

Python语言程序设计

## 第2章 Python基本图形绘制

---





# 前课复习

# Python基本语法元素

- 缩进、注释、命名、变量、保留字
- 数据类型、字符串、整数、浮点数、列表
- 赋值语句、分支语句、函数
- input()、print()、eval()、 print()格式化



and	elif	import	raise	global
as	else	in	return	nonlocal
assert	except	is	try	True
break	finally	lambda	while	False
class	for	not	with	None
continue	from	or	yield	async
def	if	pass	del	await

保留字



```
#TempConvert.py
```

```
TempStr = input("请输入带有符号的温度值: ")
```

```
if TempStr[-1] in ['F', 'f']:
```

```
    C = (eval(TempStr[0:-1]) - 32)/1.8
```

```
    print("转换后的温度是{:.2f}C".format(C))
```

```
elif TempStr[-1] in ['C', 'c']:
```

```
    F = 1.8*eval(TempStr[0:-1]) + 32
```

```
    print("转换后的温度是{:.2f}F".format(F))
```

```
else:
```

```
    print("输入格式错误")
```

温度转换





# 本课概要

# 第2章 Python基本图形绘制



- 2.1 深入理解Python语言
- 2.2 实例2: Python蟒蛇绘制
- 2.3 模块1: turtle库的使用
- 2.4 turtle程序语法元素分析



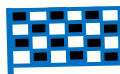
# 第2章 Python基本图形绘制

## 方法论

- Python语言及海龟绘图体系

## 实践能力

- 初步学会使用Python绘制简单图形





Python语言程序设计

## 2.2 实例2: Python蟒蛇绘制

---





# "Python蟒蛇绘制"问题分析

# Python蟒蛇绘制

## 设计蟒蛇的基本形状



# Python蟒蛇绘制

## 用程序绘制一条蟒蛇

- **问题1: 计算机绘图是什么原理？**

一段程序为何能够产生窗体？为何能在窗体上绘制图形？

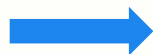
- **问题2: Python蟒蛇绘制从哪里开始呢？**

如何绘制一条线？如何绘制一个弧形？如何绘制一个蟒蛇？

# Python蟒蛇绘制

用程序绘制一条蟒蛇

实例1: 温度转换



Python蟒蛇绘制

能否借鉴？



# "Python蟒蛇绘制"实例编写

```
#PythonDraw.py
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

使用IDLE的文件方式

编写代码

并保存为

PythonDraw.py 文件

# 运行效果

**IDLE打开文件，按F5运行**





