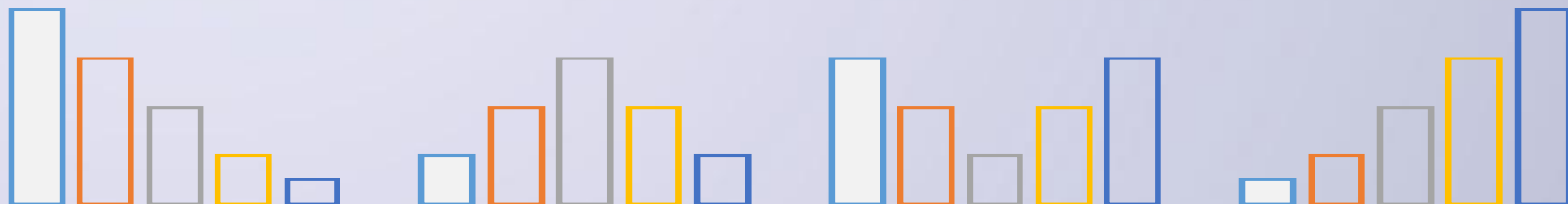


# Python语言程序设计

成都信息工程大学区块链产业学院

刘硕

# 第2章 Python程序的基本元素和 代码规范及示例



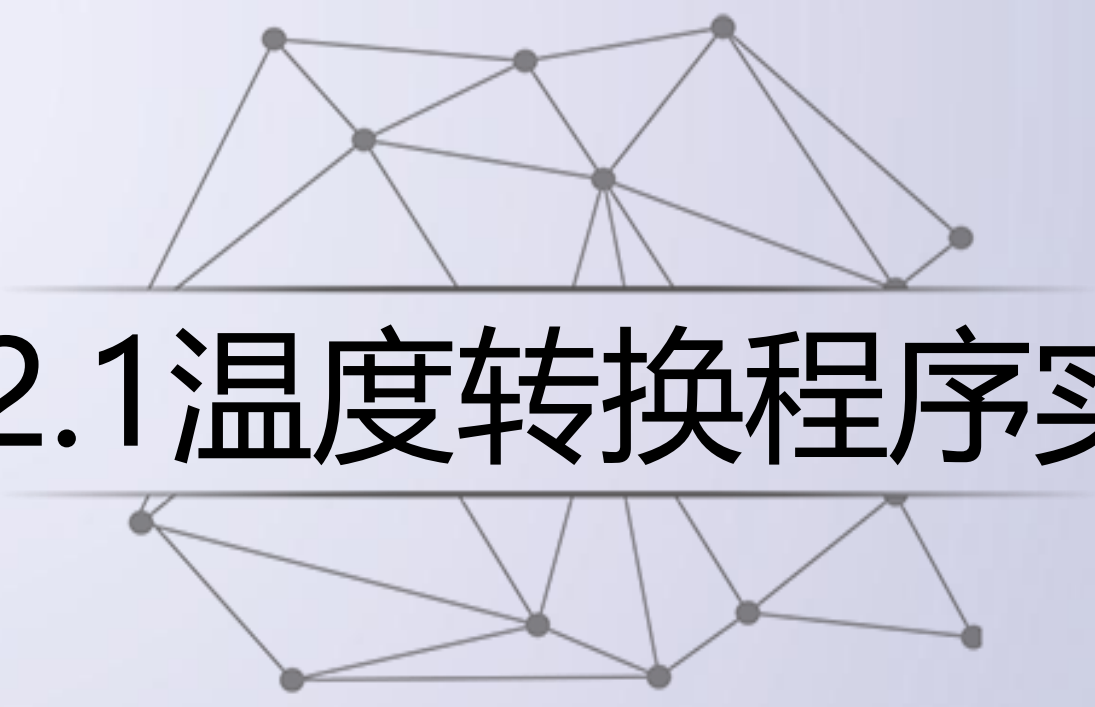
# 目录

**2.1 温度转换程序示例**

**2.2 Python语法元素分析**

**2.3 Python的对象及turtle库**

**2.4 Python的编码规范**



## 2.1 温度转换程序实例



# 温度体系


温度刻画存在不同体系，摄氏度以1标准大气压下水的结冰点为0度，沸点为100度，将温度进行等分刻画。华氏度以1标准大气压下水的结冰点为32度，沸点为212度，将温度进行等分刻画。



# 温度转换实例

问题：如何利用Python程序进行摄氏度和华氏度之间的转换

■ 步骤1：分析问题的计算部分：采用公式转换方式解决计算问题



# 温度转换实例

## ■ 步骤2：确定功能


输入：华氏或者摄氏温度值、温度标识

处理：温度转化算法

输出：华氏或者摄氏温度值、温度标识

F表示华氏度，82F表示华氏82度

C表示摄氏度，28C表示摄氏28度



# 温度转换实例

## ■ 步骤3：设计算法

根据华氏和摄氏温度定义，转换公式如下：

$$C = (F - 32) / 1.8$$

$$F = C * 1.8 + 32$$

其中，C表示摄氏温度，F表示华氏温度



# 温度转换实例

## ■ 步骤4：编写程序

```
#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")
```

编写上述代码，并保存为TempConvert.py文件


```
>>> x=input('请输入:')
请输入:
>>> input('请输入字符串:')
请输入字符串: df
'df'
```

```
s=input('请输入数字:')
print(int(s)**2)
print(s*2)
```

```
请输入数字: 34
1156
3434
```

```
>>> x=1
>>> eval(x)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    eval(x)
TypeError: eval() arg 1 must be a string, bytes or code object
>>> 1+2
3
>>> '1+2'
'1+2'
>>> s='1+2'
>>> eval(s)
3
```

```
>>> z="{ 'name': 'linux', 'age': 18 }"
>>> type(z)
<class 'str'>
>>> z1=eval(z)
>>> z1
{'name': 'linux', 'age': 18}
>>> type(z1)
<class 'dict'>
```



# 温度转换实例

## ■ 步骤5：调试、运行程序

在系统命令行上运行如下命令执行程序：

```
C:\>python TempConvert.py
```

或者：使用IDLE打开上述文件，按F5运行（推荐）

输入数值，观察输出



## 2.2 Python语法元素分析



# 程序的格式框架

Python语言采用严格的“缩进”来表明程序的格式框架。缩进指每一行代码开始前的空白区域，用来表示代码之间的包含和层次关系。

1个缩进 = 4个空格

- 用以在Python中标明代码的层次关系
- 缩进是Python语言中表明程序框架的唯一手段



# 程序的格式框架

## 单层缩进

```
#e1.1TempConvert.py
TempStr = input("请输入带有符号
if TempStr[-1] in ['F','f']:
    ↪ C = (eval(TempStr[0:-1]) -
        print("转换后的温度是{:.2f}C
elif TempStr[-1] in ['C','c']:
    ↪ F = 1.8*eval(TempStr[0:-1])
        print("转换后的温度是{:.2f}F
else:
    ↪ print("输入格式错误")
```

## 多层缩进

```
DARTS = 1000
hits = 0.0
clock()
for i in range(1, DARTS):
    ↪ x, y = random(), random()
        ↪ dist = sqrt(x ** 2 + y ** 2)
            if dist <= 1.0:
                ↪ hits = hits + 1
pi = 4 * (hits/DARTS)
print("Pi的值是{:.2f}".format
```

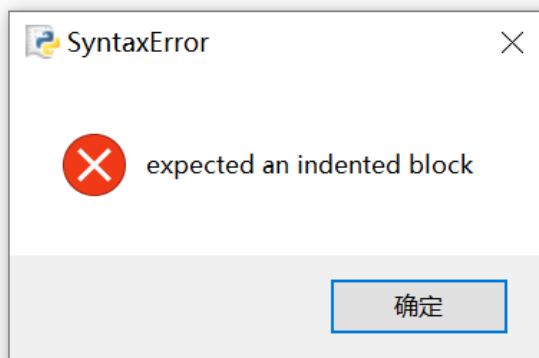
# 缩进

## 缩进表达程序的格式框架

- 严格明确：缩进是语法的一部分，缩进不正确程序运行错误
- 所属关系：表达代码间包含和层次关系的唯一手段
- 长度一致：程序内一致即可，一般用4个空格或1个TAB

File Edit Format Run Options Window Help

```
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")
```





注释：程序员在代码中加入的说明信息，不被计算机执行

注释的两种方法：

- 单行注释以#开头

```
#Here are the comments
```

- 多行注释以 ' ' ' 开头和结尾

```
' ' '
```

```
This is a multiline comment  
used in Python
```

```
' ' '
```





# 命名与保留字

- 常量：程序中值不发生改变的元素
- 变量：程序中值发生改变或者可以发生改变的元素

Python语言允许采用大写字母、小写字母、数字、下划线(\_)和汉字等字符及其组合给变量命名，但名字的首字符不能是数字，中间不能出现空格，长度没有限制

注意：标识符对大小写敏感，python和Python是两个不同的名字

```
#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")
```

**变量：**程序中用于保存和表示数据的占位符号

# 变量

## 用来保存和表示数据的占位符号

—变量采用标识符(名字) 来表示，关联标识符的过程叫命名

TempStr是变量名字

—可以使用等号(=)向变量赋值或修改值，=被称为赋值符号

TempStr = "82F"      #向变量TempStr赋值"82F"

