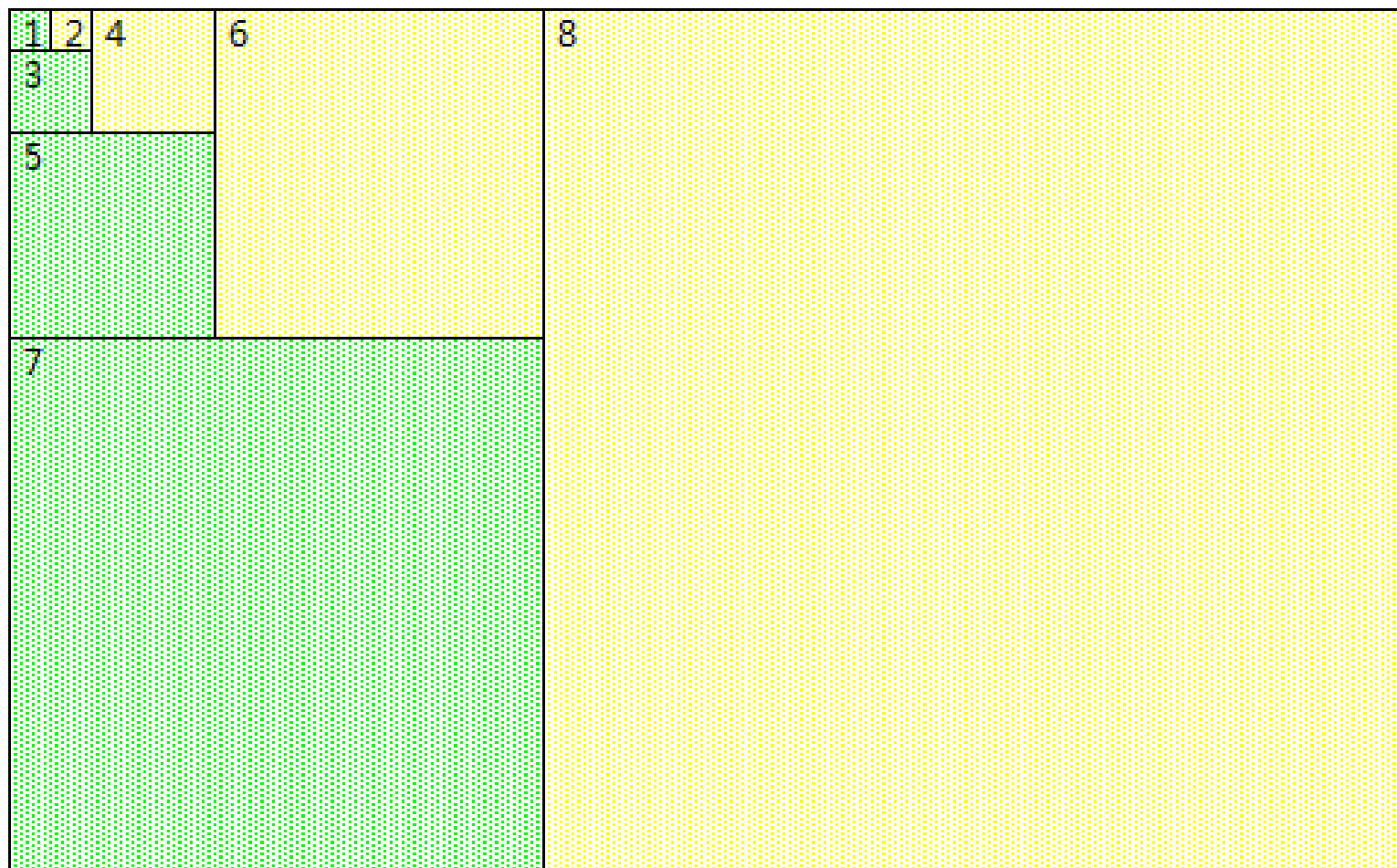
显然该和等于 $f_8 \times f_9$ ，即 21×34 。

- 例：取n=8，用图解法验证Fibonacci数列的性质：

$$f_1^2 + f_2^2 + f_3^2 + \cdots + f_8^2 = f_8 \times f_9$$

分析：要用图解法验证Fibonacci数列的性质，只要画出图即可。绘图时，主要是要确定各个正方形的坐标，坐标与Fibonacci数列各项的值有关。为便于处理，先求Fibonacci数列各项并存入一个列表中。