```
#PythonDraw.py
```

```
import turtle
```

```
turtle.setup(650, 350, 200, 200)
```

```
turtle.penup()
```

```
turtle.fd(-250)
```

```
turtle.pendown()
```

```
turtle.pensize(25)
```

```
turtle.pencolor("purple")
```

```
turtle.seth(-40)
```

```
for i in range(4):
```

```
    turtle.circle(40, 80)
```

```
    turtle.circle(-40, 80)
```

```
turtle.circle(40, 80/2)
```

```
turtle.fd(40)
```

```
turtle.circle(16, 180)
```

```
turtle.fd(40 * 2/3)
```

```
turtle.done()
```

## 程序关键

*import* 保留字

引入了一个绘图库

名字叫：*turtle*

没错，就是 *海龟*

- ✓ 与之前的代码有两个显著的不同：
- ✓ 没有使用显式的用户输入输出，即没有input()和output()函数
- ✓ 绝大多数代码都是<a>.<b>()形式，代码中没有赋值语句

# Python语言程序设计

## 2.3 模块1: turtle库的使用

---

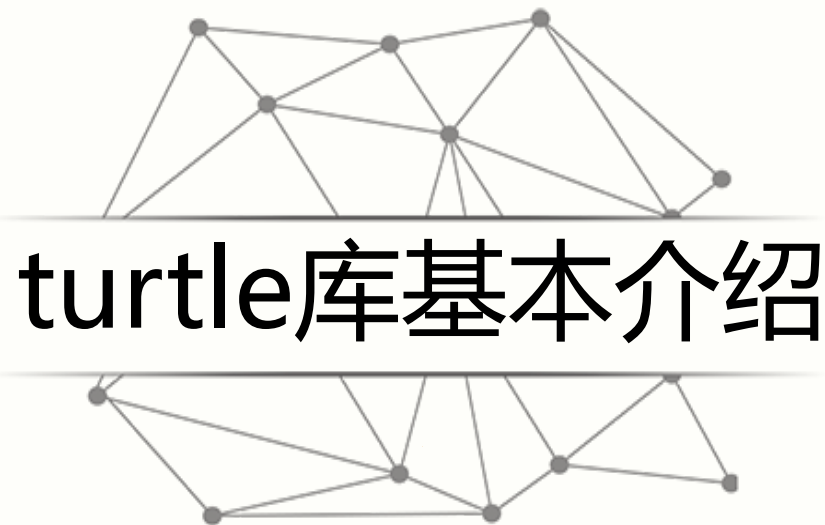


# 模块1: turtle库的使用



- turtle库基本介绍
- turtle绘图窗体布局
- turtle空间坐标体系
- turtle角度坐标体系
- RGB色彩体系





# turtle库基本介绍

# turtle库概述

**turtle(海龟)库是turtle绘图体系的Python实现**

- **turtle绘图体系：1969年诞生，主要用于程序设计入门**
- **Python语言的**标准库**之一**
- **入门级的图形绘制函数库**

# 标准库

**Python计算生态 = 标准库 + 第三方库**

- **标准库**：随解释器直接安装到操作系统中的功能模块
- **第三方库**：需要经过安装才能使用的功能模块
- **库Library、包Package、模块Module**，统称**模块**