# 命名

## 关联标识符的过程

—命名规则: 大小写字母、数字、下划线和中文等字符及组合

如: TempStr, Python\_Great, 这是门Python好课

—注意事项: 大小写敏感、首字符不能是数字、不与保留字相同

Python和python是不同变量, 123Python是不合法的



# 命名与保留字

- ✓ 保留字，也称为关键字，指被编程语言内部定义并保留使用的标识符。
- ✓ 程序员编写程序不能定义与保留字相同的标识符。
- ✓ 每种程序设计语言都有一套保留字，保留字一般用来构成程序整体框架、表达关键值和具有结构性的复杂语义等。
- ✓ 掌握一门编程语言首先要熟记其所对应的保留字。



# 命名与保留字

## ✓ Python 3.x保留字列表 (33个)

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


— 保留字是编程语言的基本单词，大小写敏感

**if** 是保留字，**If** 是变量

```
#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")
```

**变量 命名 保留字**



# 数据类型



```
#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")
```

**数据类型：**字符串、整数、浮点数、列表

# 数据类型

**10,011,101 该如何解释呢?**

**—这是一个二进制数字 或者 十进制数字**

**作为二进制数字, 10,011,101的值是十进制157**

**—这是一段文本 或者 用逗号,分隔的3个数字**

**作为一段文本, 逗号是文本中的一部分, 一共包含10个字符**

# 数据类型

## 供计算机程序理解的数据形式

—程序设计语言不允许存在语法歧义，需要定义数据的形式

需要给10,011,101关联一种计算机可以理解的形式

—程序设计语言通过一定方式向计算机表达数据的形式

"123"表示文本字符串123，123则表示数字123

# 数据类型

**10,011,101**

- **整数类型:**      10011101
- **字符串类型:**    "10,011,101"
- **列表类型:**      [10, 011, 101]

```
#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")
```

**字符串：**由0个或多个字符组成的有序字符序列

# 字符串

- Python语言中，字符串是用两个双引号 “ ” 或者单引号 ‘ ’ 括起来的一个或多个字符。
- Python字符串的两种序号体系



# 字符串的使用

使用[ ]获取字符串中一个或多个字符

—索引：返回字符串中单个字符<字符串>[M]

"请输入带有符号的温度值："[0]或者TempStr[-1]

—切片：返回字符串中一段字符串<字符串>[M: N]

"请输入带有符号的温度值："[1:3] 或者TempStr[0:-1]

```
#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")
```

**列表类型：**由0个或多个数据组成的有序序列



# 列表类型

由0个或多个数据组成的有序序列

—列表使用[ ]表示，采用逗号(,)分隔各元素

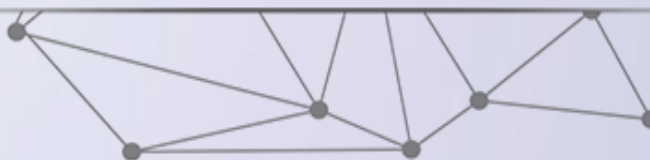
[ 'F', 'f' ]表示两个元素'F'和'f'


—使用保留字 in 判断一个元素是否在列表中

TempStr[-1] in [ 'C', 'c' ]判断前者是否与列表中某个元素相同



# 语句与函数





# 赋值语句

■ Python语言中，`=` 表示“赋值”，即将等号右侧的值计算后将结果值赋给左侧变量，包含等号（`=`）的语句称为“赋值语句”

■ 同步赋值语句：同时给多个变量赋值

`<变量1>, ..., <变量N> = <表达式1>, ..., <表达式N>`

```
#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")
```

**赋值语句：** 由赋值符号构成的一行代码

# 赋值语句

由赋值符号构成的一行代码

—赋值语句用来给变量赋予新的数据值

`C=(eval(TempStr[0:-1])-32)/1.8` #右侧运算结果赋给变量C

—赋值语句右侧的数据类型同时作用于变量

`TempStr=input("")` #input()返回一个字符串, TempStr也是字符串

# 赋值语句

例：将变量x和y交换

■采用单个赋值，需要3行语句：


即通过一个临时变量t缓存x的原始值，然后将y值赋给x，再将x的原始值通过t赋值给y。

■采用同步赋值语句，仅需要一行代码：

```
>>>t=x  
>>>x=y  
>>>y=t
```

等价于

```
>>>x, y=y, x
```



# input()函数

- 获得用户输入之前，input()函数可以包含一些提示性文字  
<变量> = input(<提示性文字>)

```
>>>input("请输入: ")
```

```
请输入: python
```

```
'python'
```

```
>>> input("请输入: ")
```

```
请输入: 1024.256
```

```
'1024.256'
```

**input():** 从控制台获得用户输入的函数

# 输入函数

## input()

从控制台获得用户输入的函数

—input()函数的使用格式：

<变量> = input(<提示信息字符串>)

—用户输入的信息以字符串类型保存在<变量>中

TempStr = input("请输入") # TempStr保存用户输入的信息

```
>>>input("请输入: ")
请输入: python
'python'

>>> input("请输入: ")
请输入: 1024.256
'1024.256'
```



```
#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")
```

**分支语句：** 由判断条件决定程序运行方向的语句



# 分支语句

- 分支语句是控制程序运行的一类重要语句，它的作用是根据判断条件选择程序执行路径，使用方式如下：

```
if <条件1>:  
    <语句块1>  
elif <条件2>:  
    <语句块2>  
...  
else:  
    <语句块N>
```

# 函数

根据输入参数产生不同输出的功能过程

—类似数学中的函数，  $y = f(x)$

`print("输入格式错误")` #打印输出

"输入格式错误"

—函数采用 `<函数名>(<参数>)` 方式使用

`eval(TempStr[0:-1])`      # `TempStr[0:-1]`是参数

```
#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")
```

赋值语句

分支语句

函数



# eval () 函数

- `eval(<字符串>)` 函数是Python语言中一个十分重要的函数，它能够以Python表达式的方式解析并执行字符串，将返回结果输出

```
>>>x = 1
>>>eval("x + 1")
2
>>>eval("1.1 + 2.2")
3.3
```

# 评估函数 eval()

去掉参数最外侧引号并执行余下语句的函数

- eval()函数的基本使用格式:

`eval(<字符串或字符串变量>)`

```
>>> eval("1")
1
>>> eval("1+2")
3
```

```
>>> eval('"1+2"') '1+2'
>>> eval('print("Hello")')
Hello
```

# 评估函数 eval()

去掉参数最外侧引号并执行余下语句的函数

```
eval(TempStr[0:-1])
```

如果TempStr[0:-1]值是"12.3"，输出是：

12.3

## • 部分内置函数的分类

数字相关

Bool int float complex数据类型

Bin oct hex进制转换

Abs sum数字运算

数据结构相关

List reversed slice Bytes repr序列

dict set数据集合

len相关内置函数

globals() locals() 作用域相关

range 迭代生成器相关

eval() 字符串类型执行

dir查看内置属性

open() 文件操作相关

hash()内存相关

Input()print()输入输出





# 输出函数

- `print()`函数用来输出字符信息，或以字符形式输出变量。
- `print()`函数可以输出各种类型变量的值。
- `print()`函数通过`%`来选择要输出的变量。

# 输出函数 print()

以字符形式向控制台输出结果的函数

—print()函数的基本使用格式：

`print(<拟输出字符串或字符串变量>)`

—字符串类型的一对引号仅在程序内部使用，输出无引号

`print("输入格式错误")`    # 向控制台输出输入格式错误



## 实例

- 用户输入两个数字，计算它们的平均数，并输出平均数

```
num1 = input("The first number is")
num2 = input("The second number is")
avg_num = (float(num1) + float(num2)) / 2
print("The average number is %f" % avg_num)
```