Fibonacci数列的性质



- `from tkinter import *`
- `w=Tk()`
- `w.title('Fibonacci数列的性质')`
- `c=Canvas(w,width=530,height=330,bg='white')`
- `c.pack()`
- `def fib(n): #求Fibonacci数列前n项`
- `L=[0]`
- `a,b=0,1`
- `for i in range(n):`
- `L.append(b)`
- `a,b=b,a+b`
- `return L`
- `lst=fib(8) #将Fibonacci数列的前8项保存在列表lst中`
- `x,y,d=25,22,14`
- `for i in range(1,len(lst)):`
- `if i%2==1:`
- `a,b=x,y+d*lst[i-1] #左上角坐标`
- `col='green1'`
- `else:`
- `a,b=x+d*lst[i-1],y #左上角坐标`
- `col='yellow'`
- `cc,dd=a+d*lst[i],b+d*lst[i] #右下角坐标`
- `c.create_rectangle(a,b,cc,dd,fill=col,stipple='gray25')`
- `c.create_text(a+8,b+8,text=str(i)) #标注序号`

12.5.2 统计图表

- 统计图表也称为统计图（**chart**），它以图形方式来呈现数据之间的关系与发展趋势，包括柱形图、折线图、饼图等。

- 例：按百分制输入学生考试成绩（输入负数时结束输入），统计各等级分数的人数（成绩 ≥ 90 为优秀，成绩 ≥ 80 但 < 90 为良好，成绩 ≥ 70 但 < 80 为中等，成绩 ≥ 60 但 < 70 为及格，成绩 < 60 为不及格），并用饼图来直观地给出各等级人数的比例。

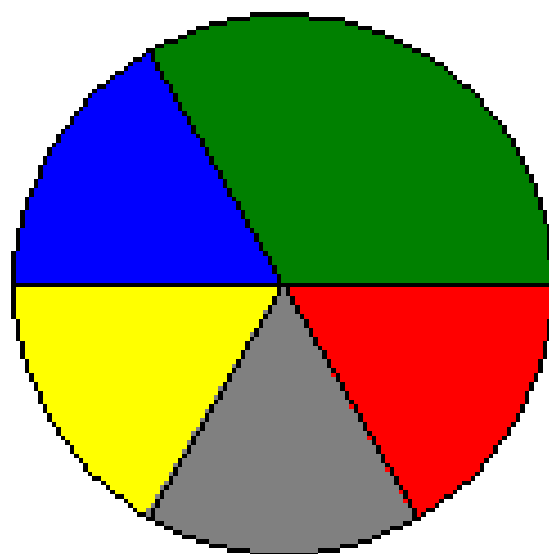
- 在tkinter模块中，需要先创建主窗口和画布，然后利用画布的`create_arc()`方法绘制代表五个等级的五个扇形。
- 扇形的角度反映了各分数等级的比例。扇形具有不同填充色以相互区分。为了显示各扇形对应的等级，还需要绘制图例。
- 最后，标出各分数等级所占的比例，这里采用的方法为，当用户将鼠标指针移入某个扇形中时，画布上就显示该扇形所代表的比例值。