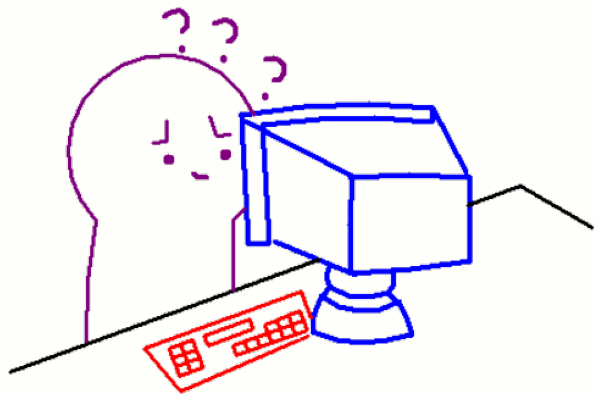
# turtle的原理

**turtle(海龟)是一种真实的存在**

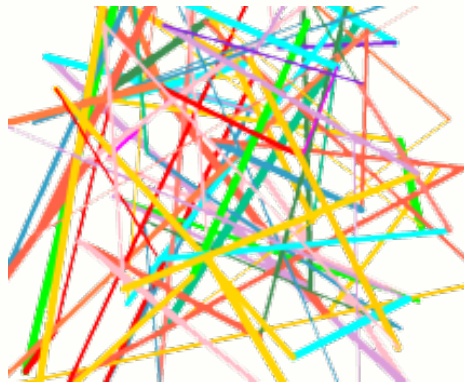
- **有一只海龟，其实在窗体正中心，在画布上游走**
- **走过的轨迹形成了绘制的图形**
- **海龟由程序控制，可以变换颜色、改变宽度等**

# turtle的魅力

Python还能画画



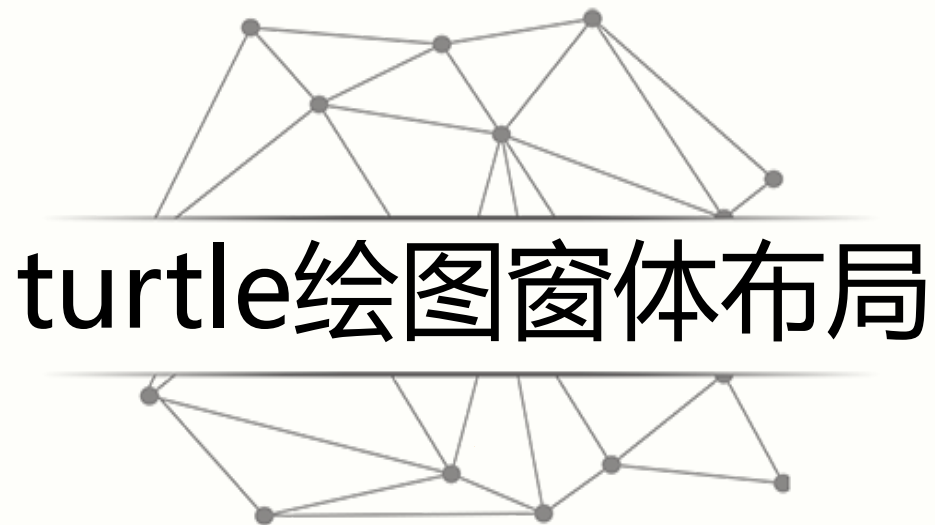
呐~花花送给你



Beijing





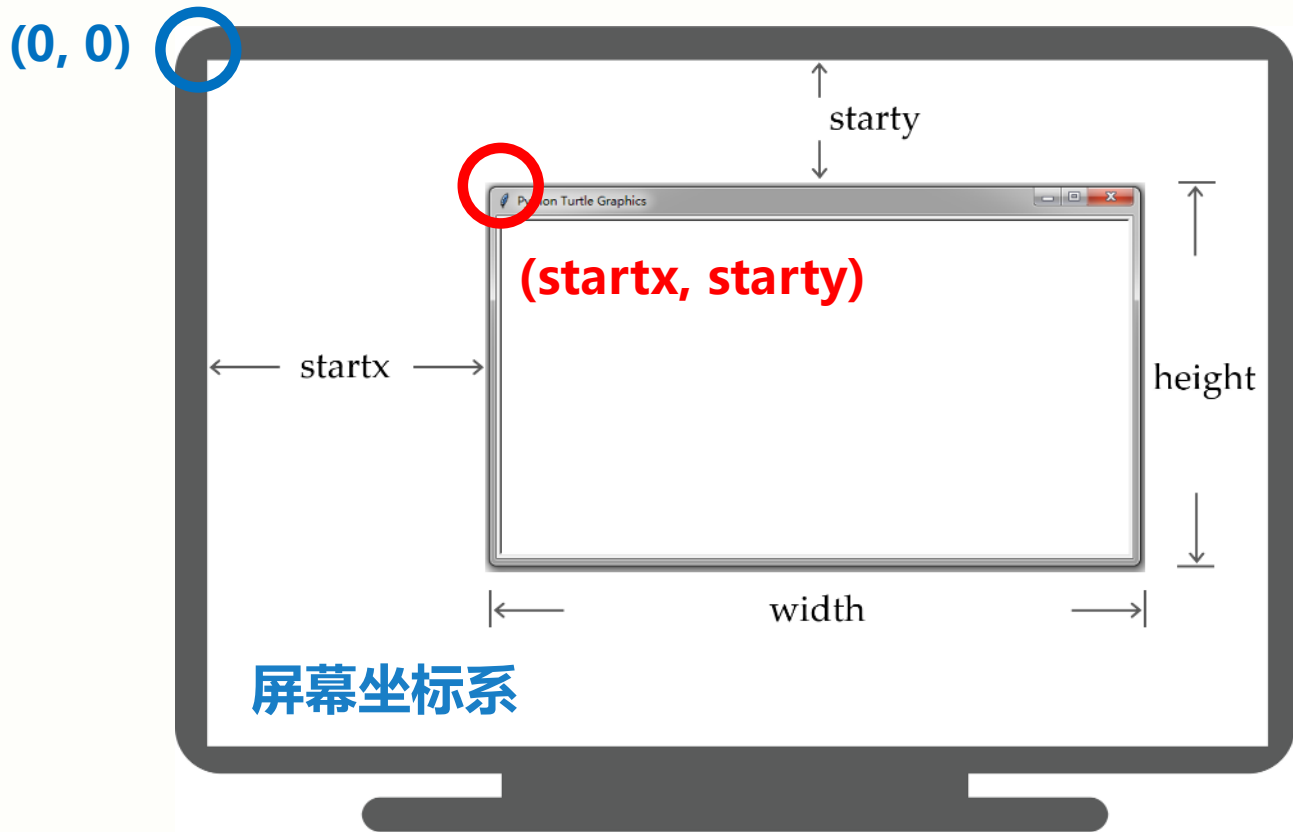


turtle绘图窗体布局

# turtle的绘图窗体

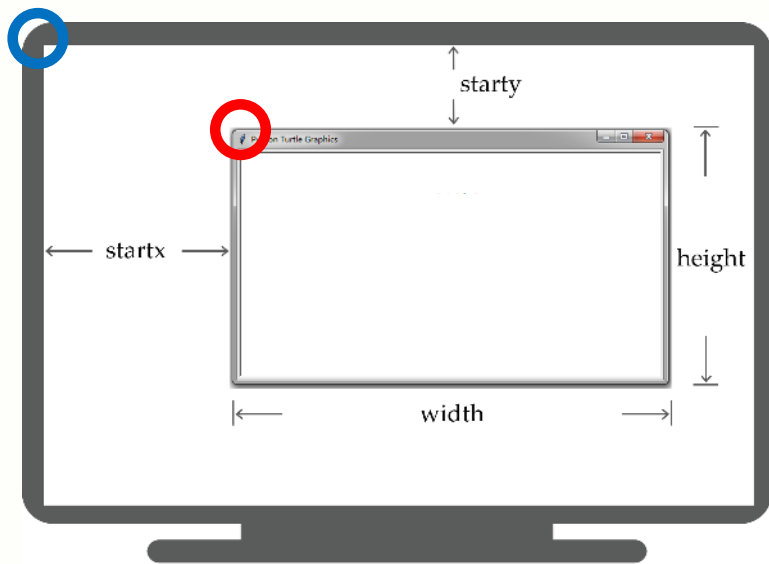


# turtle的绘图窗体



# turtle的绘图窗体

`turtle.setup(width, height, startx, starty)`



- `setup()`设置窗体大小及位置
- 4个参数中后两个可选
- `setup()`不是必须的，默认为屏幕的正中心

# turtle的绘图窗体

`turtle.setup(800,400,0,0)`



`turtle.setup(800,400)`



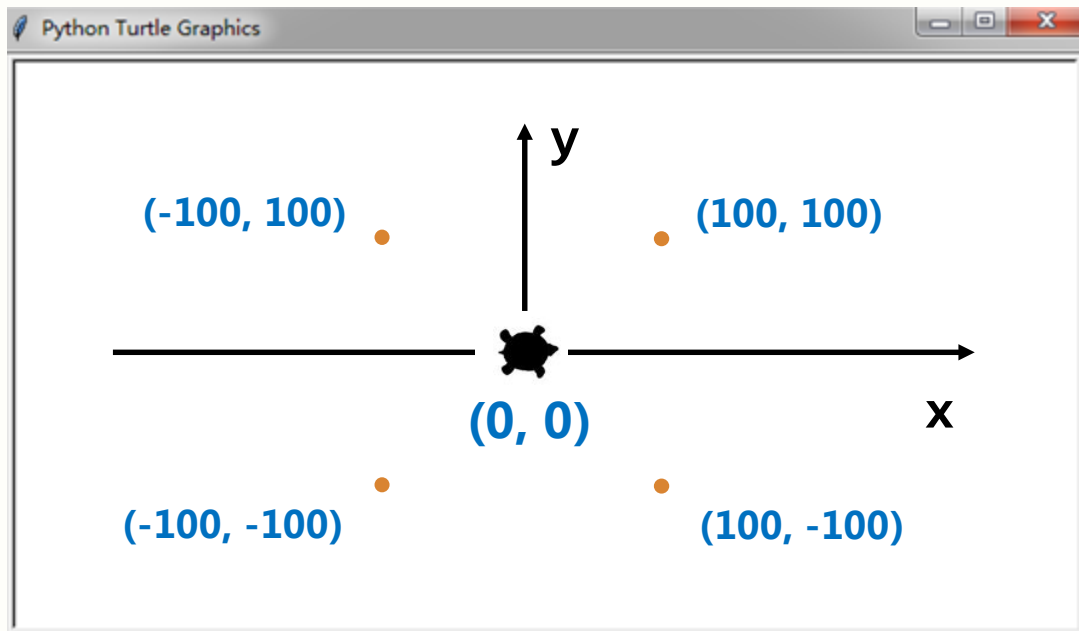
width和height，如果值是整数，表示像素值，如果是小数，表示窗口宽度/高度与屏幕的比例  
startx和starty是窗口左侧/顶部与屏幕左侧/顶部的像素距离，如果为空，则位于中央



turtle空间坐标体系

# turtle空间坐标体系

绝对坐标



# turtle的原理

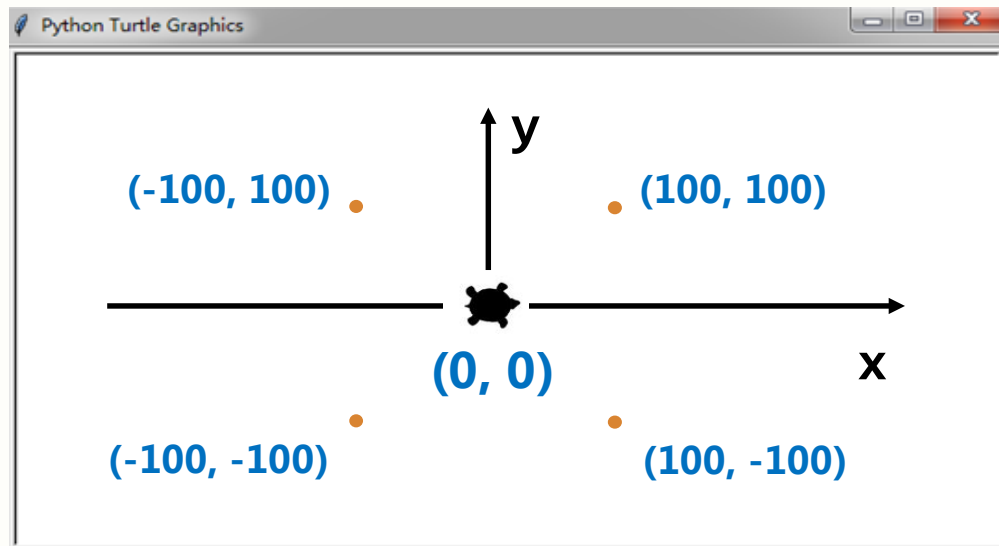
**turtle(海龟)是一种真实的存在**

- **有一只海龟，其实在窗体正中心，在画布上游走**
- **走过的轨迹形成了绘制的图形**
- **海龟由程序控制，可以变换颜色、改变宽度等**



# turtle空间坐标体系

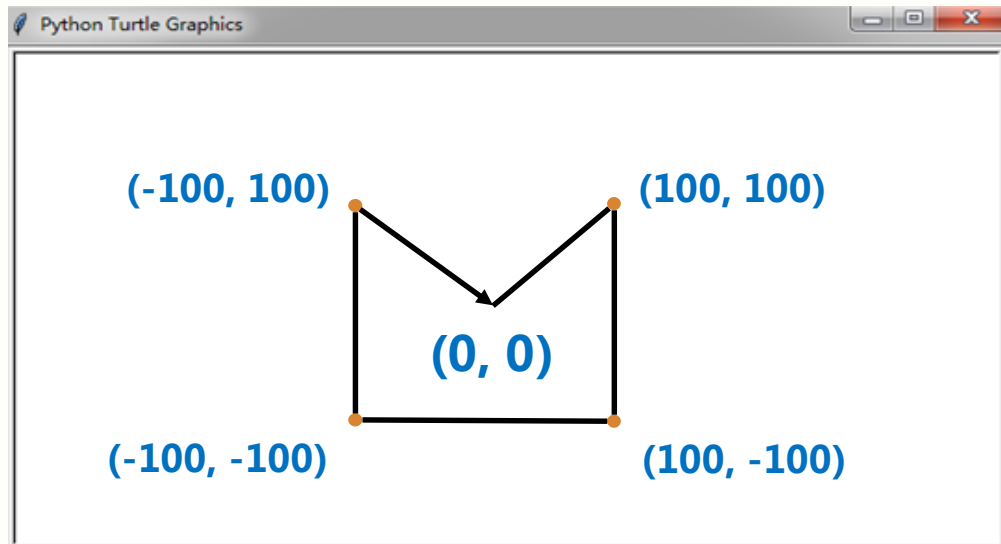
`turtle.goto(x, y)`



`turtle.goto()`函数是指从当前的点指向括号内所给的绝对坐标

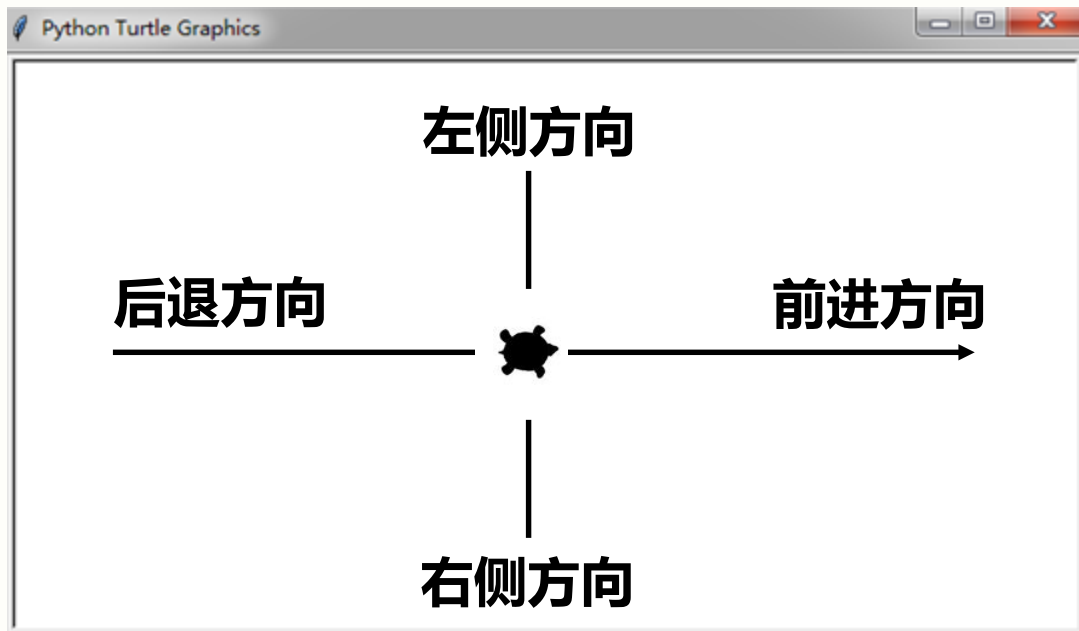
# turtle空间坐标体系