# 循环语句

- 循环语句：控制程序运行，根据判断条件或计数条件确定一段程序的运行次数

- 遍历循环，基本过程如下

- for i in range (<计数值>):  
    <表达式1>

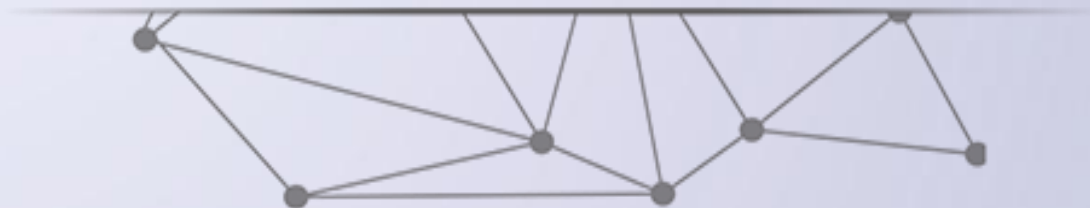
- 例如，使某一段程序连续运行10次

- for i in range (10):  
    <源代码>

- 其中，变量i用于计数



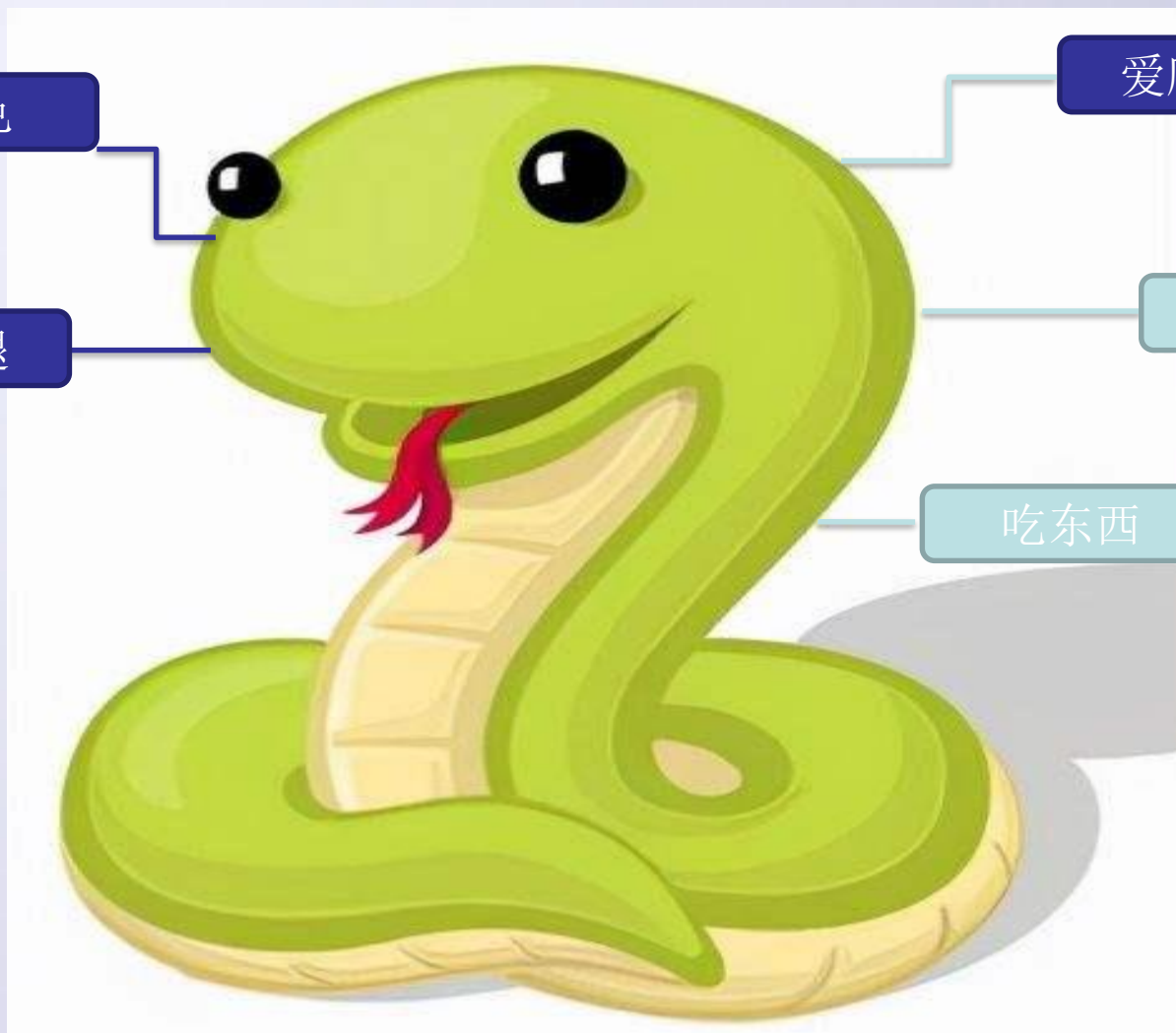
## 2.3python的对象及turtle库



# Python的对象模型

- 对象是python语言中最基本的概念，在python中处理的一切都是对象。python中有许多内置对象可供编程者使用，内置对象可直接使用，如数字、字符串、列表、del命令等；非内置对象需要导入模块才能使用，如正弦函数 $\sin(x)$ ，随机数产生函数`random()`等。

# 给大家介绍对象



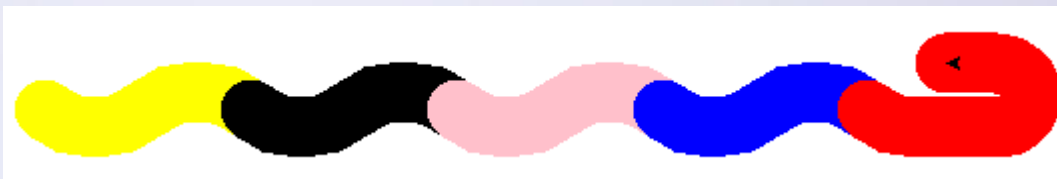
绿色

没有腿

爱爬行

咬人

吃东西



#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)
```

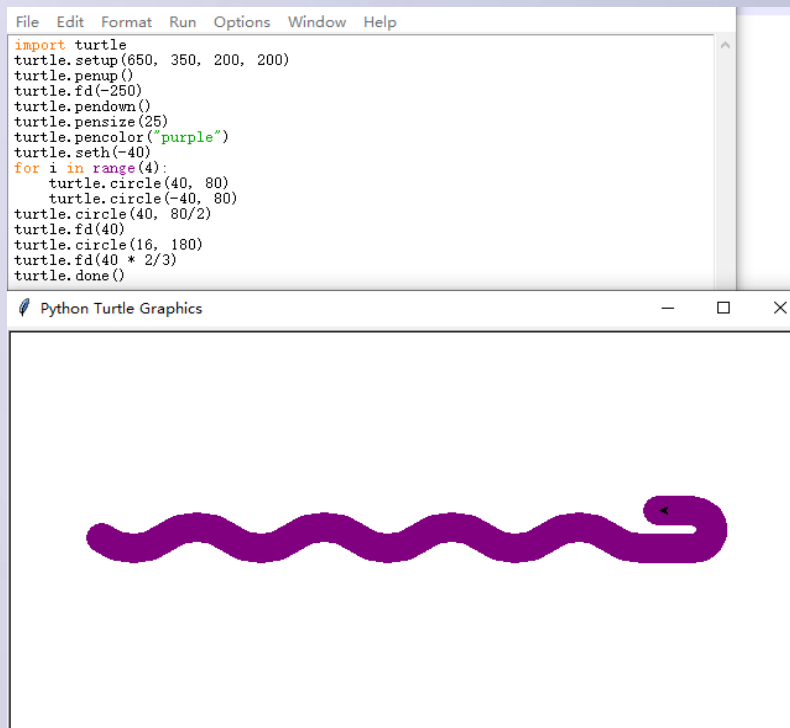
```
turtle.fd(40 * 2/3)
```

```
turtle.done()
```

*import* 保留字

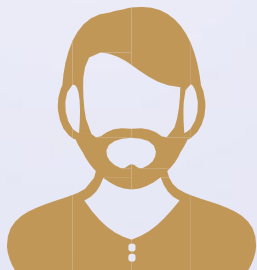
程序关键 引入了一个绘图库

名字叫: **turtle**





# 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**

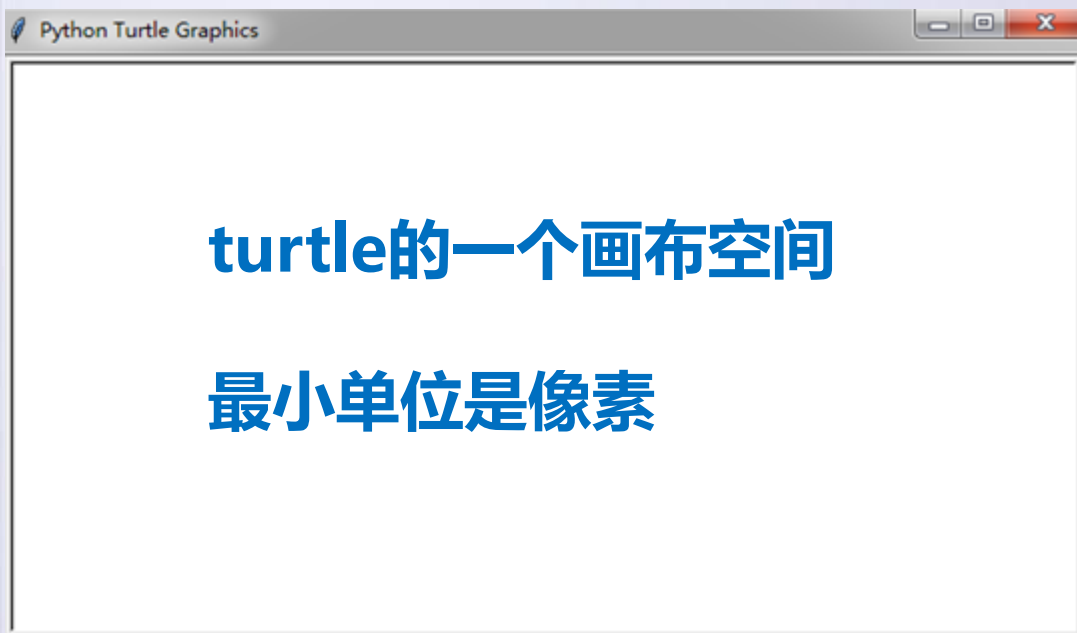
- turtle绘图体系：1969年诞生，主要用于程序设计入门**
- Python语言的**标准库**之一**
- 入门级的图形绘制函数库**

# 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



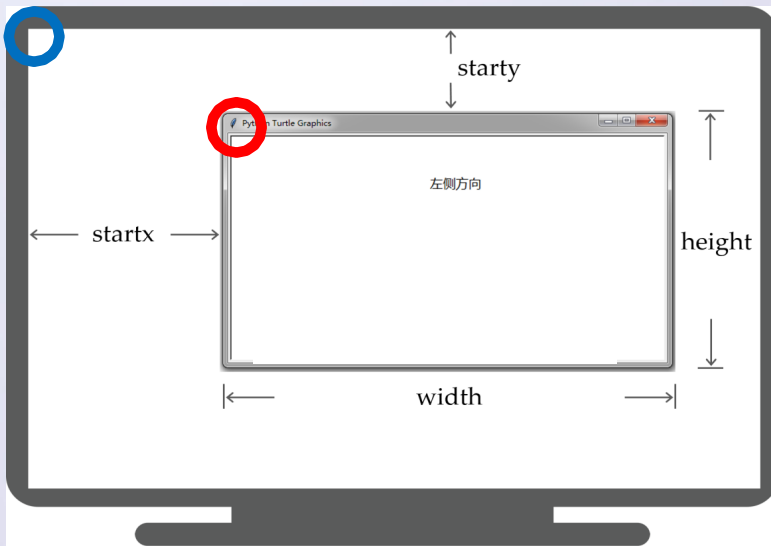
# turtle的绘图窗体



# 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



# turtle的绘图窗体

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

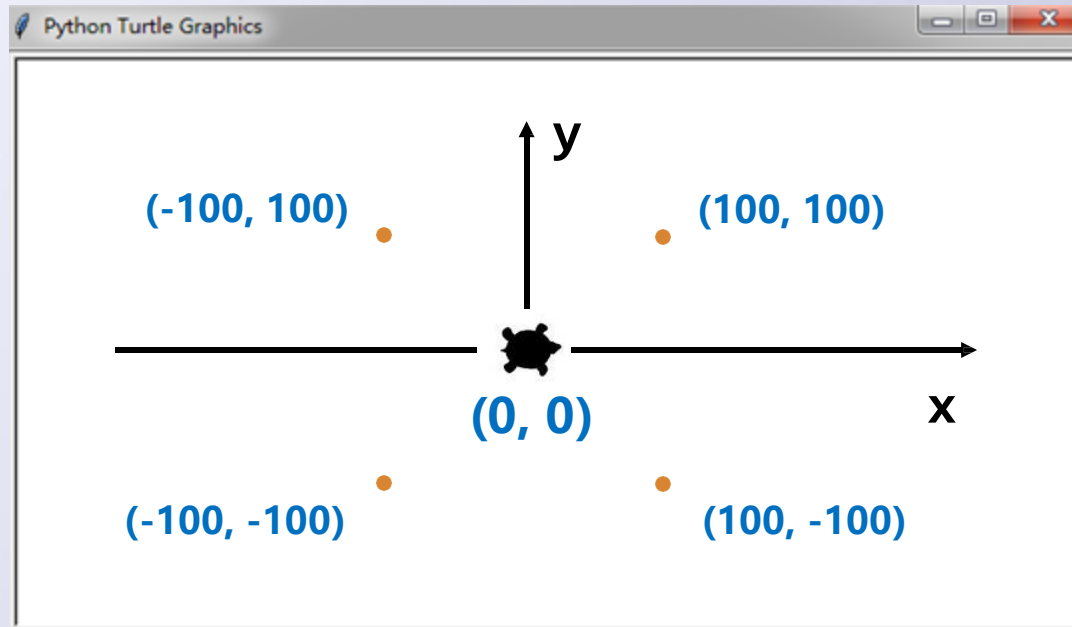


