

Python 语言程序设计



第七章

自定义函数和模块



01

函数的定义

02

函数的调用

03

函数的参数传递

04

变量的作用域

05


函数的嵌套和递归

06

lambda函数

07

模块



01 函数的定义

函数的定义

- 函数是将可以被反复使用的、用来实现单一或相关联功能的代码段封装、组织在一起，一个独立的程序单位
- 在Python 中，定义函数的语法如下：

```
def <函数名> ([参数列表]) :  
    <函数体>  
[return <表达式列表>]
```

说明：

- (1) def：定义函数的关键字，后面接函数名、圆括号和冒号，函数声明以冒号结束。
- (2) 函数名：函数的名称，由用户定义的任何有效的标识符。
- (3) 函数体：在函数定义的缩进部分，描述函数的功能。函数体中的代码段在函数被调用时执行。
- (4) 参数列表：多个参数间用 “,” 分隔。参数列表中的参数被称为形式参数，简称“形参”。在调用函数时向函数传递值。
- (5) return：用于结束函数，return后面的值就是函数的返回值，将返回值传递给调用的语句。不带表达式的return返回值为None。

定义函数：def开头

#example7.1

函数名

def MyFun():

#定义函数MyFun()

print("这里是函数的开始")

print("函数被调用了")

print("这里是函数的结束")

函数体

函数体语句缩进

print("这里是主程序，调用函数的地方")

MyFun()

#调用函数MyFun()

print("这里是主程序的结束")

程序运行结果如下：

这里是主程序，调用函数的地方

这里是函数的开始

函数被调用了

这里是函数的结束

这里是主程序的结束

02 函数的调用

函数的调用

在Python 中，函数调用要在函数定义之后进行，具体格式如下：

用def语句定义的函数名

<函数名>(<参数列表>)

实参,有确定的值,可以由多个参数组成,中间用“,”分隔,可以是常量、变量或者表达式,传递给函数中的形参

【例7.2】设计绘制任意多边形的函数。

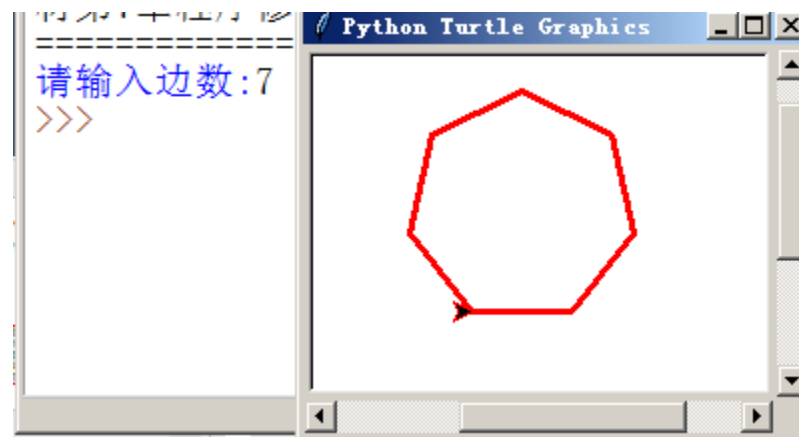
```
#example7.2
def fun(x):
    for i in range(x):
        fd(50)
        left(360/x)
from turtle import *
pensize(3)
pencolor("red")
a=eval(input("请输入边数:"))
fun(a)
```

形式参数

实际参数

调用函数:
通过函数名调用

程序结果如下:



03 函数的参数传递

7.3.1 参数传递

参数传递的方式

Python中主程序调用函数时，实参的值传递给形参，实际上是将实参所指向的对象的地址传递给了形参。

当传递的对象是不可变对象，如数值、字符、元组等，在函数体中形参值的变化不会影响到实参。

当传递的对象是可变对象，如列表、字典等，在函数中可变对象值的变化会影响到实参。

【例7.4】传递不可变对象，形参的变化不会影响到实参。

```
#example7.4
def add(x):
    print("形参x的初始值是：", x)
    x+=1
    print("形参x的最终值是：", x)
y=4
print("实参y的初始值是：", y)
add(y)
print("实参y的最终值是：", y)
```

程序运行结果如下：

实参y的初始值是： 4
形参x的初始值是： 4
形参x的最终值是： 5
实参y的最终值是： 4

形参的变化不会影响到实参

【例7.5】可变对象列表作为形参，形参的变化会影响到实参。

```
#example7.5
def change(n):
    n.append(3)
    print('函数中n值:', n)
m=[1]
print('调用函数前m的值:', m)
change(m)
print('调用函数后m的值:', m)
```