```
import turtle  
  
turtle.goto( 100, 100)  
turtle.goto( 100,-100)  
turtle.goto(-100,-100)  
turtle.goto(-100, 100)  
turtle.goto(0,0)
```



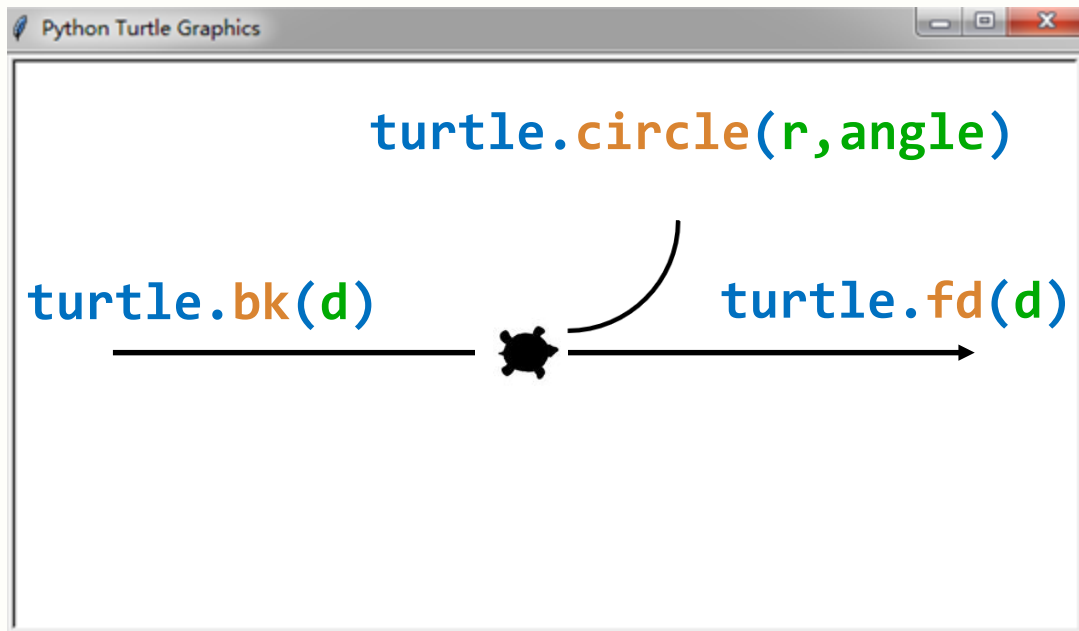
# turtle空间坐标体系

海龟坐标



海龟坐标，是把当前的点当做坐标，有“前进方向”，“后退方向”，“左侧方向”，“右侧方向”

# turtle空间坐标体系



`turtle.fd(d)` : 向海龟当前前进方向移动d距离

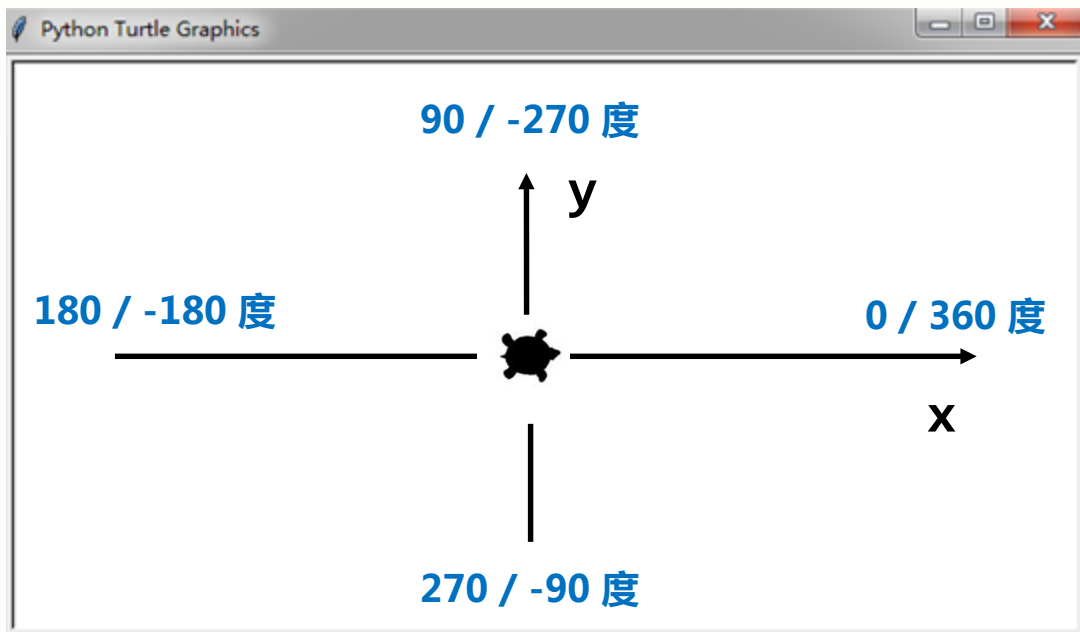
`turtle.bk(d)` : 向海龟当前后退方向移动d距离

`turtle.circle(r,angle)` : 在海龟当前位置根据半径r绘制angle角度的弧形



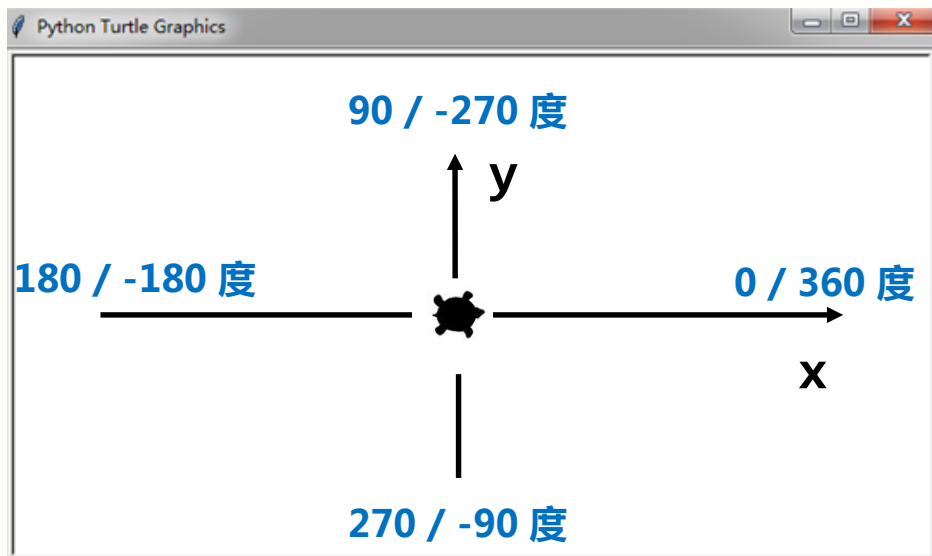
# turtle角度坐标体系

绝对角度



# turtle角度坐标体系

`turtle.seth(angle)`



- **seth()**改变海龟行进方向
- **angle**为绝对度数
- **seth()**只改变方向但不行进

# turtle角度坐标体系

`turtle.seth(45)`



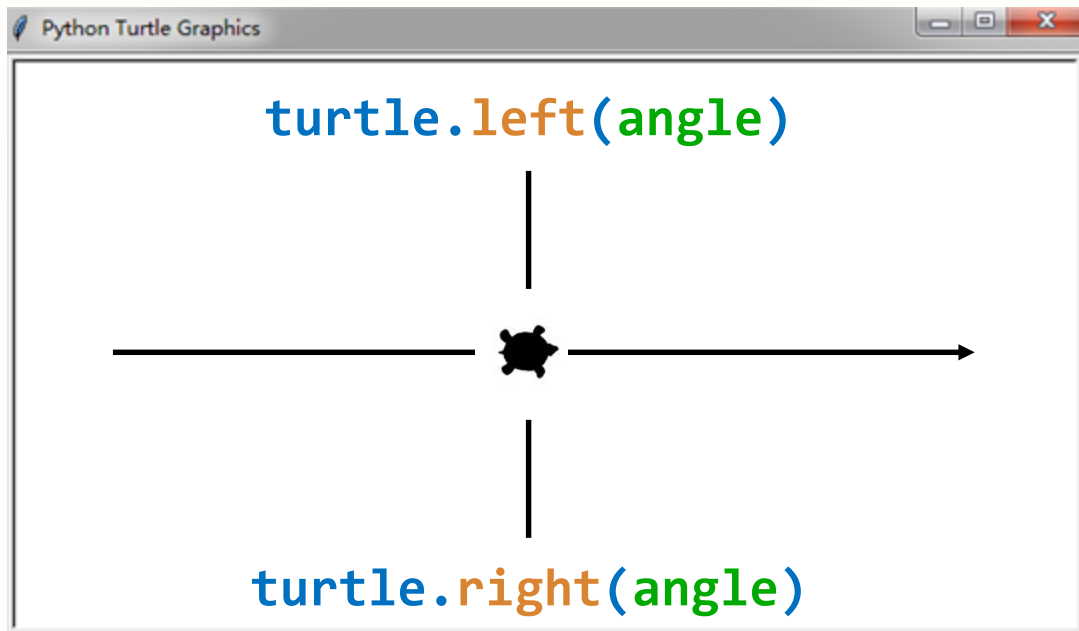
`turtle.seth(-135)`





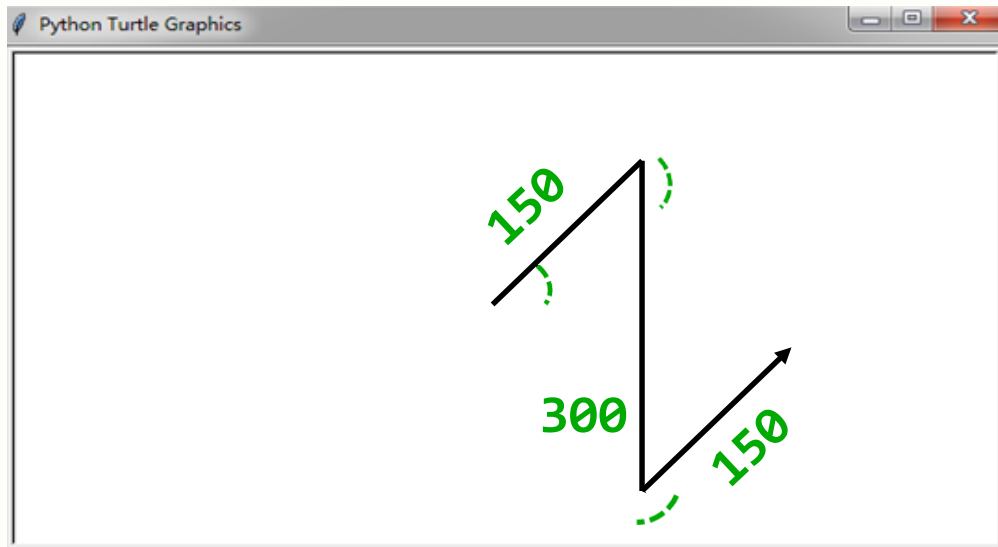
# Turtle角度坐标体系

海龟角度



# Turtle角度坐标体系