```
turtle.setup(800,800)
```



# turtle空间坐标体系

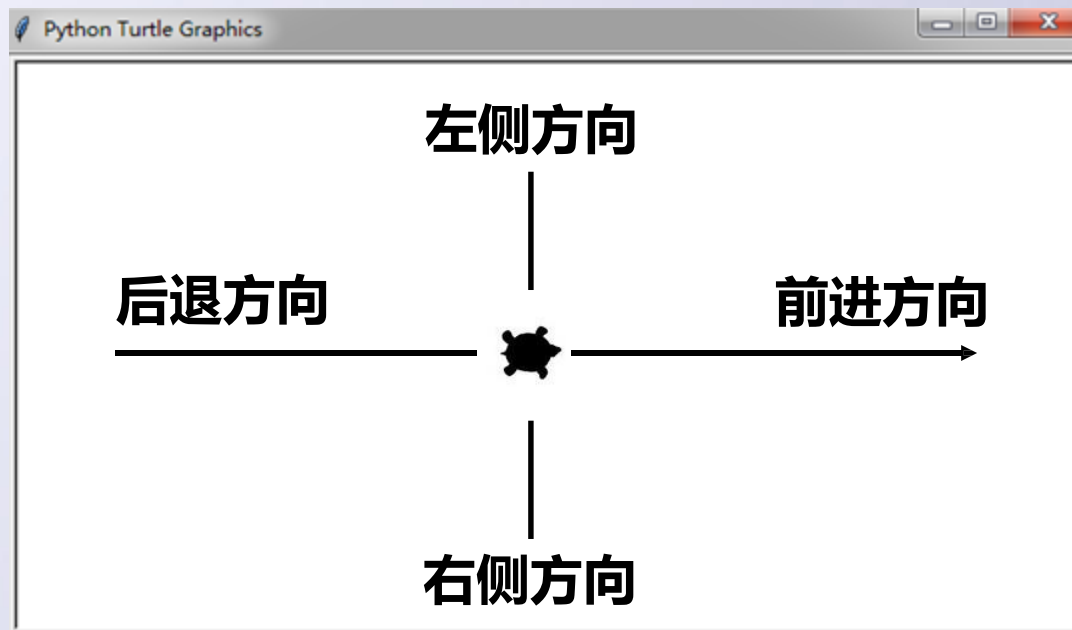
绝对坐标



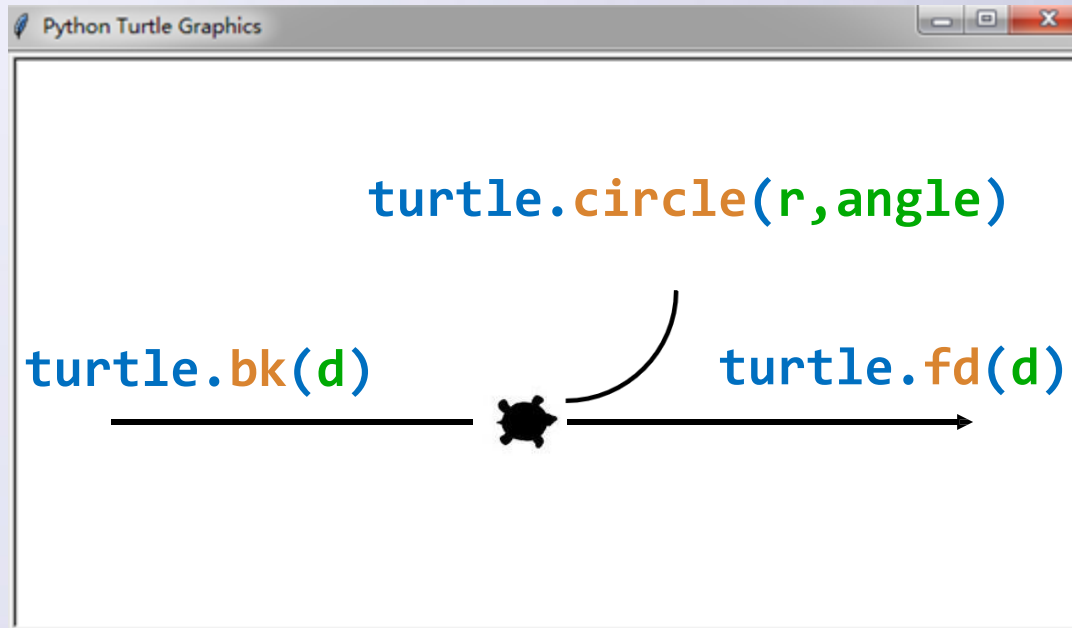


# turtle空间坐标体系

海龟坐标

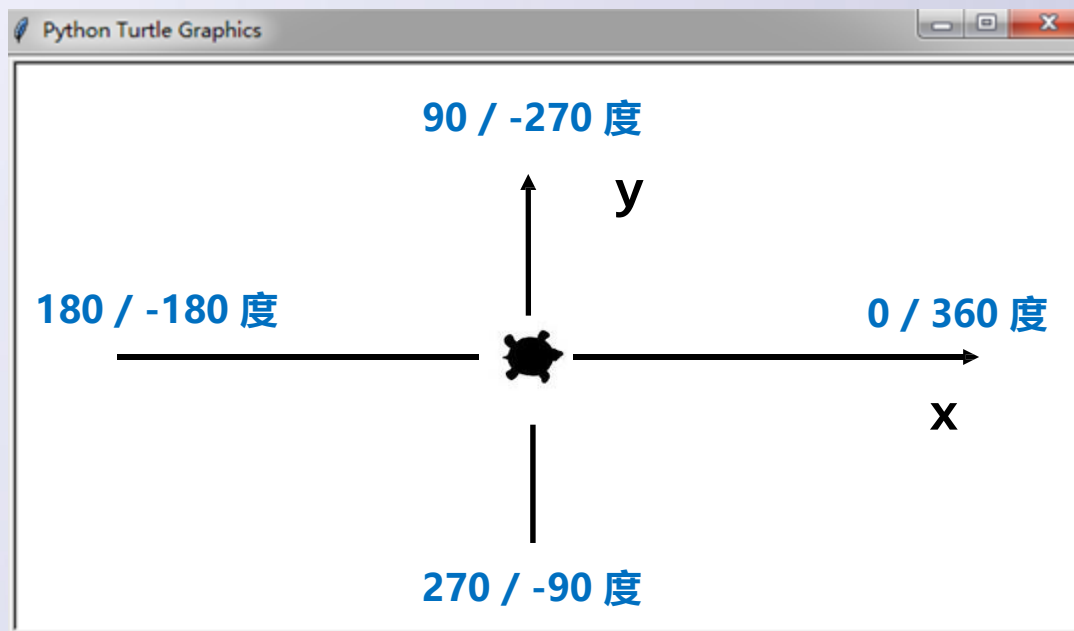


# turtle空间坐标体系



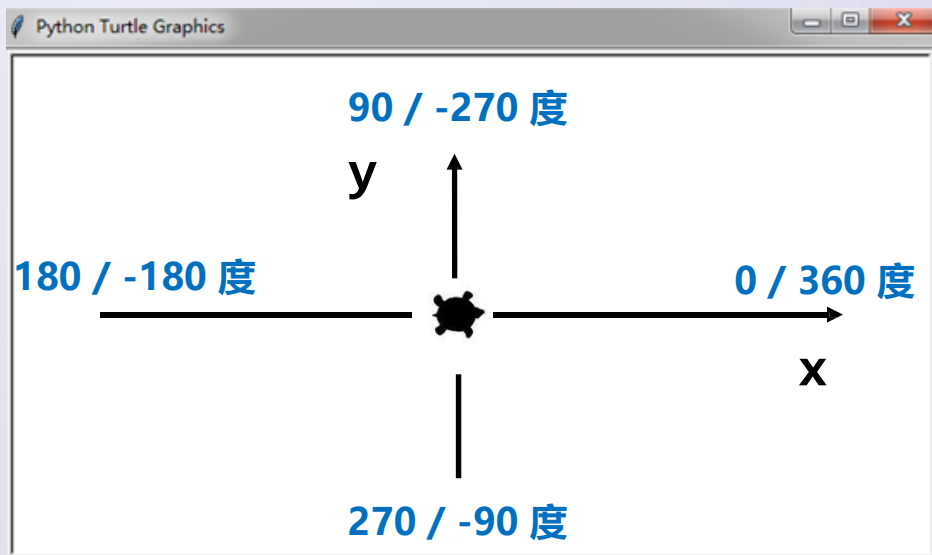
# turtle角度坐标体系

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)`

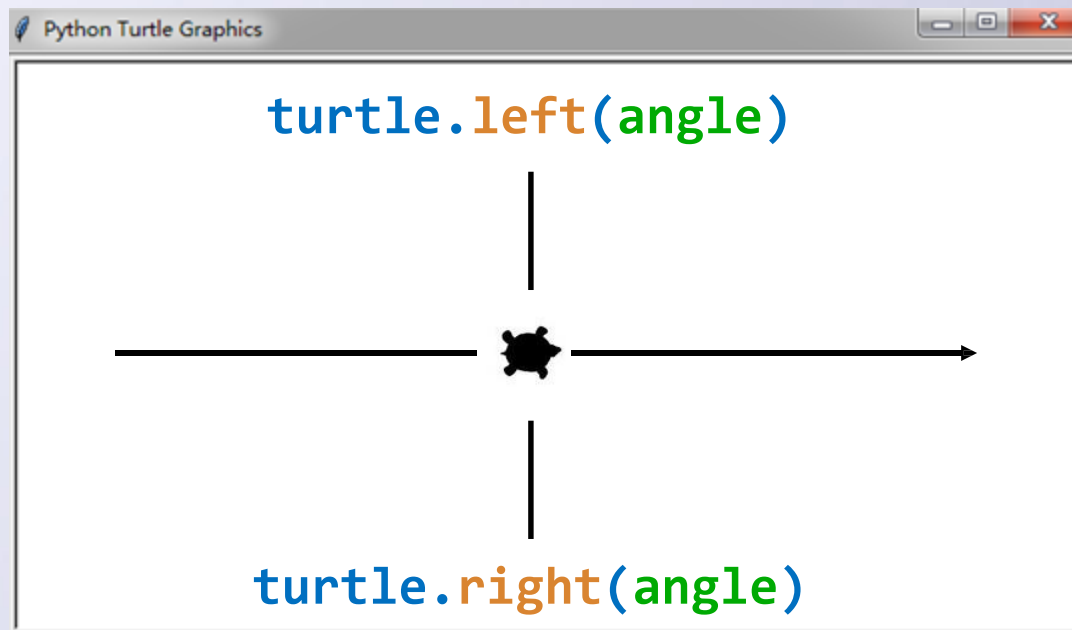


`turtle.seth(-135)`



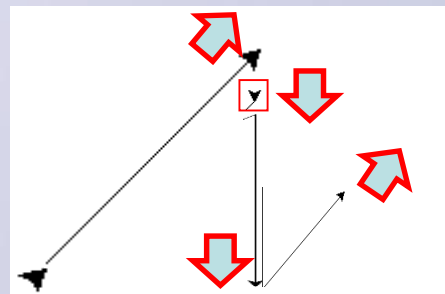
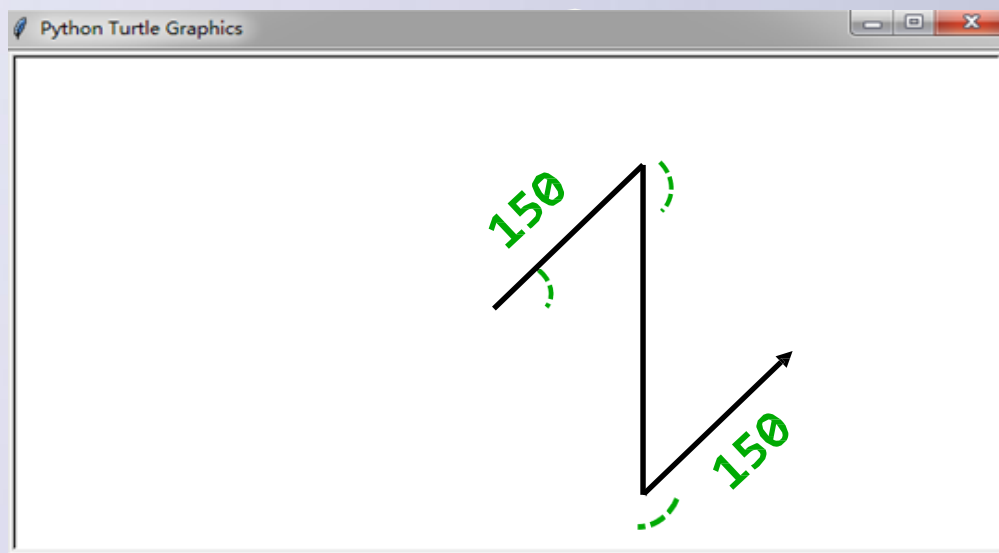
# turtle角度坐标体系

海龟角度



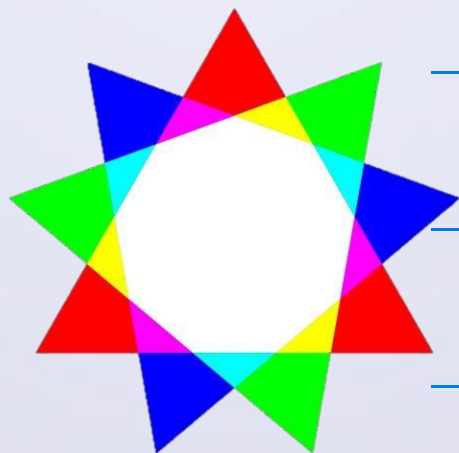
# turtle角度坐标体系

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



# RGB色彩模式

由三种颜色构成的万物色



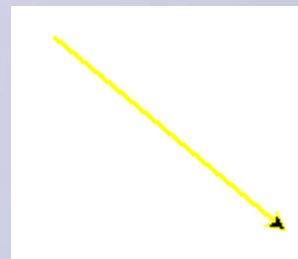
- RGB指红蓝绿三个通道的颜色组合
- 覆盖视力所能感知的所有颜色
- RGB每色取值范围0-255整数或0-1小数




# 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
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