程序运行结果如下：

调用函数前m的值: [1]
函数中n值: [1, 3]
调用函数后m的值: [1, 3]

形参的变化会影响到实参

【例7.6】可变对象字典作为形参，形参的变化会影响到实参。

```
#example7.6
def change(x):
    x['院系']='法学院'
    x['专业']='法学'
a={'学号':'20024011','姓名':'杨晓霏','院系':'管理学院','专业':'行政管理'}
print(a)
change(a)
print(a)
```

程序运行结果如下：

```
{'专业': '行政管理', '学号': '20024011', '院系': '管理学院', '姓名': '杨晓霏'}
{'专业': '法学', '学号': '20024011', '院系': '法学院', '姓名': '杨晓霏'}
>>>
```

形参的变化会影响到实参

7.3.2 位置参数和关键字参数

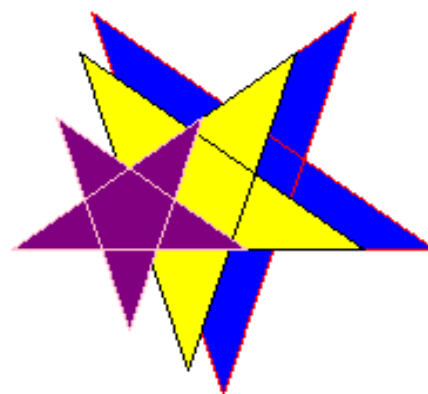
1. 位置参数

默认情况下，Python要求调用函数时参数的个数、位置和顺序要与函数定义中的一致，这种参数也被称为位置参数。

【例7.7】 位置参数的使用。

```
#example7.7
import turtle
def star(a,b,c):
    turtle.color(a,b)
    turtle.begin_fill()
    for i in range(5):
        turtle.forward(c)
        turtle.left(144)
    turtle.end_fill()
star("red","blue",180)
star("black","yellow",150)
star("pink","purple",100)
turtle.hideturtle()
```

程序运行结果如下：



▶ 【例7.8】修改【例7.7】中的函数调用语句，查看程序运行结果。

```
#example7.8-a
import turtle
def star(a,b,c):
    turtle.color(a,b)
    turtle.begin_fill()
    for i in range(5):
        turtle.forward(c)
        turtle.left(144)
    turtle.end_fill()
star("red","blue")
turtle.hideturtle()
```

```
#example7.8-b
import turtle
def star(a,b,c):
    turtle.color(a,b)
    turtle.begin_fill()
    for i in range(5):
        turtle.forward(c)
        turtle.left(144)
    turtle.end_fill()
star("red","blue",180,100)
turtle.hideturtle()
```

运行以上两个程序，都会产生错误

位置参数和关键字参数

2. 关键字参数

在调用函数的时候，可以明确指定参数值传递给哪个形参，这样的参数被称为关键字参数。使用了关键字参数，可以不考虑形参与实参的位置和顺序一一对应。

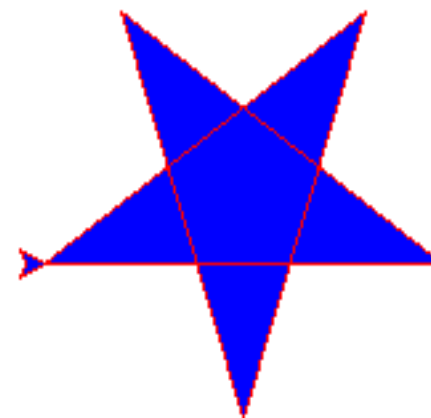
【例7.9】关键字参数

```
#example7.9
def star(a, b, c):
    import turtle
    turtle.color(a, b)
    turtle.begin_fill()
    for i in range(5):
        turtle.forward(c)
        turtle.left(144)
    turtle.end_fill()
star("red", "blue", 150)
star(b="blue", a="red", c=150)
star(c=150, a="red", b="blue")
```

#关键字参数
#关键字参数

参数位置不必一一对应

程序运行结果如下：



7.3.3 默认值参数

Python允许创建函数时为形参指定默认值。调用函数时，可以不为设置了默认值的形参传递值，此时将使用函数定义时形参的默认值，也可以通过显式赋值方式改变默认值。

带有默认值参数的函数按如下格式定义：

```
def <函数名>(...<形参=默认值>...)
    <函数体>
```

▶ 【例7.10】定义求 X^n 的函数，通过函数默认值，设定该函数在默认情况下求 X^2 。

```
#example7.10
def power(x, n = 2): #定义默认值参数n
    s = 1
    for i in range(1, n+1):
        s = s * x
    return s
print(power(5))      #形参n的默认值为2，求5的平方
print(power(6))      #形参n的默认值为2，求6的平方
print(power(5, 3))   #改变默认值参数n的值，求5的3次方
print(power(3, 4))   #改变默认值参数n的值，求3的4次方
```