```
import turtle  
turtle.left(45)  
turtle.fd(150)  
turtle.right(135)  
turtle.fd(300)  
turtle.left(135)  
turtle.fd(150)
```





# RGB色彩体系

# RGB色彩模式

由三种颜色构成的万物色



- RGB指红绿蓝三个通道的颜色组合
- 覆盖视力所能感知的所有颜色
- RGB每色取值范围0-255整数或0-1小数

# 常用RGB色彩

英文名称	RGB整数值	RGB小数值	中文名称
white	255, 255, 255	1, 1, 1	白色
yellow	255, 255, 0	1, 1, 0	黄色
magenta	255, 0, 255	1, 0, 1	洋红
cyan	0, 255, 255	0, 1, 1	青色
blue	0, 0, 255	0, 0, 1	蓝色
black	0, 0, 0	0, 0, 0	黑色

# 常用RGB色彩

英文名称	RGB整数值	RGB小数值	中文名称
seashell	255, 245, 238	1, 0.96, 0.93	海贝色
gold	255, 215, 0	1, 0.84, 0	金色
pink	255, 192, 203	1, 0.75, 0.80	粉红色
brown	165, 42, 42	0.65, 0.16, 0.16	棕色
purple	160, 32, 240	0.63, 0.13, 0.94	紫色
tomato	255, 99, 71	1, 0.39, 0.28	番茄色

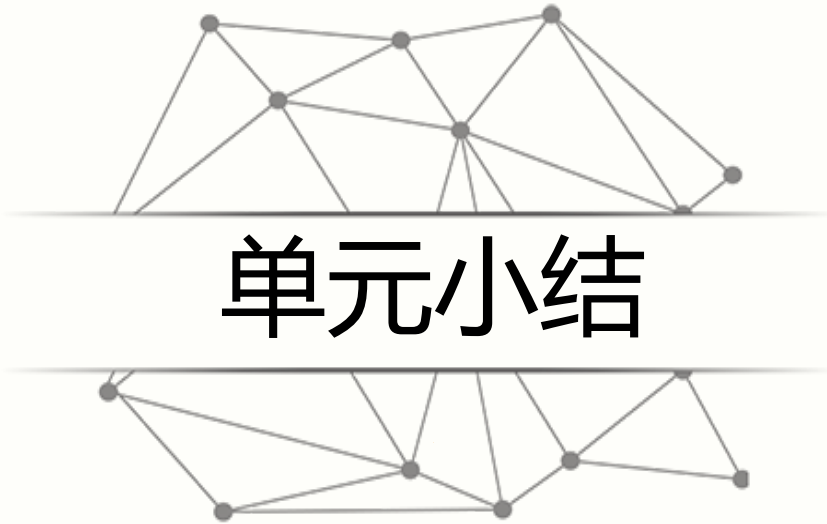
# turtle的RGB色彩模式

默认采用小数值 可切换为整数值



`turtle.colormode(mode)`

- **1.0** : RGB小数值模式
- **255** : RGB整数值模式



# 单元小结



# 模块1: turtle库的使用

- **turtle库的海龟绘图法**
- **turtle.setup()调整绘图窗体在电脑屏幕中的布局**
- **画布上以中心为原点的空间坐标系: 绝对坐标&海龟坐标**
- **画布上以空间x轴为0度的角度坐标系: 绝对角度&海龟角度**
- **RGB色彩体系, 整数值&小数值, 色彩模式切换**



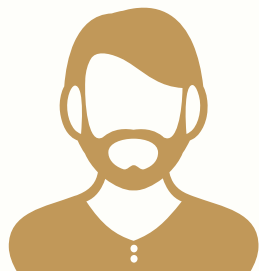
Python语言程序设计

## 2.4 turtle程序语法元素分析

---



# turtle程序语法元素分析



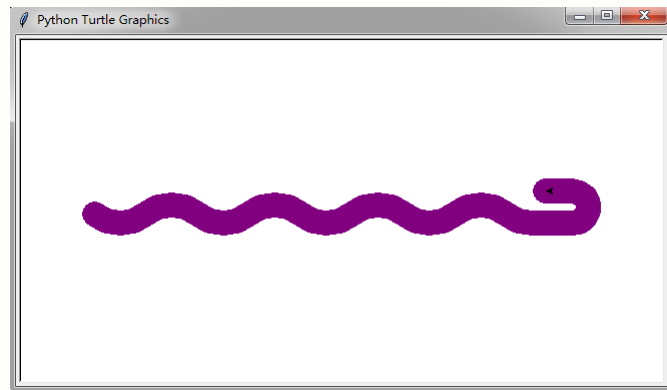
- 库引用与import
- turtle画笔控制函数
- turtle运动控制函数
- turtle方向控制函数
- 基本循环语句
- "Python蟒蛇绘制"代码分析





# 库引用与import

```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```



## <a>.<b>()的编码风格

调用了turtle库之后，所有的被调用的函数都使用了<a>.<b>()形式  
这种通过使用函数库并利用库中函数进行编程的方法是python语言最重要的特点，称为“模块编程”

# 库引用

## 扩充Python程序功能的方式

- 使用**import**保留字完成，采用<a>.<b>()编码风格

**import** <库名>

<库名>.<函数名>(<函数参数>)

```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

## 引入turtle库

## 使用turtle库函数

## 完成功能

可是可是, 好多turtle, 很繁琐嘛...

# import更多用法

使用from和import保留字共同完成

**from** <库名> **import** <函数名, 函数名, ..., >

**from** <库名> **import** \* # \* 是通配符, 表示所有函数

<函数名>(<函数参数>)

调用该库的函数时不再需要使用库名, 直接使用函数名



```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```



```
from turtle import *
setup(650, 350, 200, 200)
penup()
fd(-250)
pendown()
pensize(25)
pencolor("purple")
seth(-40)
for i in range(4):
    circle(40, 80)
    circle(-40, 80)
circle(40, 80/2)
fd(40)
circle(16, 180)
fd(40 * 2/3)
done()
```

这么好的方法为何不早  
说...

# import更多用法

## 两种方法比较

**import** <库名>

<库名>.<函数名>(<函数参数>)

**from** <库名> **import** <函数名>

**from** <库名> **import** \*

<函数名>(<函数参数>)

**第一种方法不会出现函数重名问题，第二种方法则会出现**

第一种能够显式注明函数来源，在引用较多库时代码可读性更好

第二种利用保留字直接引用库中函数，可以使代码更简洁

第二种可能会出现，当用户定义一个函数<b>，库中的函数名<b>将会与用户自定义的函数名冲突，python要求函数名命名唯一

# import更多用法

使用import和as保留字共同完成

**import** <库名> **as** <库别名>

<库别名>.<函数名>(<函数参数>)

给调用的外部库关联一个**更短、更适合自己的名字**

```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```