各等级成绩比例分布



优秀 33.3%



-  优秀
-  良好
-  中等
-  及格
-  不及格

```

from tkinter import *
from math import *
def getGrades():
    n1,n2,n3,n4,n5=0,0,0,0,0
    grade=int(input("输入成绩:"))
    while grade>=0: #求各等级人数
        if grade>=90:
            n1+=1
        elif grade>=80:
            n2+=1
        elif grade>=70:
            n3+=1
        elif grade>=60:
            n4+=1
        else:
            n5+=1
    grade=int(input("输入成绩:"))
    return n1,n2,n3,n4,n5
def main():
    try:
        w=Tk()
        w.title("各等级成绩比例分布")
        c=Canvas(w,width=320,height=200,bg="white")
        c.pack()
        n1,n2,n3,n4,n5=getGrades()
        n=n1+n2+n3+n4+n5
        #根据各等级分数比例确定各扇区起止角
        eN1,sN1=360*n1/n,0
        eN2,sN2=360*n2/n,eN1
        eN3,sN3=360*n3/n,eN1+eN2
        eN4,sN4=360*n4/n,eN1+eN2+eN3
        eN5,sN5=360*n5/n,eN1+eN2+eN3+eN4
        p=(100,40,220,160)
        #绘制扇区
        pieN1=c.create_arc(p,start=sN1,extent=eN1,fill="green")
        pieN2=c.create_arc(p,start=sN2,extent=eN2,fill="blue")
        pieN3=c.create_arc(p,start=sN3,extent=eN3,fill="yellow")
        pieN4=c.create_arc(p,start=sN4,extent=eN4,fill="gray")
        pieN5=c.create_arc(p,start=sN5,extent=eN5,fill="red")
        #绘制图例
        c.create_rectangle(240,40,260,50,fill="green")
        c.create_rectangle(240,40+24,260,50+24,fill="blue")
        c.create_rectangle(240,40+48,260,50+48,fill="yellow")
        c.create_rectangle(240,40+72,260,50+72,fill="gray")
        c.create_rectangle(240,40+96,260,50+96,fill="red")
        c.create_text(275,40,text="优秀",anchor=N)
        c.create_text(275,40+22,text="良好",anchor=N)
        c.create_text(275,40+46,text="中等",anchor=N)
        c.create_text(275,40+70,text="及格",anchor=N)
        c.create_text(280,40+94,text="不及格",anchor=N)
        piepct=c.create_text(50,100,text="")
        #定义事件处理函数
        def PieN1(event):
            pct="%5.1f%%"%(100*n1/n)
            c.itemconfig(piepct,text="优秀"+pct)
        def PieN2(event):
            pct="%5.1f%%"%(100*n2/n)
            c.itemconfig(piepct,text="良好"+pct)
        def PieN3(event):
            pct="%5.1f%%"%(100*n3/n)
            c.itemconfig(piepct,text="中等"+pct)
        def PieN4(event):
            pct="%5.1f%%"%(100*n4/n)
            c.itemconfig(piepct,text="及格"+pct)
        def PieN5(event):
            pct="%5.1f%%"%(100*n5/n)
            c.itemconfig(piepct,text="不及格"+pct)
        c.tag_bind(pieN1,"<Enter>",PieN1)#为各扇区绑定鼠标进入事件
        c.tag_bind(pieN2,"<Enter>",PieN2)
        c.tag_bind(pieN3,"<Enter>",PieN3)
        c.tag_bind(pieN4,"<Enter>",PieN4)
        c.tag_bind(pieN5,"<Enter>",PieN5)
        w.mainloop()
    except ZeroDivisionError:
        print("总人数为0,产生异常!")
main()

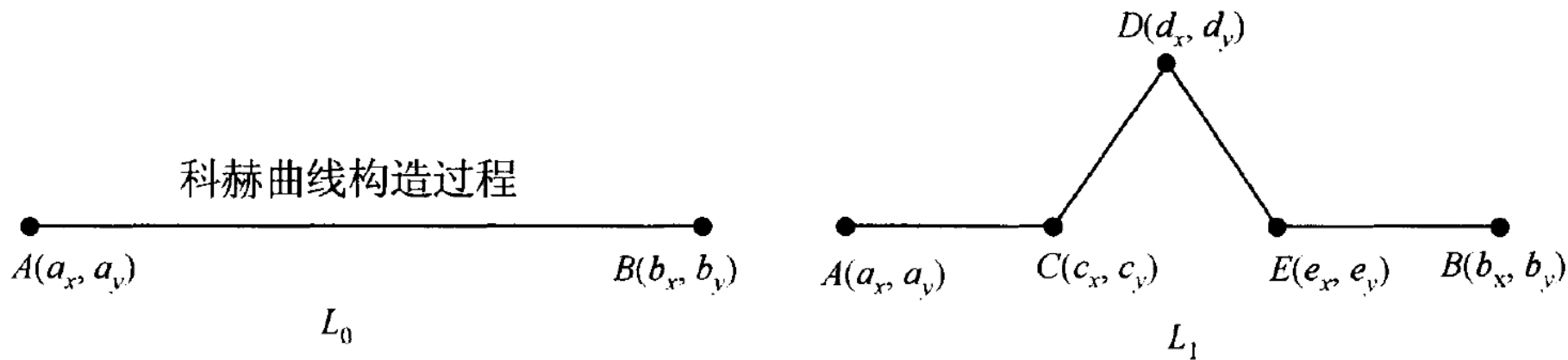
```

12.5.3 分形曲线

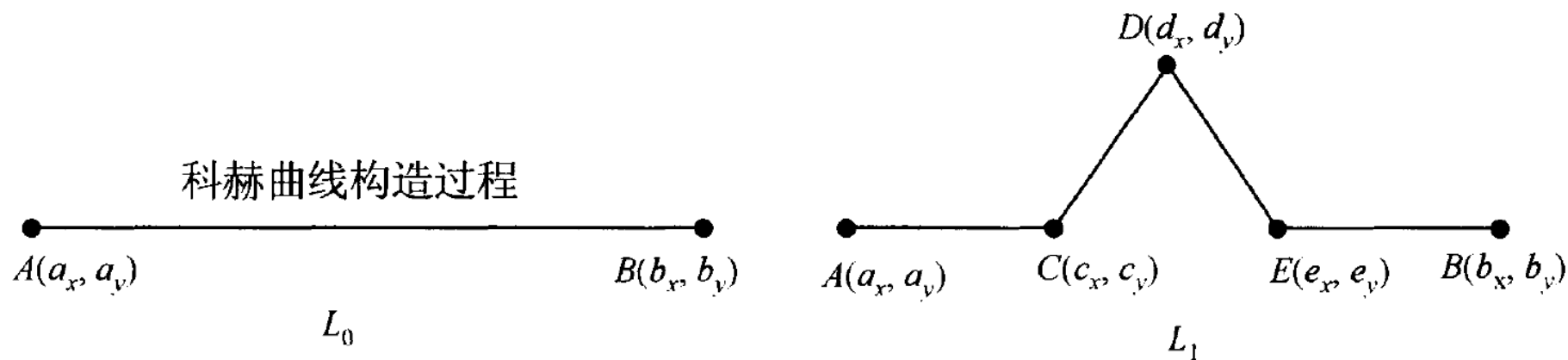
- 自然界存在许多复杂事物和现象，如蜿蜒曲折的海岸线、天空中奇形怪状的云朵、错综生长的灌木、太空中星罗棋布的星球等，还有许多社会现象，如人口的分布、物价的波动等，它们呈现异常复杂而毫无规则的形态，但它们具有自相似性。