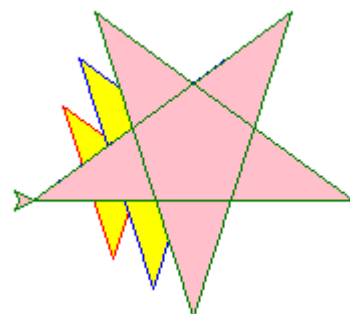
程序运行结果如下：

```
25
36
125
81
```

【例7.11】多个默认值参数的使用。

```
#example7.11
import turtle
def star(c, a='red', b='yellow'):
    turtle.color(a, b)
    turtle.begin_fill()
    for i in range(5):
        turtle.forward(c)
        turtle.left(144)
    turtle.end_fill()
star(80)
star(120, "blue")
star(160, "green", "pink")
```

程序运行结果如下：



7.3.4 可变参数

但在实际应用中，有时并不能事先确定函数个数，为了解决这一问题，在Python中，可以定义可变参数，不事先指定参数的数量，调用函数时，可变参数可以接收任意多个参数。

7.3.4 可变参数

1. 单星号参数——接收元组

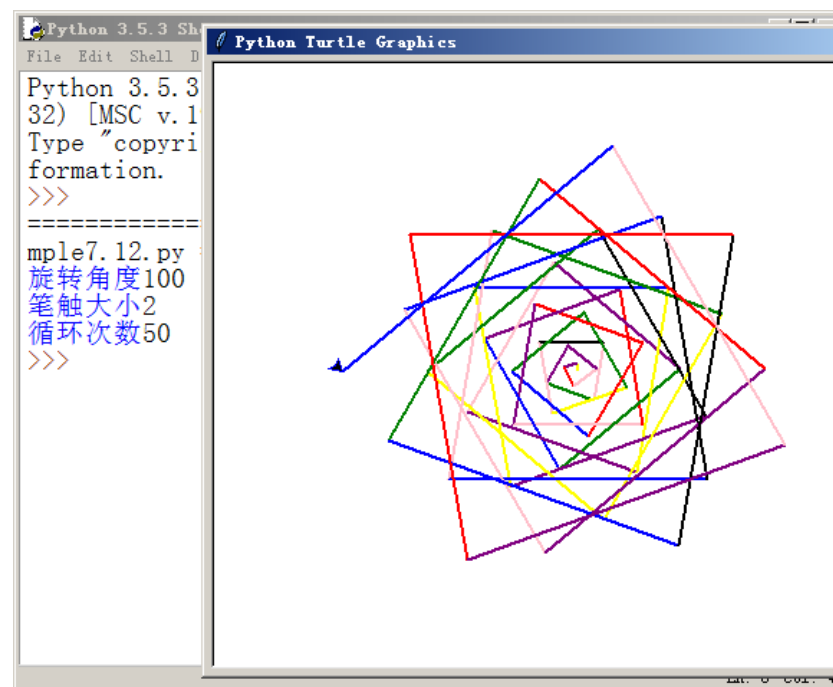
可以通过带星号 (*) 的参数定义，用来接收可变数量的参数，可变参数在函数调用时自动组装为一个元组。

(调用时可以接收任意多个参数)

【例7.12】可变参数调用。

```
#example7.12
def fun(x,*y):
    pensize(y[0])
    for i in range(y[1]):
        pencolor(choice(c))
        fd(i*5)
        left(a)
    end_fill()
from turtle import *
from random import choice,randint
c = ['yellow','green','purple','pink','red','black','blue']
a=eval(input('旋转角度'))
b=eval(input('笔触大小'))
d=eval(input('循环次数'))
fun(a,b,d)
```

程序运行结果如下：



7.3.4 可变参数

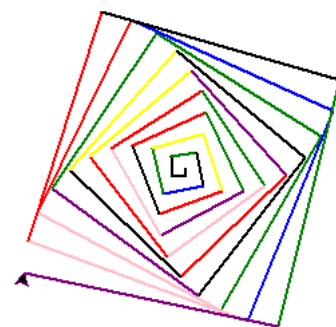
2. 双星号参数——接收字典

定义函数时，通过在参数前面添加两个星号（**），指定调用时关键字参数被放置在一个字典中传递给函数。

【例7.13】双星号参数。

```
#example7.13
def fun(x,**y):
    pensize(y["b"])
    for i in range(y["d"]):
        pencolor(choice(c))
        fd(i*5)
        left(x)
    end_fill()
from turtle import *
from random import choice,randint
c = ['yellow', 'green', 'purple', 'pink', 'red', 'black', 'blue']
a=randint(0,360)
fun(a,b=randint(1,5),d=randint(40,50))
```