```
import turtle as t
t.setup(650, 350, 200, 200)
t.penup()
t.fd(-250)
t.pendown()
t.pensize(25)
t.pencolor("purple")
t.seth(-40)
for i in range(4):
    t.circle(40, 80)
    t.circle(-40, 80)
t.circle(40, 80/2)
t.fd(40)
t.circle(16, 180)
t.fd(40 * 2/3)
t.done()
```

**这个方法好！**

# 1. 模块

编程，很多的时候要会使用别人的积木块来搭建自己的程序，站在巨人的肩上编程，将会做到事半功倍。Python语言的优点之一是免费开源，除自带丰富的模块外，有很多的编程爱好者，编写了大量功能强大的第三方库。用好Python自带的库和第三方库，是学习Python语言的基本目标。

## ● 模块

Python中的**模块**实际上就是包含**函数和类的Python程序**，它以“.py”为后缀。可以在需要的时候，通过import将它引用过来。

## ● 模块导入

Python中提供了以下三种方式导入模块：

- (1) import 模块名
- (2) import 模块名 as 模块别名
- (3) from 模块名 import 函数名/子模块名/属性

## 2. 模块的路径

Python自带的模块或者是第三方库，在安装时，系统自动将模块的存放路径记录在**sys.path**列表中，在导入时，Python解释器会根据**sys.path**记录的路径去寻找要导入的模块。那么自己编写的模块，如何能让解释器知道路径呢？有两种方法。第一种方法是在**sys.path**列表里添加自己所写模块的路径；第二条方法是设置系统的环境变量，使其包含模块的路径。

### (1) 列表里添加路径

【例5-6】程序中调用了【例5-5】的函数，即通过“import eg5\_5”把eg5\_5.py模块导入到eg5\_6.py中。在那里我们没有特别强调eg5\_5.py的路径，是因为eg5\_5.py和eg5\_6.py在同一个路径下，所以Python解释器能够找到。如果eg5\_5.py和eg5\_6.py不在同一路径下，则需要将eg5\_5.py的路径添加到sys.path列表中。现在eg5\_5.py存放在“e:\py\_study”路径下，则eg5\_6.py代码修改如下：

```
#eg5_6.py
#导入sys模块
import sys
#sys.path列表里追加eg5_5.py的路径 "d:\py_study"
sys.path.append( "e:\py_study")
#导入eg5_5.py模块
import eg5_5
def evenDec(m):
    if m%2 ==0 and m>0:
        #range()的终止值是int(m/2)+1，不能超过此值，否则会有重复的输出
        for i in range(3,int(m/2)+1):
            if i%2==1 and eg5_5.prime_judg(i) and eg5_5.prime_judg(m-i):
                print("%d=%d+%d"%(m,i,m-i))
evenDec(34)
```

```
>>> import sys
>>> sys.path
```

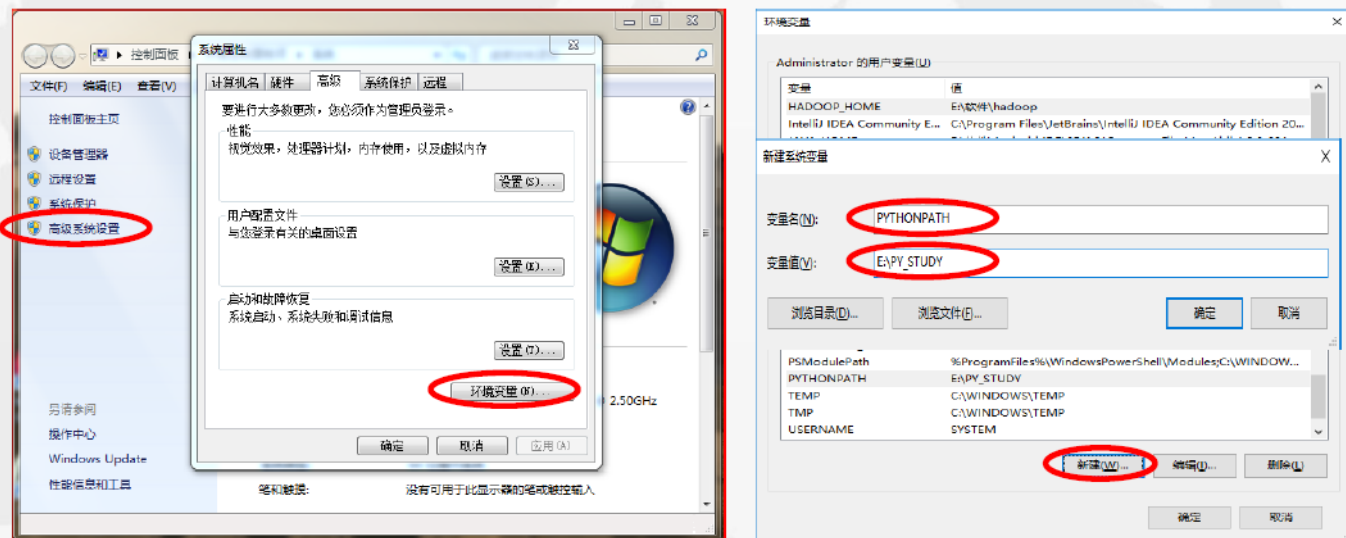
```
['',
 'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python37\\Lib\\idlelib',
 'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python37\\python37.zip',
 .....]
```

查看path的值

## 2. 模块的路径

### (2) 修改系统环境变量的值

- ① 在windows桌面上，右击“计算机”，选择“属性”命令；
- ② 在左图上点击“高级系统设置”，弹出“系统属性”对话框，点击“环境变量...”即可进入右图所示的环境变量设置对话框；



- ③ 在右图所示“新建系统变量”对话框中，变量名框输入“PYTHONPATH”，把“e:\py\_study”添加到“变量值”的列表中，点击“确定”结束环境变量的设置。

## 2. 模块的路径

设置好环境变量后，【例5-6】的代码可以做如下修改。不需要修改`sys.path`列表的值，直接导入`eg5_5.py`模块即可。

```
#eg5_6.py

import eg5_5

def evenDec(m):
    if m%2 == 0 and m > 0:
        #range()的终止值是int(m/2)+1，不能超过此值，否则会有重复的输出
        for l in range(3,int(m/2)+1):
            if i%2==1 and eg5_5.prime_judg(i) and eg5_5.prime_judg(m-i):
                print( "%d=%d+%d" %(m,l,m-i))

evenDec(34)
```



### 3. 命名空间

**命名空间表示标识符的可见范围。**标识符就是用来标识某个对象的，包括变量名、函数名、模块名、类名等。一个标识符可在多个命名空间中定义，但它在不同命名空间中的含义是互不相关的。如有两个“刘卫东”同学在同一个学校不同的班级里，老师在各自的班级里点名时，直接喊“刘卫东”就可以了，自然就是本班的“刘卫东”同学，如果在全校点名时，就需要区分是哪个班的“刘卫东”了，这时候点名，需要采用“班级+刘卫东”的形式了。那么班级就是命名空间。

在Python中，每个模块都会维护一个独立的命名空间，在模块外使用标识符时，需要加上模块名，如`math.pi`，当然，也需要结合模块的导入方式，如果使用“`from 模块名 import 函数名/属性/子模块名`”方式导入时，一定要注意不同的模块里不要存在相同的标识符。

## 4. \_\_name\_\_ 属性

Python中为了区分代码块是单独运行，还是作为模块导入到另一个代码中进行运行，通过对模块的\_\_name\_\_属性值的判断来进行识别。

**模块作为单独的程序运行时，\_\_name\_\_的属性值是“\_\_main\_\_”，而作为模块导入时，\_\_name\_\_属性的值就是该模块的名字了**，因此，从现在开始，每个py程序文件中，如果有函数、类的定义，则都进行这样的判断。即增加一个if \_\_name\_\_ == '\_\_main\_\_' 的判断。如【例5-5】的代码修改如下：