- 人们把这些部分与整体以某种方式相似的形体称为分形（**fractal**），在此基础上，形成了研究分形性质及其应用的科学，称为分形理论。
- 科赫曲线是典型的分形曲线，由瑞典数学家科赫（**Koch**）于**1904**年提出。

- 例：绘制科赫曲线。
- 分析：科赫曲线的构造过程是，取一条直线段 L_0 ，将其三等分，保留两端的线段，将中间的一段用以该线段为边的等边三角形的另外两边代替，得到曲线 L_1 。

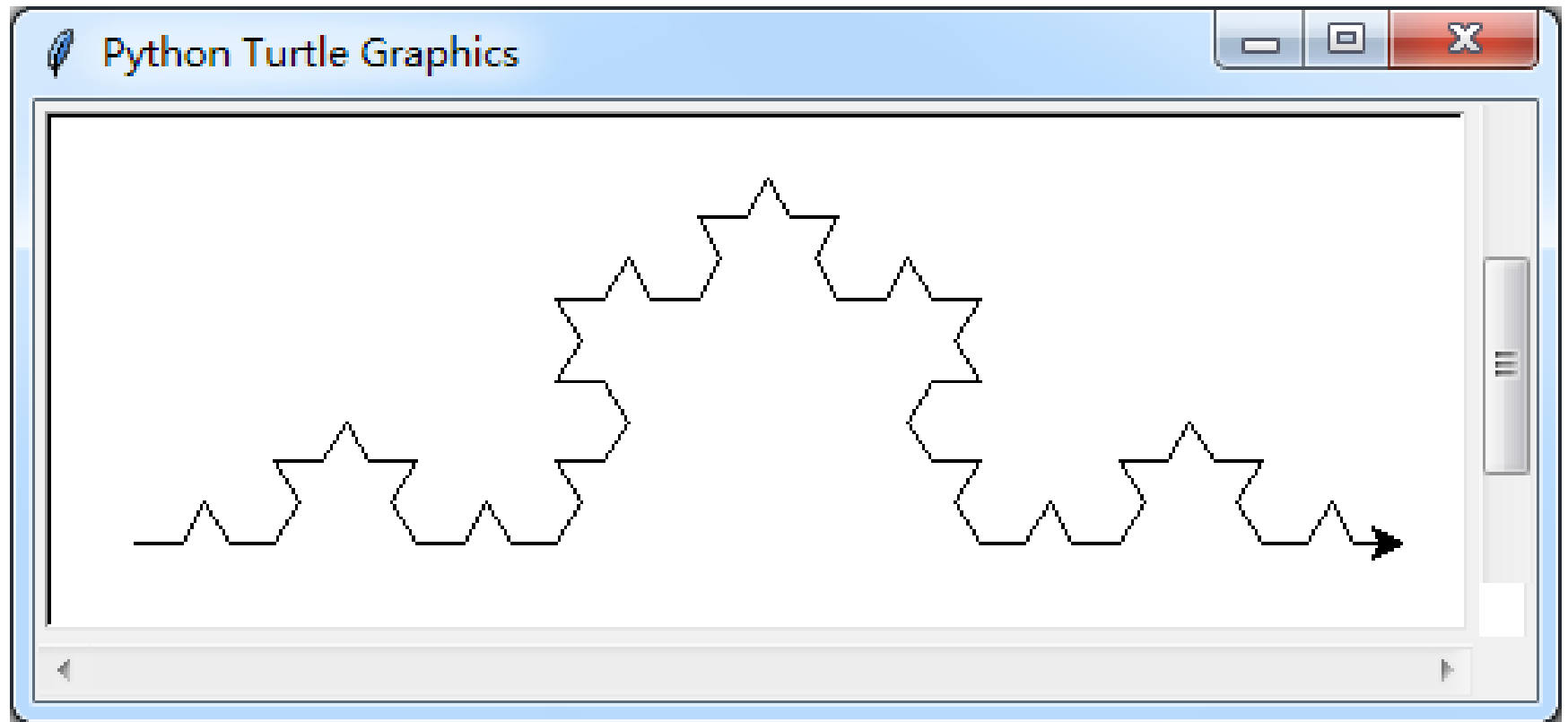