程序运行结果如下：



【例7.14】形参为序列类型的可变参数传递。

```
#example7.14
def square_sum(*number):
    print(number)
    sum=0
    for i in number:
        sum = sum + i * i
    return sum
nums = [1, 2, 3]
print(square_sum(nums))
```

运行该程序，会产生如下错误：

```
([1, 2, 3],)
Traceback (most recent call last):
  File "C:\Users\Administrator\Desktop\第7章源程序+教材\example7.14.py", line 9,
in <module>
    print(square_sum(nums))
  File "C:\Users\Administrator\Desktop\第7章源程序+教材\example7.14.py", line 6,
in square_sum
    sum = sum + i * i
TypeError: can't multiply sequence by non-int of type 'list'
```

【例7.15】修改【例7.14】为正确的调用形式。

```
#example7.15
def square_sum(*number):
    print(number)
    sum=0
    for i in number:
        sum = sum + i * i
    return sum
nums = [1, 2, 3]
print(square_sum(nums[0], nums[1], nums[2]))
```

程序运行结果如下：

(1, 2, 3)
14

04 变量的作用域

7.4 变量的作用域

变量的作用域指的是变量的作用范围。根据变量作用域的不同，可以将变量分为**全局变量**和**局部变量**。

1. 全局变量

- (1) 在主程序中定义的变量是全局变量；
- (2) 用global声明的变量是全局变量，多个变量用逗号隔开；
- (3) 在函数体内只是引用了某个变量的值而没有为其赋新值，则该变量为全局变量。

7.4 变量的作用域

2. 局部变量

- (1) 函数的形式参数是局部变量;
- (2) 在函数体内部赋值号左侧出现的变量是局部变量, 作用域在该函数体内部。

【例7.16】局部变量的作用域。

```
#example7.16
def fun1(a, b):
    x=a                #fun1函数中的局部变量x
    y=b                #fun1函数中的局部变量y
    sum=x*0.3+y*0.7    #fun1函数中的计算sum
    fun2()              #在fun1函数中调用fun2函数
    print("在fun1中, 上机x=", x)    #在fun1函数中输出x
    print("在fun1中, 笔试y=", y)
    print("在fun1中, 总分sum=", sum)
def fun2():
    x=90                #在fun2函数中给局部变量x赋值
    y=80                #在fun2函数中给局部变量y赋值
    sum=(x+y)/2          #fun2函数中另一种计算分数方法
    print("在fun2中, 上机x=", x)    #在fun2函数中输出x
    print("在fun2中, 笔试y=", y)
    print("在fun2中, 总分sum=", sum)
    print("-----")
fun1(100, 85)           # 在主程序中调用fun1函数
|
```

程序运行结果如下:

```
在fun2中, 上机x= 90
在fun2中, 笔试y= 80
在fun2中, 总分sum= 85.0
-----
在fun1中, 上机x= 100
在fun1中, 笔试y= 85
在fun1中, 总分sum= 89.5
>>> |
```

【例7.17】全局变量的作用域。

```
#example7.17
def fun1(b):
    y=b
    sum=a*0.3+y*0.7
    fun2()
    print("在fun1中, 上机=", a)
    print("在fun1中, 笔试y=", y)
    print("在fun1中, 总分sum=", sum)
def fun2():
    y=80
    sum=(a+y)/2
    print("在fun2中, 上机=", a)
    print("在fun2中, 笔试y=", y)
    print("在fun2中, 总分sum=", sum)
    print("-----")
a=100
fun1(85)      # 在主程序中调用fun1函数
```

#fun1函数中的局部变量y
#fun1函数中的计算sum, a为全局变量
#在fun1函数中调用fun2函数
#在fun2函数中给局部变量y赋值
#fun2函数中a为全局变量

程序运行结果如下:

```
在fun2中, 上机= 100
在fun2中, 笔试y= 80
在fun2中, 总分sum= 90.0
-----
在fun1中, 上机= 100
在fun1中, 笔试y= 85
在fun1中, 总分sum= 89.5
>>> |
```