```
#eg5_5_1.py
#write by ding
#2018-5-8
import math
def prime_judg(s):
    #由于range()不包含终值，所以要加1
    for i in range(2,int(math.sqrt(s))+1):
        if s%i==0:
            break
    else:
        return True
    return False

if __name__ == '__main__':
    print(prime_judg(13))
```

## 包

大型复杂的项目，通常会有多个模块，为了更好地管理这些模块，避免命名空间的冲突，Python中采用包进行管理。**包其实就是一个文件夹或者叫目录，但其中必须包含一个名为“\_\_init\_\_.py”的文件。**“\_\_init\_\_.py”文件的内容可以是空，仅用于表示该目录是一个包，也可以写入一些初始化代码。另外包可以嵌套，即把子包放在某个包内。

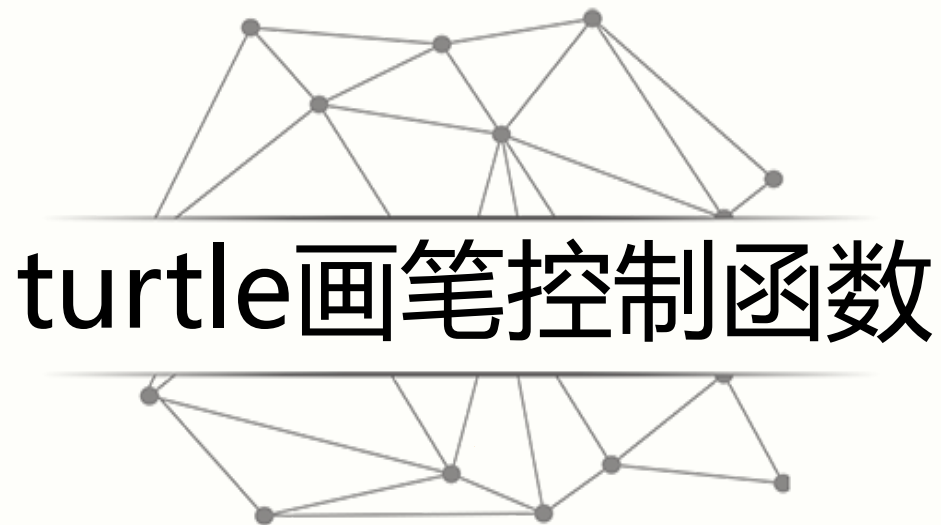
有了包之后，导入模块时，需要加上包的名称，即包名.模块名。如【例5-6】和【例5-5】改用包进行管理。

# 包

## 属性

- ①在“d:\”下创建“py”文件夹;
- ②把eg5\_5\_1.py复制到py文件夹;
- ③在“d:\py”目录下,新建一个空的\_\_init\_\_.py文件;
- ④修改【例5\_6】的代码如下,保存成eg5\_6\_2.py。

```
#eg5_6_2.py
import sys
if "d:\\\" not in sys.path:
    sys.path.append("d:\\") #添加包py的路径到sys.path中
import py.eg5_5_1
def even_Dec(m):
    if m%2==0 and m>0 :
        #range()的终止值是int(m/2)+1, 不能超过此值,
        #否则会有重复的输出
        for i in range(3,int(m/2)+1):
            #调用模块要加上包名
            if i%2==1 and py.eg5_5_1.prime_judg(i) and py.eg5_5_1.prime_judg(m-i):
                print("%d=%d+%d"%(m,i,m-i))
if __name__ == '__main__':
    even_Dec(34)
```



```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

**penup(), pendown()**

**pensize(), pencolor()**

# 画笔控制函数

画笔操作后一直有效，一般成对出现

- **turtle.penup()**      别名    **turtle.pu()**

抬起画笔，不绘制形状

- **turtle.pendown()**    别名    **turtle.pd()**

落下画笔，绘制形状

# 画笔控制函数

画笔设置后一直有效，直至下次重新设置

- `turtle.pensize(width)` 别名 `turtle.width(width)`

画笔宽度，无参数时返回当前宽度

- `turtle.pencolor(color)` `color`为颜色字符串或r,g,b值

画笔颜色，无参数时返回当前颜色



# 画笔控制函数

**pencolor(color)**的color可以有三种形式

- 颜色字符串 : `turtle.pencolor("purple")`
- RGB的小数值 : `turtle.pencolor(0.63, 0.13, 0.94)`
- RGB的元组值 : `turtle.pencolor((0.63,0.13,0.94))`

```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

**penup()**

**pendown()**

**pensize(width)**

**pencolor(colorstring)**

**pencolor(r,g,b)**

**pencolor((r,g,b))**



```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

**fd()**

**circle()**

# 运动控制函数

控制海龟行进：走直线 & 走曲线

- `turtle.forward(d)` 别名 `turtle.fd(d)`

向前行进，海龟走直线

- `d`: 行进距离，可以为负数

# 运动控制函数

控制海龟行进：走直线 & 走曲线

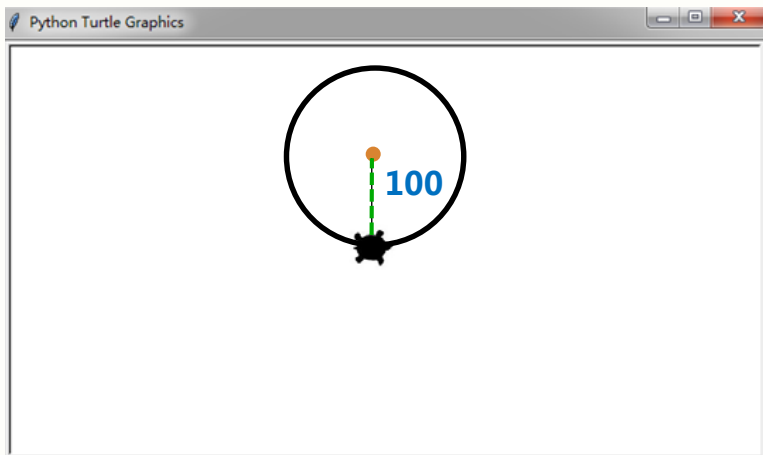
- `turtle.circle(r, extent=None)`

根据半径`r`绘制`extent`角度的弧形

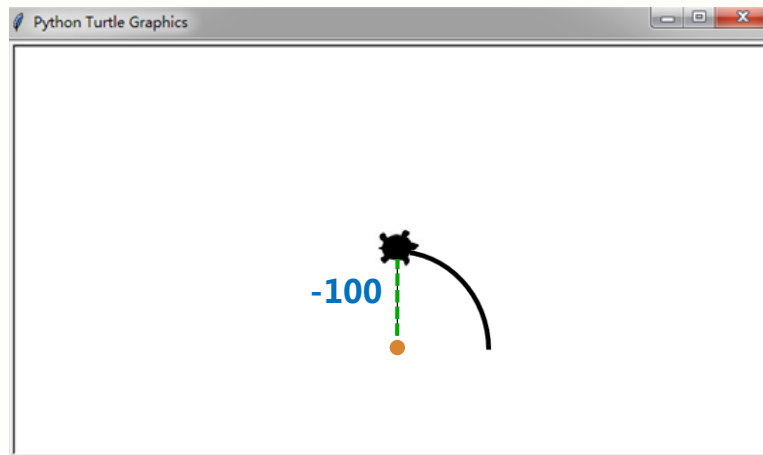
- `r`: 默认圆心在海龟左侧`r`距离的位置
- `extent`: 绘制角度，默认是360度整圆

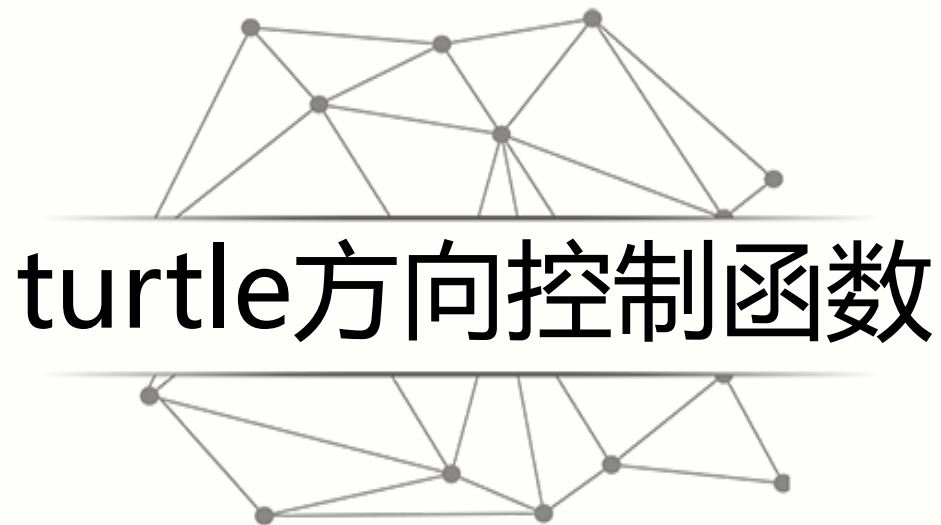
# 运动控制函数

`turtle.circle(100)`



`turtle.circle(-100,90)`





turtle方向控制函数



```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

**seth()**

# 方向控制函数

控制海龟面对方向: 绝对角度 & 海龟角度

- `turtle.setheading(angle)`      别名      `turtle.seth(angle)`

改变行进方向，海龟走角度

- **angle**: 行进方向的绝对角度

# 方向控制函数

`turtle.seth(45)`



`turtle.seth(-135)`



# 方向控制函数

控制海龟面对方向: 绝对角度 & 海龟角度

- `turtle.left(angle)`      海龟向左转
- `turtle.right(angle)`      海龟向右转
- **angle**: 在海龟当前行进方向上旋转的角度