```
>>> import turtle
>>> turtle.pensize(2)
>>> turtle.seth(-40)
>>> turtle.pencolor(1, 1, 0)
>>> turtle.fd(150)
```





# 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**



# 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>)

```
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更多用法

使用from和import保留字共同完成

from <库名> import <函数名>

from <库名> import \*

<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()
```



```
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()
```

# 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



# 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

from 模块名 import 对象名[ as 别名] #可以减少查询次数，提高执行速度

```
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()
```



```
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()
```

# 模块导入与使用

## ❖ import 模块名

```
>>>import math
```

```
>>>math.sin(0.5)           #求0.5的正弦
```

```
>>>import random
```

```
>>>x=random.random( )      #获得[0,1) 内的随机小数
```

```
>>>y=random.random( )
```

```
>>>n=random.randint(1,100) #获得[1,100]上的随机整数
```

- ✓ 可以使用**dir**函数查看任意模块中所有的对象列表，如果调用不带参数的**dir()**函数，则返回当前所有名字列表。
- ✓ 可以使用**help**函数查看任意模块或函数的使用帮助。

```
>>> dir()
['__annotations__', '__builtins__',
>>> dir(turtle)
['Canvas', 'Pen', 'RawPen', 'RawTurtle',
 '__all__', '__builtins__', '__cached__',
 '__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'tg_screen_functions', 'tg_turtle',
 'color', 'colormode', 'config dict', ...]
```

# 模块导入与使用

- ❖ 如果需要导入多个模块，一般建议按如下顺序进行导入：
  - ✓ 标准库
  - ✓ 成熟的第三方扩展库
  - ✓ 自己开发的库

# 模块导入与使用

- Python默认安装仅包含部分基本或核心模块，但用户可以安装大量的扩展模块，**pip**是管理模块的重要工具。
- 在Python启动时，仅加载了很少的一部分模块，在需要时由程序员显式地加载（可能需要先安装）其他模块。
- 减小运行的压力，仅加载真正需要的模块和功能，且具有很强的可扩展性。
- 可以使用**sys.modules.items()**显示所有预加载模块的相关信息。