【例7.18】全局变量和局部变量同名。

```
#example7.18
def fun1(b):
    a=90
    y=b
    sum=a*0.3+y*0.7
    fun2()
    print("在fun1中, 上机=", a)
    print("在fun1中, 笔试y=", y)
    print("在fun1中, 总分sum=", sum)
def fun2():
    y=80
    sum=(a+y)/2
    print("在fun2中, 上机=", a)
    print("在fun2中, 笔试y=", y)
    print("在fun2中, 总分sum=", sum)
    print("-----")
a=100
fun1(85)      # 在主程序中调用fun1函数
```

#fun1函数中的局部变量y
#fun1函数中的计算sum, a为局部变量
#在fun1函数中调用fun2函数

#在fun2函数中给局部变量y赋值
#fun2函数中a为全局变量

程序运行结果如下:

```
在fun2中, 上机= 100
在fun2中, 笔试y= 80
在fun2中, 总分sum= 90.0
-----
在fun1中, 上机= 90
在fun1中, 笔试y= 85
在fun1中, 总分sum= 86.5
```

【例7.19】 global语句应用

```
#example 7.19
```

```
def fun1(b):
```

```
    global a
```

```
    a=100
```

```
    y=b
```

```
    sum=a*0.3+y*0.7
```

```
    print("在fun1中, 上机=", a)
```

```
    print("在fun1中, 笔试y=", y)
```

```
    print("在fun1中, 总分sum=", sum)
```

#fun1函数中的局部变量y
#fun1函数中的计算sum, a为局部变量

```
a=90
```

```
fun1(85) # 在主程序中调用fun1函数
```

```
print("a=", a)
```

程序运行结果如下:

```
在fun1中, 上机= 100
在fun1中, 笔试y= 85
在fun1中, 总分sum= 89.5
a= 100
```

05 函数的嵌套和递归

7.5.1 函数的嵌套定义

Python支持嵌套函数，即在函数定义的时候，函数体内部又包含另外一个函数的完整定义，并且可以多层嵌套。

【例7.19】 嵌套函数实例1。

```
#example7.19
def First():
    a = 3
    def Second():
        b = 4
        print(a + b)
    Second()
    print(a)
First()
```

程序运行结果如下：

7
3

【例7.20】 嵌套函数实例2。

```
#example7.20
def First():      #定义函数First()
    a = 3
    def Second(): #在函数First()内部定义函数Second()
        a = 5
        b = 4
        print(a + b)
    Second()      #在First()函数内调用Second()函数
    print(a)
First()
```

程序运行结果如下：

9
3

【例7.21】函数多层嵌套实例。

```
#example7.21
def first():
    x1 = "Dream1"
    print(x1)
    def second():
        x1 = "second_Dream1"
        global x2
        x2 = "Dream2"
        print(x1, x2)
        def third():
            x3 = "Dream3"
            print(x1, x2, x3)
        return third()
    second()
    print(x1, x2)
first()
```

程序运行结果如下：

```
Dream1
second_Dream1 Dream2
second_Dream1 Dream2 Dream3
Dream1 Dream2
...
```

7.5.2 递归

在函数内部调用函数自身，这种函数的调用方式被称为递归。

【例7.22】利用递归函数求斐波拉契数列前n项。

```
#example7.22
def Fib(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return Fib(n-1)+Fib(n-2)
print("输出斐波拉契数列前8项: ", end="")
for i in range(8):
    print(Fib(i), end=" ")

print()
print("输出斐波拉契数列第15项: ", Fib(14))
```

运行程序，输出斐波拉契数列8项和前15项的结果如下：

输出斐波拉契数列前8项: 0 1 1 2 3 5 8 13
输出斐波拉契数列第15项: 377

06 lambda函数

lambda函数

lambda函数语法格式如下：

<函数名>=lambda <参数列表>:<表达式>

lambda函数相当于下面的普通函数定义格式：

```
def <函数名>(参数列表):  
    return <表达式>
```

lambda函数

```
>>> fun=lambda x, y, z :min(x, y, z)
>>> fun(12, 23, 56)
12
```

lambda 函数等价于使用def关键字定义的函数:

```
>>> def fun(x, y, z):
    fun=min(x, y, z)
    return fun

>>> fun(10, 20, 0)
0
```

lambda 函数支持默认值参数和关键字参数.

```
>>> fun=lambda x, y=3, z=15 :min(x, y, z)  #默认值参数
>>> fun(30)
3
>>> fun(y=10, z=0, x=30)  #关键字参数
0
```

lambda 函数可以作为列表或字典元素，以列表为例：

```
>>> list1=[22, 68, -31, 96]
>>> list2=sorted(list1, key=lambda x:-x)  #列表元素降序排序
>>> list2
[96, 68, 22, -31]
>>> list2=sorted(list1, key=lambda x:x)  #列表元素升序排序
>>> list2
[-31, 22, 68, 96]
```