```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

**seth**(angle)



# 循环语句与range()函数

```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

**for 和 in 保留字**

**range()**

# 循环语句

按照一定次数循环执行一组语句

**for** <变量> **in** range(<次数>):

<被循环执行的语句>

- <变量>表示每次循环的计数，0到（<次数>-1）



# 循环语句

```
>>> for i in range(5):  
    print(i)
```

0

1

2

3

4

```
>>> for i in range(5):  
    print("Hello:",i)
```

Hello: 0

Hello: 1

Hello: 2

Hello: 3

Hello: 4

# range()函数

## 产生循环计数序列

- range(N)

产生 0 到 N-1的整数序列，共N个

range(5)

0, 1, 2, 3, 4

- range(M, N)

产生 M 到 N-1的整数序列，共N-M个

range(2, 5)

2, 3, 4

```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

*for* i *in* range(N):

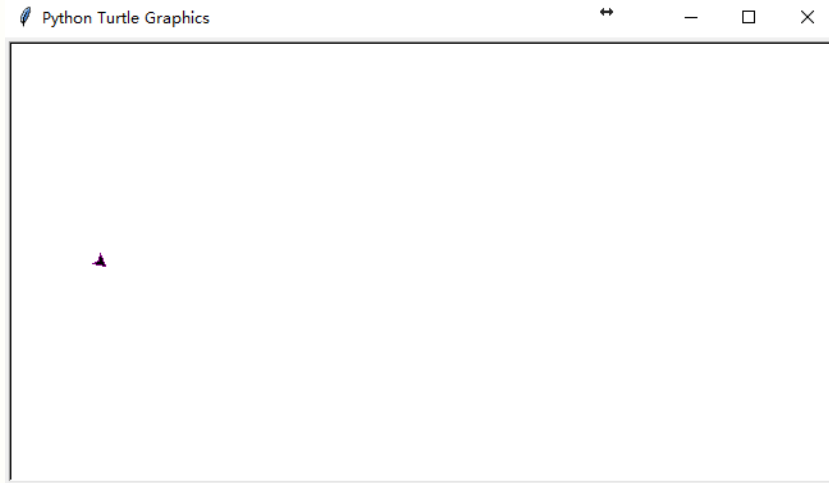
range(N)

range(M, N)

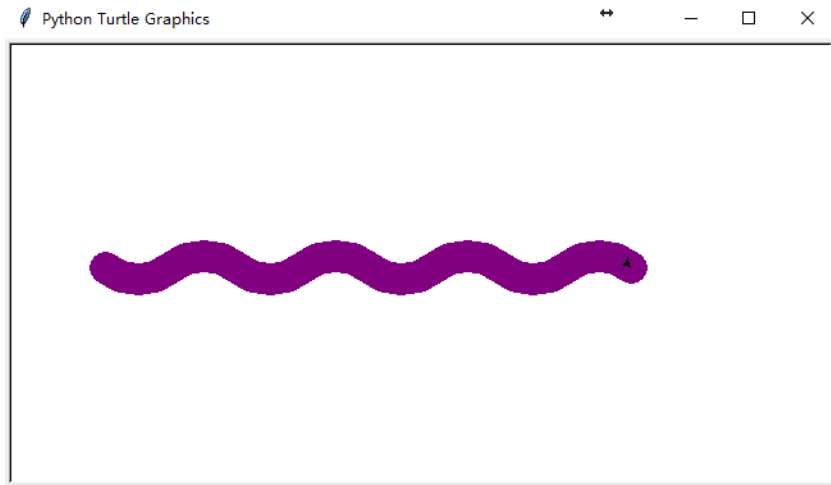


# "Python蟒蛇绘制"代码分析

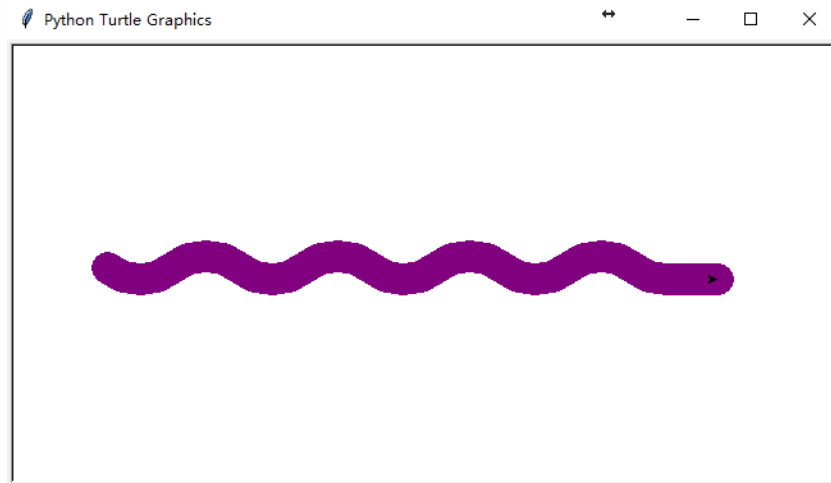
```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```



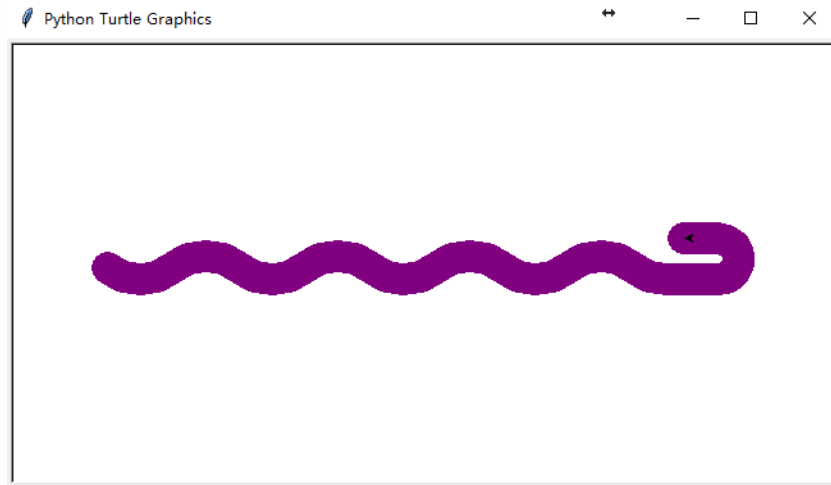
```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```



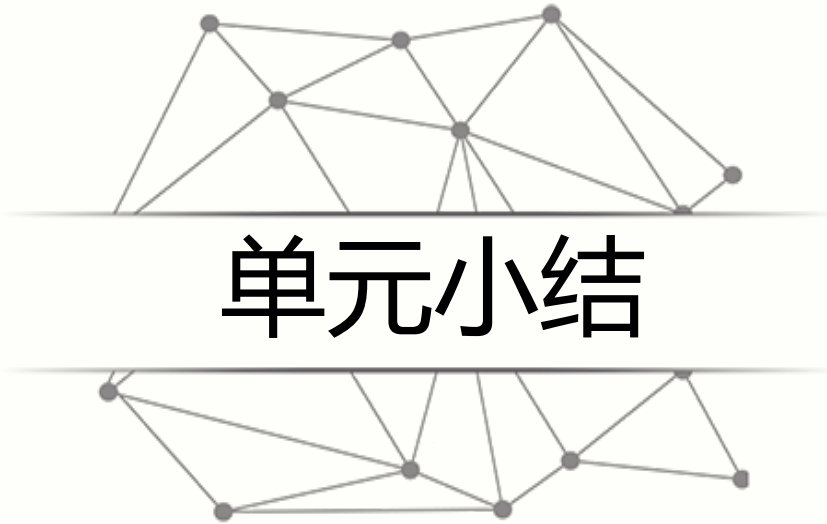
```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```



```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```







# 单元小结

# turtle程序语法元素分析

- 库引用: `import`、`from...import`、`import...as...`
- `penup()`、`pendown()`、`pensize()`、`pencolor()`
- `fd()`、`circle()`、`seth()`
- 循环语句 : `for`和`in`、`range()`函数