- 再对 L_1 中的4条线段都按上述方式修改，得到曲线 L_2 ，如此继续下去进行 n 次修改得到曲线 L_n ，当 $n \rightarrow \infty$ 时得到一条连续曲线 L ，这条曲线 L 就称为科赫曲线。



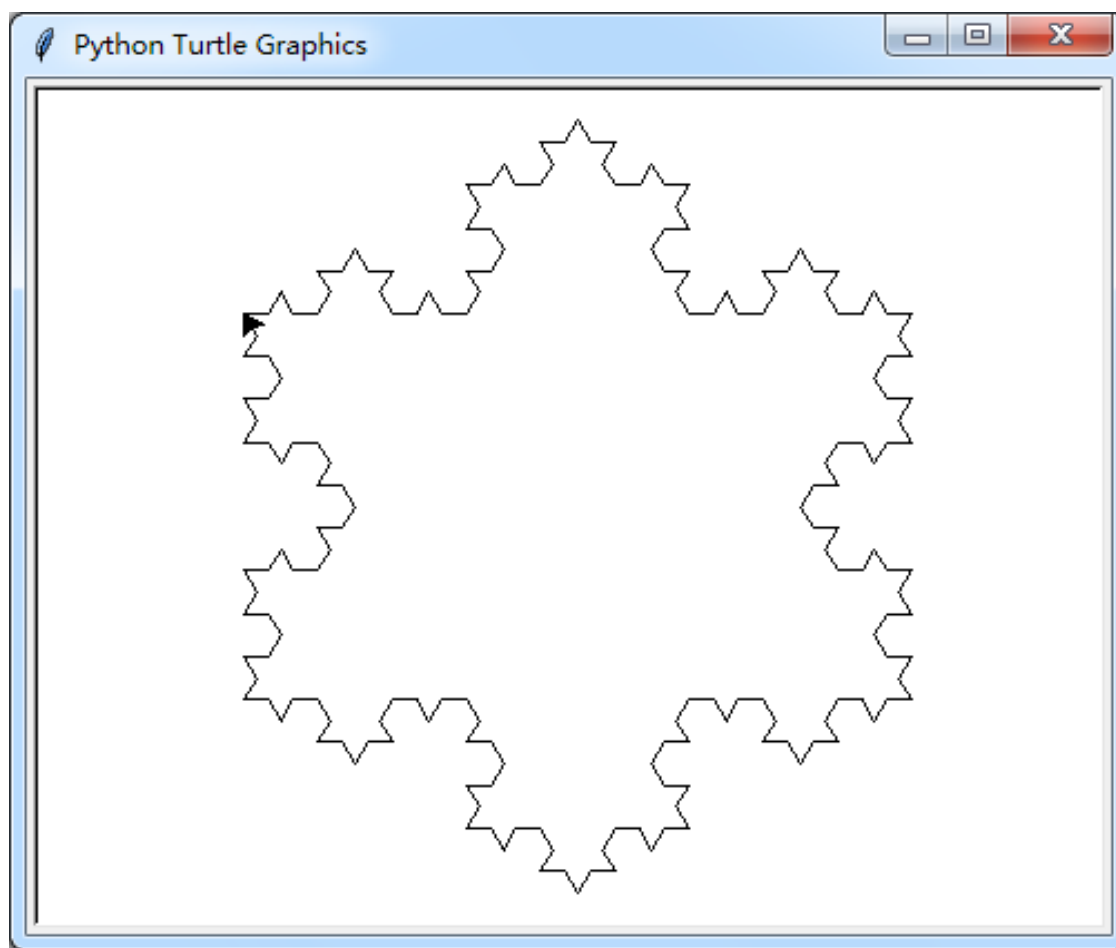
- 科赫曲线的构造规则是将每条直线用一条折线替代，通常称之为该分形的生成元，分形的基本特征完全由生成元决定。给定不同的生成元，就可以生成各种不同的分形曲线。
- 分形曲线的构造过程是通过反复用一个生成元来取代每一直线段，因而图形的每一部分都和它本身的形状相同，这就是自相似性，这是分形最为重要的特点。