07 模块

7.7.1 模块的导入

1. import语句导入

使用import 语句导入的基本语法格式如下：

import 模块名

import 模块名1,模块名2...

import 模块名 as 模块别名

2. from...import 导入

使用 from...import 导入模块的基本语法格式如下：

from 模块名 import <函数名>

from 模块名 import <函数名> as <别名>

from 模块名 import *

7.7.2模块的搜索路径

使用标准模块sys中的path变量查看当前搜索路径

```
>>> import sys
>>> sys.path
['', 'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35\\Lib\\idlelib',
'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35\\python35.zip',
'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35\\DLLs',
'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35\\lib',
'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35',
'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35\\lib\\site-packages']
>>>
```



通过调用列表的append()方法增加目录

```
>>> import sys
```

```
>>> sys.path.append("C:\\Users\\Administrator\\Desktop")
```



7.7.3自定义模块和包

1.自定义模块

用户自定义模块就是建立一个python程序文件，文件的名字就是模块的名字。

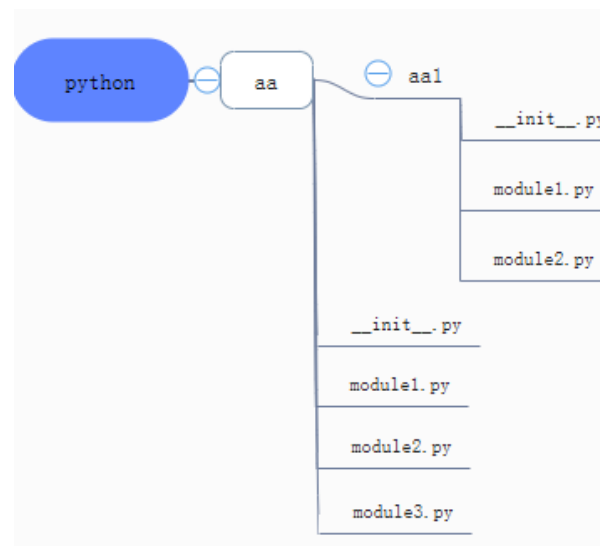
```
def triangle(n):  
    for i in range(1,n+1):  
        print(" "*(n-i)+"*"* (2*i-1))
```

```
>>> import aa  
>>> aa.triangle(5)  
*  
***  
*****  
*****  
*****  
>>>
```