```
>>> import sys
>>> sys.modules.items()
dict_items([('sys', <module 'sys' (built-in)>), ('builtins', <module 'builtins' (built-in)>), ('_frozen_importlib', <module '_frozen_importlib' (frozen)>), ('_imp', <module '_imp' (built-in)>), ('_thread', <module '_thread' (built-in)>), ('_warnings', <module '_warnings' (built-in)>), ('_weakref', <module '_weakref' (built-in)>), ('_io', <module '_io' (built-in)>), ('marshal', <module 'marshal' (built-in)>), ('nt', <module 'nt' (built-in)>), ('winreg', <module 'winreg' (built-in)>), ('_frozen_importlib_external', <module '_frozen_importlib_external' (frozen)>), ('time', <module 'time' (built-in)>), ('zipimport', <module 'zipimport' (frozen)>), ('_codecs', <module '_codecs' (built-in)>), ('codecs', <module 'codecs' from 'D:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python310\\lib\\codecs.py'>), ('encodings', <module 'encodings' from 'D:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python310\\lib\\encodings\\__init__.py'>), ('encodings.aliases', <module 'encodings.aliases' from 'D:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python310\\lib\\encodings\\aliases.py'>), ('encodings.gbk', <module 'encodings.gbk' from 'D:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python310\\lib\\encodings\\gbk.py'>), ('encodings.utf_8', <module 'encodings.utf_8' from 'D:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python310\\lib\\encodings\\utf_8.py'>), ('_codecs_cn', <module '_codecs_cn' (built-in)>), ('multibytecodec', <module 'multibytecodec' (built-in)>), ('encodings.gbk', <module 'encodings.gbk' from 'D:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python310\\lib\\encodings\\gbk.py'>), ('_signal', <module '_signal' (built-in)>), ('_abc', <module '_abc' (built-in)>), ('abc', <module 'abc' from 'D:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python310\\lib\\abc.py'>), ('io', <module 'io' from 'D:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python310\\lib\\io.py'>), ('__main__', <module '__main__' (built-in)>), ('_stat', <module '_stat' (built-in)>), ('stat', <module 'stat' from 'D:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python310\\lib\\stat.py'>), ('collections.abc', <module 'collections.abc' from 'D:\\Users\\PC\\AppData\\Local\\Pr
```



# turtle画笔控制函数

```
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()
```

**penup(), pendown()**

**pensize(), pencolor()**

# 画笔控制函数

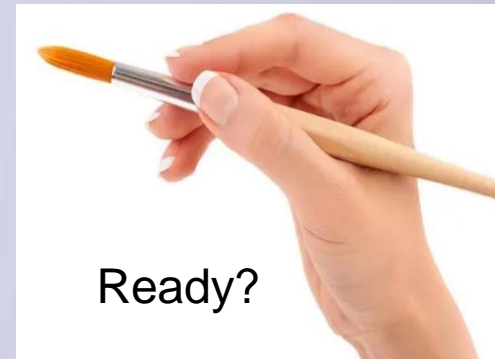
画笔操作后一直有效，一般成对出现

– `turtle.penup()`      别名  
   `turtle.pu()`

抬起画笔，海龟在飞行

– `turtle.pendown()`    别名  
   `turtle.pd()`

落下画笔，海龟在爬行







# 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，海龟的腰围

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

画笔颜色，海龟在涂装

# 画笔控制函数

**pencolor(color)的color参与可以有三种形式**

- 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

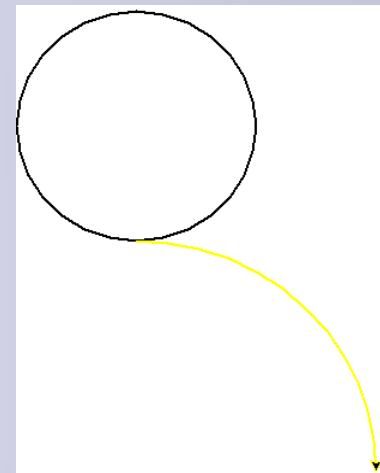
—`turtle.circle(r, extent=None)`

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

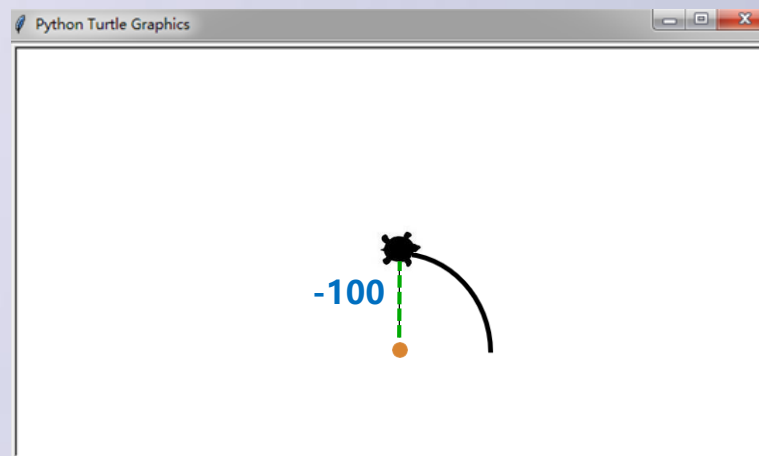
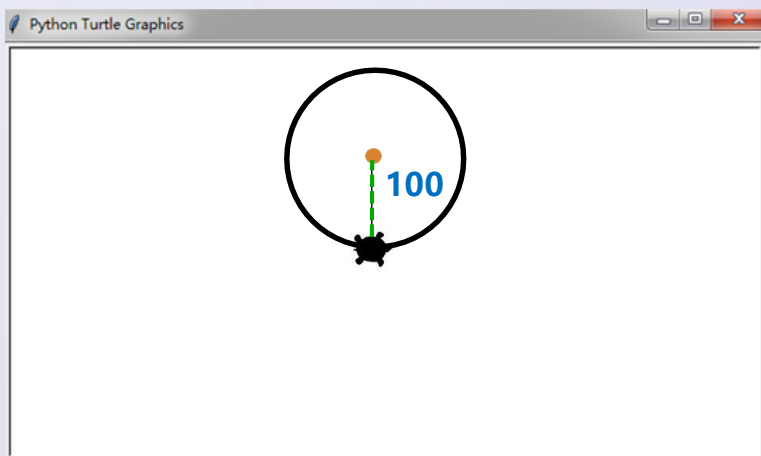
—`extent`: 绘制角度，默认是360度整圆

# 运动控制函数



`turtle.circle(100)`

`turtle.circle(-100,90)`



```
>>> import turtle
>>> turtle.pensize(2)
>>> turtle.circle(100)
>>> turtle.pencolor(1,1,0)
>>> turtle.circle(-200,90)
\\>>>
```



```
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(d)**

**circle(r, extent=None)**



```
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



# 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(M, N)

产生 M 到 N-1的整数序列，共N-M个