- 分形曲线的构造过程也决定了制作该曲线可以用递归方法，即函数自己调用自己的过程。
- 参数 n 表示科赫曲线的细分度，参数 k 表示科赫曲线每段的长度。
- 当 $n=0$ 时，科赫曲线是一条直线。
- 当 $n=1$ 时，科赫曲线会在中间 $1/3$ 的位置画出一个边长为 $k/3$ 的等边三角形。
- 改变 n 的值可以获得不同细腻程度的科赫雪花曲线。



- **from turtle import ***
- **def koch(n,k):**
- **if n==0:**
- **forward(k)**
- **else:**
- **for angle in (60,-120,60,0):**
- **koch(n-1,k/3)**
- **left(angle)**
- **up()**
- **goto(-200,-50)**
- **down()**
- **koch(3,400)**

- 将科赫曲线以 60° 旋转组合就形成科赫雪花曲线效果。

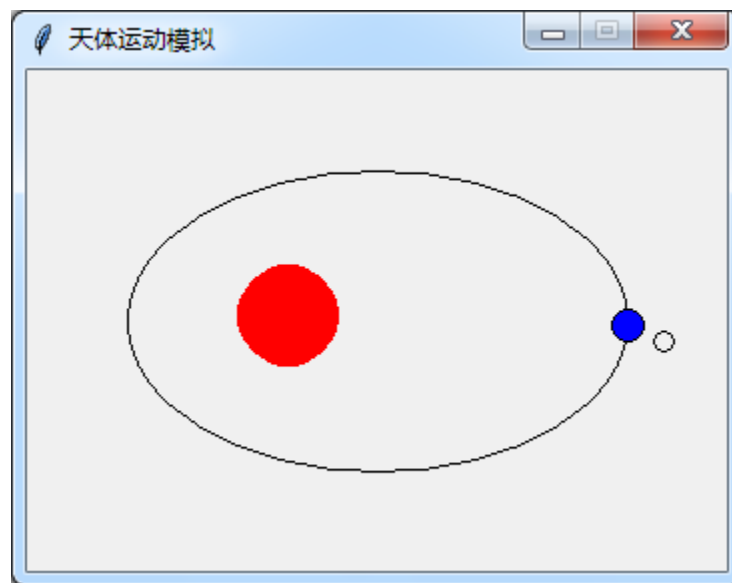


- **from turtle import ***
- **def koch(n,k):**
- **if n==0:**
- **forward(k)**
- **else:**
- **for angle in (60,-120,60,0):**
- **koch(n-1,k/3)**
- **left(angle)**
- **up()**
- **goto(-150,90)**
- **down()**
- **koch(3,300)**
- **right(120)**
- **koch(3,300)**
- **right(120)**
- **koch(3,300)**

12.5.4 利用动画模拟天体运动

- 动画是通过在屏幕上快速地交替显示一组静止图形（图像），或者让一幅图形（图像）快速地移动而实现的。
- 每一幅静止图形（图像）称为一帧，帧与帧之间在画面上只有小部分的不同，于是人眼的视觉暂留现象会给人以连续运动的感觉。

- 实验表明，每秒显示**24**帧画面能使人眼感觉到最佳的连续运动效果，所以在连续两帧画面之间应该停顿**0.04s**（秒）左右。



- 例：利用动画演示太阳、地球和月球三个天体之间的运动情况，即月球绕地球运动，并且和地球一起绕太阳运动。地球绕太阳运动**10**圈后暂停。
- 分析：首先建立图形窗口，然后画出太阳、地球和月球三个天体，并考虑它们之间的相互运动情况。
- 太阳位置固定不动，先考虑地球的运行轨迹，其方程如下：

$$\begin{cases} x = acost \\ y = bsint \end{cases}$$

- 地球在轨道上自西向东旋转时的每一个位置(x, y)都可利用该方程求出，其中椭圆轨道的a、b值是固定不变的，位置只由旋转角度t决定。假设地球每次旋转 0.01π 弧度，则地球的下一位置就是：