7.7.3自定义模块和包

2.包

当一个项目中有很多个模块时，需要再进行组织。Python将功能类似的模块放到一起，称为包（Package）。



7.7.3 安装第三方模块

1.第三方模块介绍

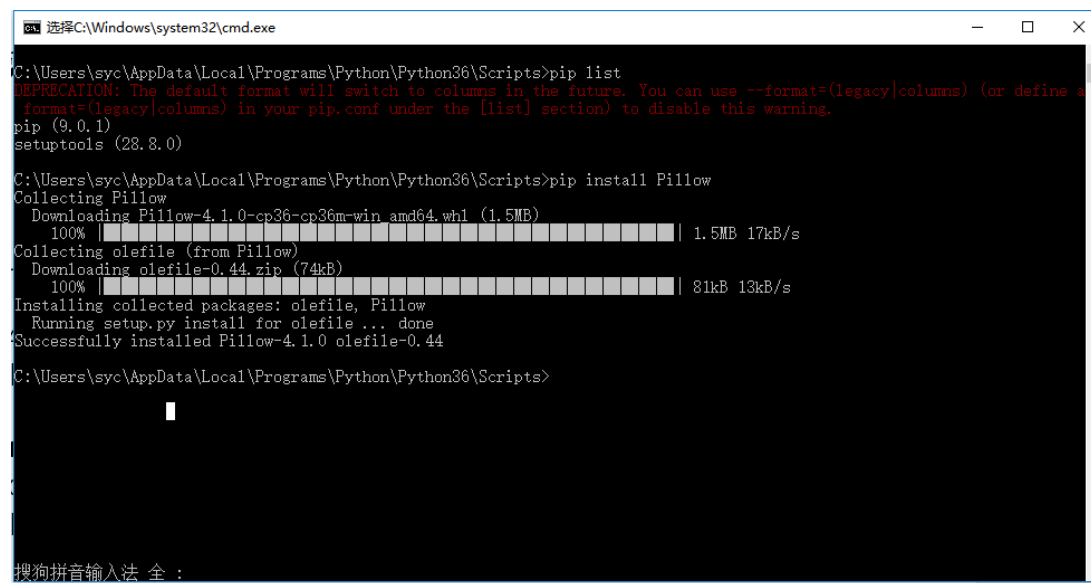
表7.1 常用pip命令使用方法

pip命令示例	说明
pip help	列出pip系列的子命令
pip list	列出当前已安装的所有模块
pip install	安装模块
pip uninstall	卸载模块
pip install-upgrade	升级模块
pip download	下载模块
pip show	显示模块信息
pip search	查找模块

2. 第三方模块Pillow安装与卸载

以安装第三方模块Pillow为例，介绍利用pip命令安装第三方模块的过程

`: \> pip install Pillow`



```
C:\Users\syc\AppData\Local\Programs\Python\Python36\Scripts>pip list
DEPRECATION: The default format will switch to columns in the future. You can use --format=(legacy|columns) (or define a
format=(legacy|columns) in your pip.conf under the [list] section) to disable this warning.
pip (9.0.1)
setuptools (28.8.0)

C:\Users\syc\AppData\Local\Programs\Python\Python36\Scripts>pip install Pillow
Collecting Pillow
  Downloading Pillow-4.1.0-cp36-cp36m-win_amd64.whl (1.5MB)
    100% |#####| 1.5MB 17kB/s
Collecting olefile (from Pillow)
  Downloading olefile-0.44.zip (74kB)
    100% |#####| 81kB 13kB/s
Installing collected packages: olefile, Pillow
  Running setup.py install for olefile ... done
Successfully installed Pillow-4.1.0 olefile-0.44

C:\Users\syc\AppData\Local\Programs\Python\Python36\Scripts>
```

2. 第三方模块Pillow卸载

`: \> pip uninstall Pillow`

3.更新pip版本

更新pip版本命令如下:

```
pip install --upgrade pip
```

7.7.4 常见模块应用实例

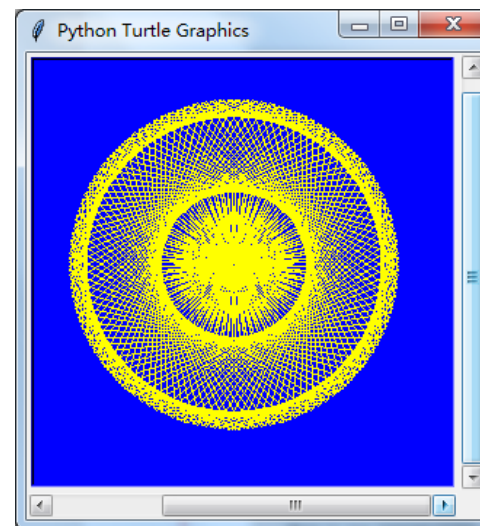
1. 图形绘制模块turtle

Python内置的标准模块turtle可以非常方便地进行图形绘制，turtle绘图模块通常称为海龟绘图。

【例7.24】利用turtle模块绘制多边形。

```
#example7.24
from turtle import *
bgcolor("blue")
color("yellow")
speed(0)
for i in range(120): #这里设定正方形的个数
    for j in range(5):
        forward(100)
        right(90)
    right(93) #旋转角度
```

程序运行结果：

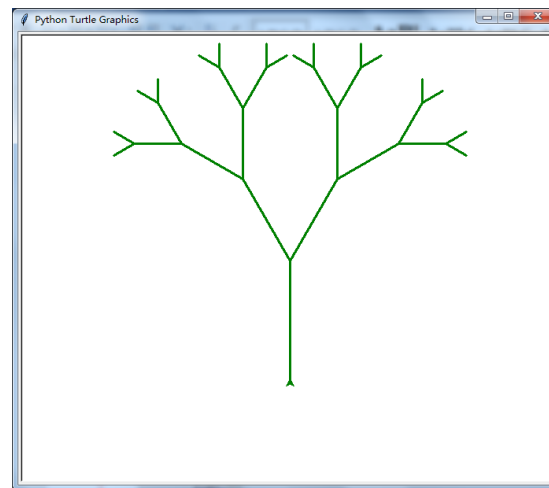


【例7.25】利用turtle模块绘制圆形及曲线。

```
#example7.25
from turtle import *
def tree(length):
    if length > 10:
        forward(length)
        right(30)      #向右旋转30度
        tree(length-30) #调用函数tree
        left(60)       #向左旋转60度
        tree(length-30) #调用函数tree
        right(30)      #向右旋转30度
        backward(length) #原路返回长度length

pensize(3)
color('green')
speed(1)
left(90) #向左旋转90度
backward(150) #向下绘制150
tree(150) #调用函数tree
```