```
range(5)
```

```
0, 1, 2, 3, 4
```

```
range(2, 5)
```

```
2, 3, 4
```

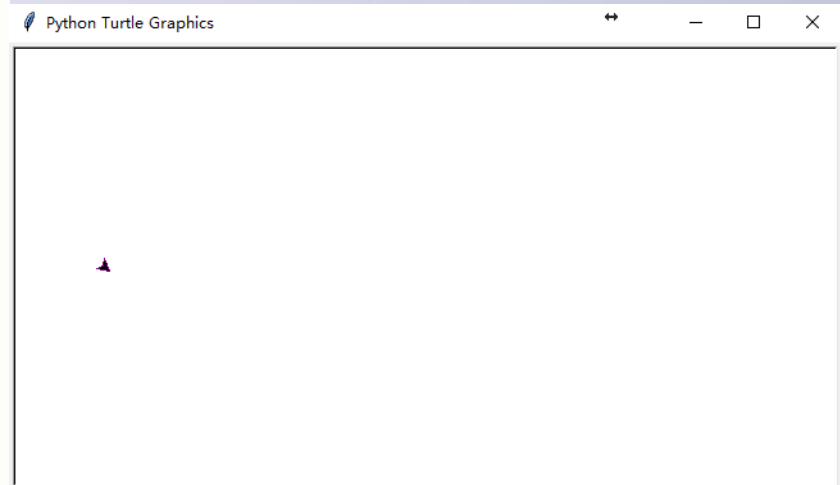


# "Python蟒蛇绘制"代码分析

```
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()
```

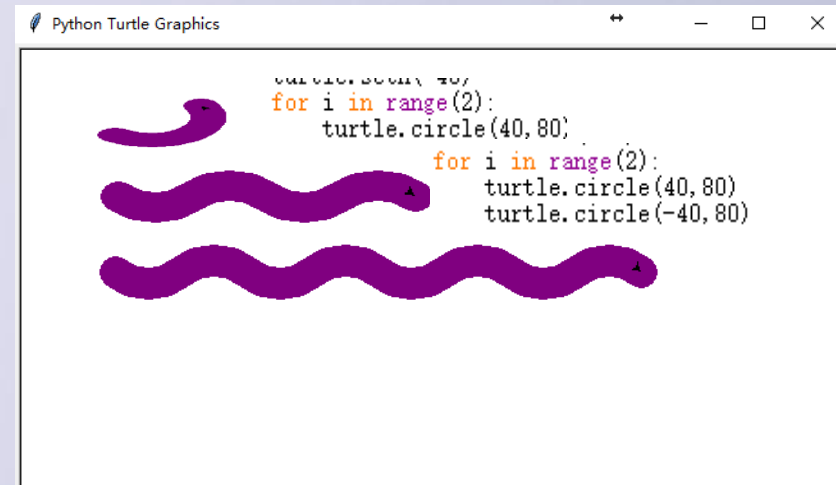
200

Show  
window

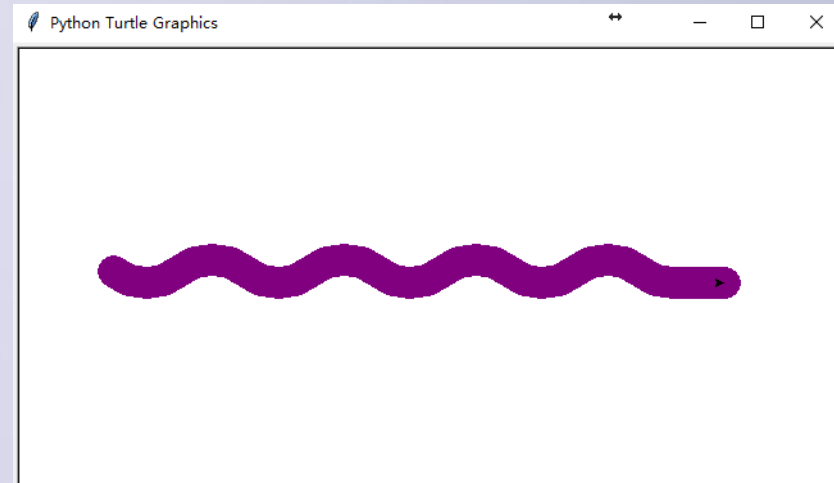


200

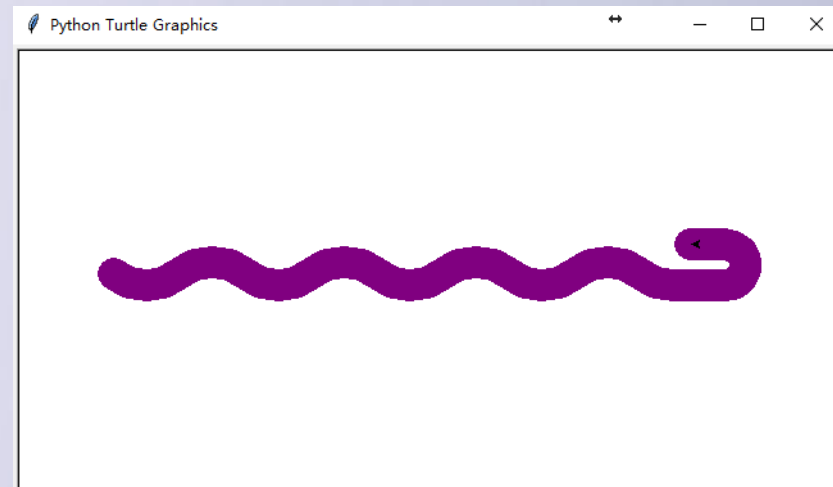
```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40) turtle.circle(16,
180) turtle.fd(40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



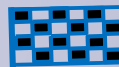




# 单元小结

# 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

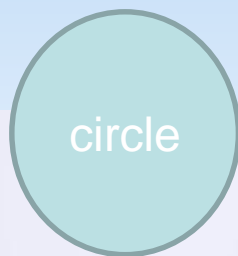


# Python的对象模型

## 常用内置对象

对象类型	类型名称	示例	简要说明
数字	int, float, complex	1234, 3.14, 1.3e5, 3+4j	数字大小没有限制，内置支持复数及其运算
字符串	str	'swfu', "I'm student", '''Python ''', r'abc', R'bcd'	使用单引号、双引号、三引号作为定界符，以字母r或R引导的表示原始字符串
字节串	bytes	b'hello world'	以字母b引导，可以使用单引号、双引号、三引号作为定界符
列表	list	[1, 2, 3], ['a', 'b', ['c', 2]]	所有元素放在一对方括号中，元素之间使用逗号分隔，其中的元素可以是任意类型
字典	dict	{1:'food', 2:'taste', 3:'import'}	所有元素放在一对大括号中，元素之间使用逗号分隔，元素形式为“键:值”
元组	tuple	(2, -5, 6), (3,)	所有元素放在一对圆括号中，元素之间使用逗号分隔，如果元组中只有一个元素的话，后面的逗号不能省略

# Python的对象模型



Center;  
radius...

Draw;  
Move...

```
myCircle = Circle(Point(0,0), 20)
```

续表

图纸 → 房子

Class → instance

isinstance(var,type)

对象：属性+方法

对象类型	类型名称	示例	简要说明
集合	set frozenset	{'a', 'b', 'c'}	所有元素放在一对大括号中，元素之间使用逗号分隔，元素不允许重复；另外，set是可变的，而frozenset是不可变的
布尔型	bool	True, False	逻辑值，关系运算符、成员测试运算符、同一性测试运算符组成的表达式的值一般为True或False
空类型	NoneType	None	空值
异常	Exception、 ValueError、 TypeError		Python内置大量异常类，分别对应不同类型的异常
文件		f = open('data.dat', 'rb')	open是Python内置函数，使用指定的模式打开文件，返回文件对象
其他迭代对象		生成器对象、range对象、zip对象、 enumerate对象、map对象、filter 对象等等	具有惰性求值的特点
编程单元		函数（使用def定义）、类（使用 class定义）、模块（类型为 module）	类和函数都属于可调用对象，模块用来集中存放函数、类、常量或其他对象

# Python变量

- 在Python中，不需要事先声明变量名及其类型，直接赋值即可创建各种类型的对象变量。这一点适用于Python任意类型的对象。

例如语句

```
>>> x = 3
```

创建了整型变量x，并赋值为3，再例如语句

```
>>> x = 'Hello world.'
```

创建了字符串变量x，并赋值为'Hello world.'。

# Python变量

- ❖ Python属于强类型编程语言，Python解释器会根据赋值或运算来自动推断变量类型。Python还是一种动态类型语言，变量的类型也是可以随时变化的。

```
>>> x = 3
>>> print(type(x))
<class 'int'>
>>> x = 'Hello world.'
>>> print(type(x))                                #查看变量类型
<class 'str'>
>>> x = [1,2,3]
>>> print(type(x))
<class 'list'>
>>> isinstance(3, int)                             #测试对象是否是某个类型的实例
True
>>> isinstance('Hello world', str)
True
```

# Python变量

- ❖ 如果变量出现在赋值运算符或复合赋值运算符（例如+=、\*=等等）的左边则表示创建变量或修改变量的值，否则表示引用该变量的值，这一点同样适用于使用下标来访问列表、字典等可变序列以及其他自定义对象中元素的情况。

```
>>> x = 3          #创建整型变量
>>> print(x**2)
9
>>> x += 6         #修改变量值
>>> print(x)       #读取变量值并输出显示
9
>>> x = [1, 2, 3]  #创建列表对象
>>> x[1] = 5       #修改列表元素值
>>> print(x)       #输出显示整个列表
[1, 5, 3]
>>> print(x[2])    #输出显示列表指定元素
3
```

# Python变量

- 字符串和元组属于不可变序列，不能通过下标的方式来修改其中的元素值，试图修改元组中元素的值时会抛出异常。

```
>>> x = (1, 2, 3)
>>> print(x)
(1, 2, 3)
```

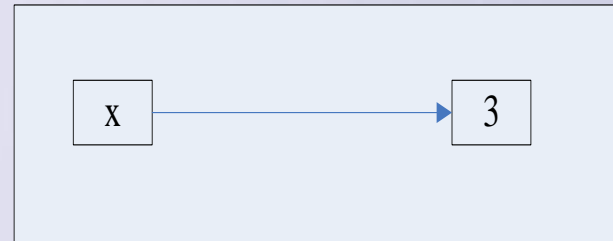
```
>>> x[1] = 5
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x[1] = 5
TypeError: 'tuple' object does not support item assignment
```



# Python变量

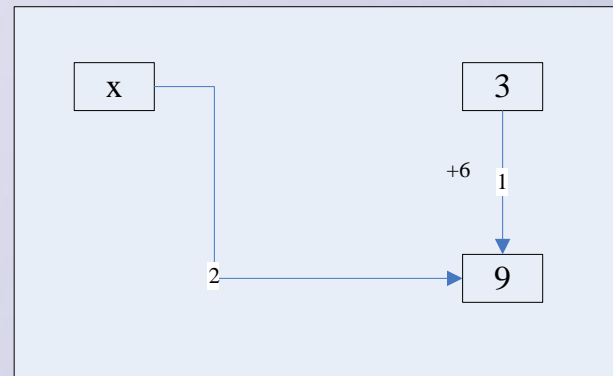
- 在Python中，允许多个变量指向同一个值，例如：

```
>>> x = 3
>>> id(x)
1786684560
>>> y = x
>>> id(y)
1786684560
```



- 然而，当为其中一个变量修改值以后，其内存地址将会变化，但这并不影响另一个变量，例如接着上面的代码再继续执行下面的代码：

```
>>> x += 6
>>> id(x)
1786684752
>>> y
3
>>> id(y)
1786684560
```



# Python变量

- Python采用的是基于值的内存管理方式，如果为不同变量赋值为相同值，这个值在内存中只有一份，多个变量指向同一块内存地址。

```
>>> x = 3
>>> id(x)
10417624
>>> y = 3
>>> id(y)
10417624
>>> x = [1, 1, 1, 1]
>>> id(x[0]) == id(x[1])
True
```