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

由此可以求出地球移动的距离：

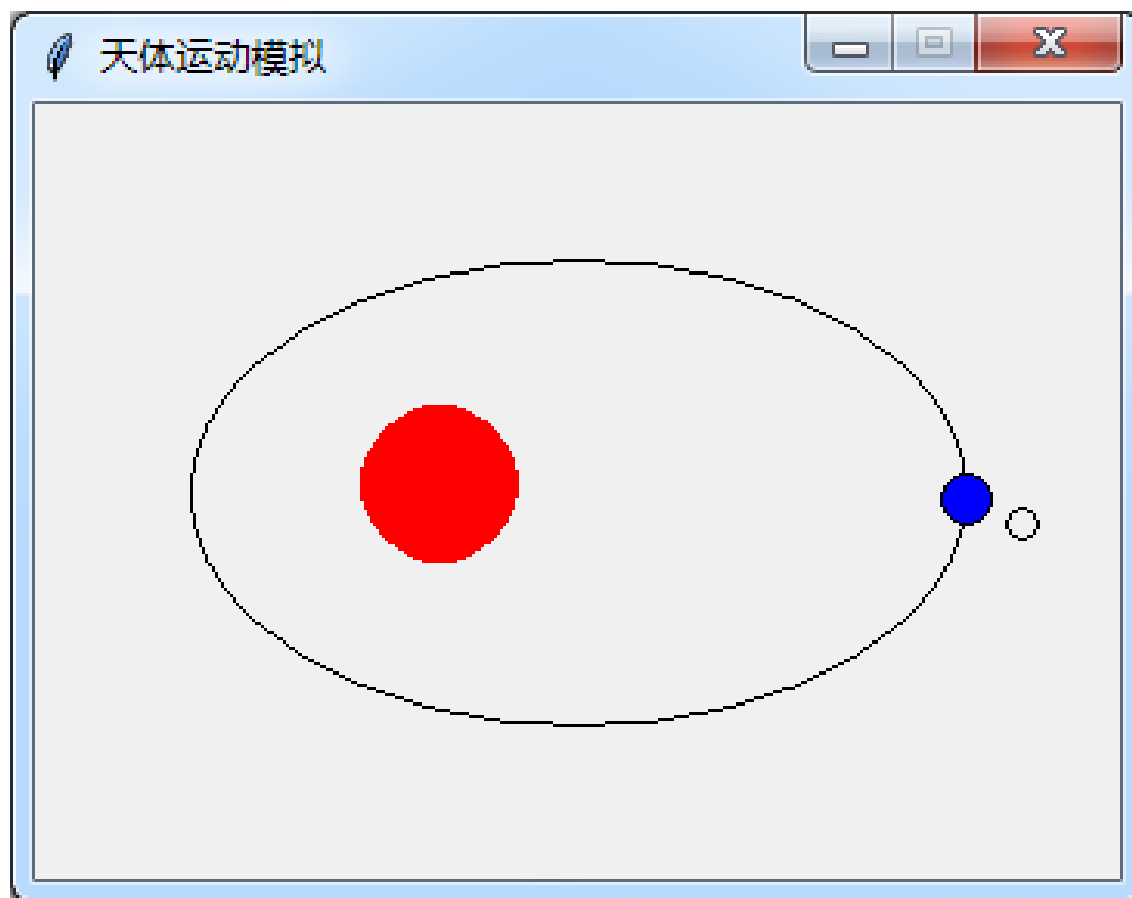
$$\begin{cases} \Delta x = x' - x \\ \Delta y = y' - y \end{cases}$$

- 可利用地球对象的`move()`方法将地球移动到新的位置，从而产生地球运动的效果。
- 假设椭圆中心在图形窗口中的坐标是(175, 125)，根据图形窗口的坐标系，地球在`t`角度时的位置需做以下变换：

$$\begin{cases} x = 175 + acost \\ y = 125 - bsint \end{cases}$$

- 月球的运动轨迹：
- 月球是与地球一起沿椭圆轨道绕太阳运动的，因此月球相对于太阳的位置变化与地球一样，由上述 x 和 y 决定（分别用 e_dx 和 e_dy 表示）。
- 月球又在绕地球旋转，利用同样方法可算出月球沿绕地球椭圆轨道（设 a 、 b 的值分别为25和20）运动时相对于地球的位置变化（分别用 m_dx 和 m_dy 表示），最终月球的位置变化为 e_dx+m_dx 和 e_dy+m_dy 。

- 月球绕地球的旋转速度大约是地球绕太阳的旋转速度的**12**倍（一年有**12**个月）。



- `from graphics import *`
- `from math import *`
- `from time import sleep`
- `w=GraphWin("天体运动模拟",350,250)`
- `orbit=Oval(Point(50,50),Point(300,200));orbit.draw(w)`
- `sun=Circle(Point(130,122),25)`
- `sun.setFill('red');sun.setOutline('red');sun.draw(w)`
- `earth=Circle(Point(300,125),8)`
- `earth.setFill('blue');earth.draw(w)`
- `moon=Circle(Point(315,110),5)`
- `moon.draw(w)`
- `e_X0=300;e_Y0=125 #地球当前位置`
- `m2e_X0=20;m2e_Y0=-15 #月球相对于地球的位置`
- `t=0`
- `while abs(t-2*pi*10)>1e-5:`
- `e_X1=175+125*cos(t) #计算地球的新位置`
- `e_Y1=125-75*sin(t)`
- `m2e_X1=25*cos(12*t) #计算月球的新位置`
- `m2e_Y1=-20*sin(12*t)`
- `e_dx=e_X1-e_X0`
- `e_dy=e_Y1-e_Y0`
- `m_dx=m2e_X1-m2e_X0`
- `m_dy=m2e_Y1-m2e_Y0`
- `earth.move(e_dx,e_dy)`
- `moon.move(e_dx+m_dx,e_dy+m_dy)`
- `e_X0=e_X1 #更新地球的当前位置`
- `e_Y0=e_Y1`
- `m2e_X0=m2e_X1 #更新月球的当前位置`
- `m2e_Y0=m2e_Y1`
- `sleep(0.04)`
- `t+=0.01*pi`

- 程序通过循环控制**3**个天体的运动，每次循环修改图形位置后显示新的画面。
- 两个画面之间的停顿可以用**time**模块中的**sleep()**函数来实现（以秒为单位）。
- 思考：
- 如何调节地球和月球运动的速度？

自测题

一、选择题

- 1. 画布坐标系的坐标原点是主窗口的（ ）。
A
● A. 左上角 B. 左下角 C. 右上角
D. 右下角
- 2. 从画布c删除图形对象r，使用的命令是（ ）。 D
● A. c.pack(r) B. r.pack(c)
● C. r.delete(c) D. c.delete(r)

● 3. 从画布c中将矩形对象r在x方向移动20像素，在y方向移动10像素，执行的语句是（ ）。C

● A. r.move(c,20,10) B. r.remove(c,10,20)

● C. c.move(r,20,10) D. c.move(r,10,20)

● 4. 以下不能表示红色的是（ ）。A

● A. red5 B. #f00

● C. #ff0000 D. #fff000000

● 5. 以下不能绘制正方形的图形对象是（ ）。 **B**

● **A. 矩形** **B. 图像**

● **C. 多边形** **D. 线条**

● 6. 语句`c.create_arc(20,20,100,100,style=PIESLICE)`

执行后，得到的图形是（ ）。 **C**

● **A. 曲线** **B. 弧**

● **C. 扇形** **D. 弓形**

● 7. 下列程序运行后，得到的图形是（ ）。 D

● `from tkinter import *`

● `w=Tk()`

● `c=Canvas(w,bg='white')`

● `c.create_oval(50,50,150,150,fill='red')`

● `c.create_oval(50,150,150,250,fill='red')`

● `c.pack()`

● `w.mainloop()`

● A. 两个相交的大小一样的圆

● B. 两个同心圆

● C. 两个相切的大小不一样的圆

● D. 两个相切的大小一样的圆

● 8. 下列程序运行后，得到的图形是（ ）。 **A**

● `from turtle import *`

● `reset()`

● `up()`

● `goto(100,100)`

● **A.** 只移动坐标不作图

B. 水平直线

● **C.** 垂直直线

D. 斜线

- 9. graphics模块中可以绘制从点（10， 20）到点（30， 40）直线的语句是（ ）。 D
 - A. Line(10,20,30,40)
 - B. Line((10,20),(30,40))
 - C. Line(10,30,20,40)
 - D. Line(Point(10,20),Point(30,40)).draw(w)
- 10. graphics模块中color.rgb(250,0,0)表示的颜色是（ ）。 C
 - A. 黑色 B. 绿色 C. 红色 D. 蓝色

- 二、填空题

- 1. **tkinter**图形处理程序包含一个顶层窗口，也称或____。主窗口或根窗口
- 2. 如果使用“**import tkinter**”语句导入**tkinter**模块，则创建主窗口对象**r**的语句是____。
r=tkinter.Tk()
- 3. **Python**中用于绘制各种图形、标注文本以及放置各种图形用户界面控件的区域称作____。画布

- 4. 将画布对象a在主窗口显现出来，使用的语句是____。 `a.pack()`
- 5. 画布对象用____方法绘制椭圆或____，其位置和尺寸通过____坐标和____坐标来定义。 `create_oval()`，圆，左上角，右下角
- 6. `turtle`绘图有三个要素，分别是____、和____。位置，方向，画笔

- 7. 与graphics方法Rectangle功能等价的tkinter画布方法是____，与graphics方法Point功能等价的tkinter画布方法是____，与graphics方法Circle功能等价的tkinter画布方法是____。
- create_rectangle, create_rectangle, create_oval
- 8. graphics模块把Point、Line等看作类，利用类可以创建相应的____，各种图形对象都具有自己的____和____。图形对象，属性，方法

● 三、问答题

- 1. 在Python中如何导入tkinter模块？
- 2. 画布对象的坐标如何确定的？和数学中的坐标系有何不同？
- 3. 在Python中如何表示颜色？
- 4. 画布对象中有哪些图形对象？如何创建？
- 5. graphics模块有哪些图形对象？如何创建？
- 6. 利用tkinter模块、turtle模块和graphics模块绘图各有哪些步骤？