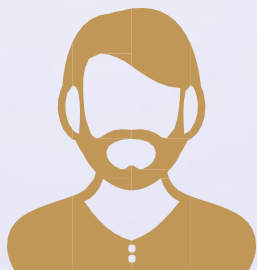
# Python变量

- ❖ Python具有自动内存管理功能，对于没有任何变量指向的值，Python自动将其删除。Python会跟踪所有的值，并自动删除不再有变量指向的值。因此，Python程序员一般情况下不需要太多考虑内存管理的问题。
- ❖ 尽管如此，显式使用del命令删除不需要的值或显式关闭不再需要访问的资源，仍是一个好的习惯，同时也是一个优秀程序员的基本素养之一。

# Python变量

- 在定义变量名的时候，需要注意以下问题：
  - ✓ 变量名**必须**以字母或下划线开头，但以下划线开头的变量在Python中有特殊含义；
  - ✓ 变量名中**不能**有空格以及标点符号（括号、引号、逗号、斜线、反斜线、冒号、句号、问号等等）；
  - ✓ **不能**使用关键字作变量名，可以导入keyword模块后使用`print(keyword.kwlist)`查看所有Python关键字；
  - ✓ **不建议**使用系统内置的模块名、类型名或函数名以及已导入的模块名及其成员名作变量名，这将会改变其类型和含义，可以通过`dir(__builtins__)`查看所有内置模块、类型和函数；
  - ✓ 变量名对英文字母的大小写敏感，例如student和Student是不同的变量。

# 数字类型及操作



- **整数类型**
- **浮点数类型**
- **复数类型**
- **数值运算操作符**
- **数值运算函数**



# 数字

❖ 数字是不可变对象，可以表示任意大小的数字。

```
>>> a=999999999999999999999999999999999999
```

$$\ggg a^*a$$
[illegible]

```
>>> a**3
```

[illegible]

❖ Python的IDEL交互界面可以当做简便计算器来使用。

```
>>> ((3**2) + (4**2)) ** 0.5
```

## 5.0

# 整数类型

## 与数学中整数的概念一致

—可正可负，没有取值范围限制

—`pow(x,y)`函数：计算  $x^y$ ，想算多大算多大

```
>>> pow(2,100)
```

```
1267650600228229401496703205376
```

```
>>> pow(2,pow(2,15))
```

```
1415461031044954789001553.....
```

# 数字

❖ Python中的整数类型可以分为：

- 十进制整数如，0、-1、9、123
- 十六进制整数，需要16个数字0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f来表示整数，必须以0x或0X开头，如0x10、0xfa、0xabcdef
- 八进制整数，只需要8个数字0、1、2、3、4、5、6、7来表示整数，必须以0o或0O开头，如0o35、0o11
- 二进制整数，只需要2个数字0、1来表示整数，必须以0b或0B开头如，0b101、0b100

- 整数无限制 `pow()`

- 4种进制表示形式



# 数字

## ❖ 浮点数

**与数学中实数的概念一致**

**— 带有小数点及小数的数字**

**— 浮点数取值范围和小数精度都存在限制，但常规计算可忽略**

**— 取值范围数量级约 $-10^{307}$ 至 $10^{308}$ ，精度数量级 $10^{-16}$**

# 浮点数类型

浮点数间运算存在不确定尾数，不是bug

```
>>> 0.1 + 0.3
```

```
0.4
```

```
>>> 0.1 + 0.2
```

```
0.30000000000000004
```

---

不确定尾数

# 浮点数类型

## 浮点数间运算存在不确定尾数

```
>>> 0.1 + 0.2 == 0.3
```

```
False
```

```
>>> round(0.1+0.2, 1) ==  
0.3
```

```
True
```

- `round(x, d)`: 对x四舍五入, d是小数截取位数
- 浮点数间运算与比较用`round()`函数辅助
- 不确定尾数一般发生在 $10^{-16}$ 左右, `round()`十分有效

# 浮点数类型

浮点数可以采用科学计数法表示

- 使用字母e或E作为幂的符号，以10为基数，格式如下：  
：

**<a>e<b>** 表示  $a \cdot 10^b$

- 例如：**4.3e-3** 值为0.0043 **9.6E5** 值为960000.0

# 复数类型

与数学中复数的概念一致

如果 $x^2 = -1$ ，那么 $x$ 的值什么？

— 定义  $j = \sqrt{-1}$ ，以此为基础，构建数学体系

—  $a + bj$  被称为复数，其中， $a$ 是实部， $b$ 是虚部

# 数字

- Python内置支持复数类型。

```
>>> a = 3+4j
>>> b = 5+6j
>>> c = a+b
>>> c
(8+10j)
>>> c.real          #查看复数实部
8.0
>>> c.imag          #查看复数虚部
10.0
>>> a.conjugate()   #返回共轭复数
(3-4j)
>>> a*b              #复数乘法
(-9+38j)
>>> a/b              #复数除法
(0.6393442622950819+0.03278688524590165j)
```

# 运算符和表达式

## Python运算符与功能

运算符	功能说明
+	算术加法，列表、元组、字符串合并与连接，正号
-	算术减法，集合差集，相反数
*	算术乘法，序列重复
/	真除法
//	求整商，但如果操作数中有实数的话，结果为实数形式的整数
%	求余数，字符串格式化
**	幂运算
<、<=、>、>=、==、!=	（值）大小比较，集合的包含关系比较
or	逻辑或
and	逻辑与
not	逻辑非
in	成员测试
is	对象同一性测试，即测试是否为同一个对象或内存地址是否相同
、^、&、<<、>>、~	位或、位异或、位与、左移位、右移位、位求反
&、 、^	集合交集、并集、对称差集
@	矩阵相乘运算符

# Python位运算符

```
>>> a=12
>>> b=36
>>> a&b
4
>>> a|b
44
>>> a^b
40
>>> ~12
-13
>>> 12<<2
48
>>> 12>>2
3
```

a=12  
b=36

位与&运算规则: 0&0=0 0&1=0 1&0=0 1&1=1		12	00001100
	&	36	00100100
		4	00000100

位或 运算规则: 0 0=0 0 1=1 1 0=1 1 1=1		12	00001100
		36	00100100
		44	00101100

位异或^运算规则: 0^0=0 0^1=1 1^0=1 1^1=0		12	00001100
	^	36	00100100
		40	00101000

12 00001100

负数的位运算符都是在补码上进行

1. 先对正数求补码 00001100
2. 然后对补码取反, 包括符号位 11110011
3. 再求原码10001101即-13  
负数亦然

原码  
(符号位+真值的绝对值)

反码  
(正数为自身,  
负数除符号位均取反)

补码  
(正数为自身,  
负数为反码+1)

请区分取反 反码

-11

10001  
011

11110  
100

11110  
101



## 2.4 Python的编码规范

### 1、命名规则

- **Python**语言有一套自己的命名规则。命名规则并不是语法规定，只是一种习惯用法。

#### (1) 变量名、模块名、包名

- 通常采用小写，可使用下划线。
- 例如：

**rulemodule.py** #模块名，即文件名

**\_rule='rule information'** #\_rule变量名，通常前缀有一个下划线的变量名为全局变量

## (2) 类名、对象名

- 类首字母采用大写；
- 类中的方法名首字母小写，其后的每个单词的首字母大写；
- 对象名（类的实例）采用小写；
- 类外引用其属性和方法名时，以对象名作为前缀。
- 类的私有变量、私有方法以两个下划线作为前缀。

## 例：类及对象命名举例

```
class Student:                                #类名大写
    __name=""                                #私有实例变量前必须有两个下划线
    def __init__(self,name):
        self.__name=name                    #self相当于C++中的this
    def getName(self):                        #方法名首字母小写，其后每个单词的首字母大写
        return self.__name

if __name__=="__main__":                     #程序执行入口，相当于C++的main()函数
    student=Student("borphi")               #对象名小写
    print(student.getName())
```

导入模块时，\_\_name\_\_为模块的名字，运行脚本而不导入模块时，\_\_name\_\_为\_\_main\_\_。

# Python脚本的“\_\_name\_\_”属性

- 每个Python脚本在运行时都有一个“\_\_name\_\_”属性。如果脚本作为模块被导入，则其“\_\_name\_\_”属性的值被自动设置为模块名；如果脚本独立运行，则其“\_\_name\_\_”属性值被自动设置为“\_\_main\_\_”。例如，假设文件nametest.py中只包含下面一行代码：

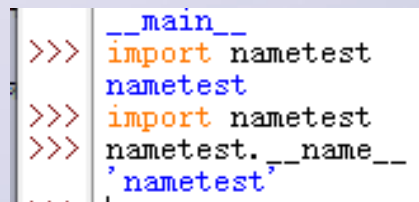
```
print(__name__)
```

- 在IDLE中直接运行该程序时，或者在命令行提示符环境中运行该程序文件时，运行结果如下：

```
__main__
```

- 而将该文件作为模块导入时得到如下执行结果：

```
>>> import nametest  
nametest
```



```
>>> import nametest  
nametest  
>>> import nametest  
>>> nametest.__name__  
'nametest'
```

# Python脚本的“\_\_name\_\_”属性

- 利用“\_\_name\_\_”属性即可控制Python程序的运行方式。例如，编写一个包含大量可被其他程序利用的函数的模块，而不希望该模块可以直接运行，则可以在程序文件中添加以下代码：

```
if __name__ == '__main__':  
    print('Please use me as a module.')
```

- 这样一来，程序直接执行时将会得到提示“Please use me as a module.”，而使用import语句将其作为模块导入后可以使用其中的类、方法、常量或其他成员。

# 作业1

- 从33个python的保留字中选择6个，并阐述其大致使用场景。