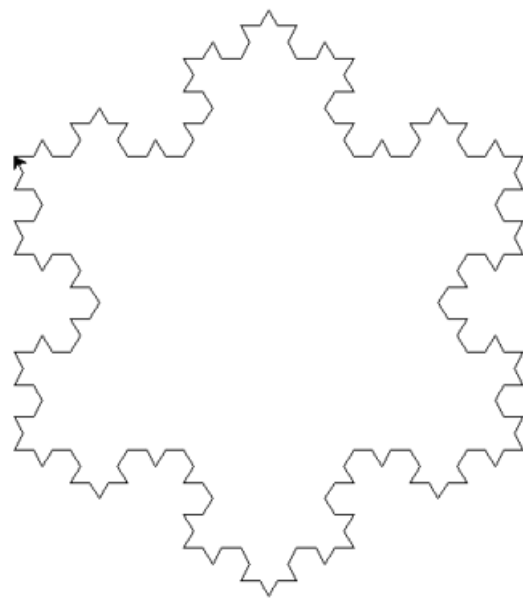
程序运行结果：



【例7.26】利用turtle模块绘制科赫曲线。

```
#example7.26
from turtle import *
def koch(n, k):      #n表示科赫曲线的阶数，k表示科赫曲线每段的长度
    if n == 0:       #n=0时，科赫曲线是一条直线
        fd(k)
    else:
        for angle in (60, -120, 60, 0):
            koch(n-1, k/3)  #n=1时，科赫曲线在中间1/3位置画一个边长为k/3等边三角形
            left(angle)
up()
goto(-200, 100)
down()
koch(3, 400)        #n=3时的科赫曲线，改变n的值，画出不同程度的科赫曲线
right(120)
koch(3, 400)
right(120)
koch(3, 400)
```

程序运行结果如图所示：



2.文件打包模块pyinstaller

安装 PyInstaller 模块输入如下命令：

```
pip install pyinstaller
```

表7.4 pyinstaller的常用参数

函数名	功能
-h、--help	显示帮助信息
-v、--version	查看版本号
-distpath	生成文件放在哪里，默认：当前目录的dist文件夹内
-y	如果dist文件夹内已经存在生成文件，则不询问用户，直接覆盖，默认：询问是否覆盖
--clean	在本次编译开始时，清空上一次编译生成的各种文件，默认：不清除
-D、--onedir	生成dist目录
-F、--onefile	在dist文件夹中只生成独立的打包文件
-p DIR、--paths DIR	添加python文件使用的第三方模块路径，DIR是第三方模块路径

3.图像处理模块pillow

pillow (PILP : ython Imaging Library) 是 Python 中最常用的图像处
理库, pillow是第三方模块, 先安装后再使用, 安装pillow的命令如下:

```
pip install pillow
```

(1) 打开图像

使用函数 `open()` 语法格式如下所示：

`open(fp,mode)`

说明：

fp：打开文件的路径。

mode：可选参数，表示打开文件的方式

表7.5 Pillow 库支持的常用图片模式信息

mode (模式)	功能
1	1位像素, 黑和白, 存成8位像素
L	8位像素, 黑白
P	8位像素, 使用调试板映射到任何其他模式
RGB	3*8位像素, 真彩
RGBA	4*8位像素, 真彩+透明通道
CMYK	4*8位像素, 颜色隔离
YCbCr	3*8位像素, 彩色视频格式
LAB	3*8位像素, lab颜色空间
I	32位整型像素
F	32位浮点像素

【例7.27】打开图像，并查看显示相关信息。

```
#example7.27
from PIL import Image
im = Image.open('f:\\photo1.png')    #打开f:\\photo1.png
print(' 图像格式:', im.format)
print(' 图像大小 (宽度, 高度):', im.size)
print(' 图像宽度:', im.width, ' 图像高度:', im.height)
im.show()    #显示图片
```

图像格式: JPEG

图像大小 (宽度, 高度) : (500, 333)

图像宽度: 500 图像高度: 333



(2) 新建图像

新建图像的语法格式如下：

`new(mode,size,color=0)`

说明：

mode: 图片模式，具体取值如表7.5所示.

size: 表示图片尺寸，是使用宽和高两个元素构成的元组

color: 默认颜色（黑色）

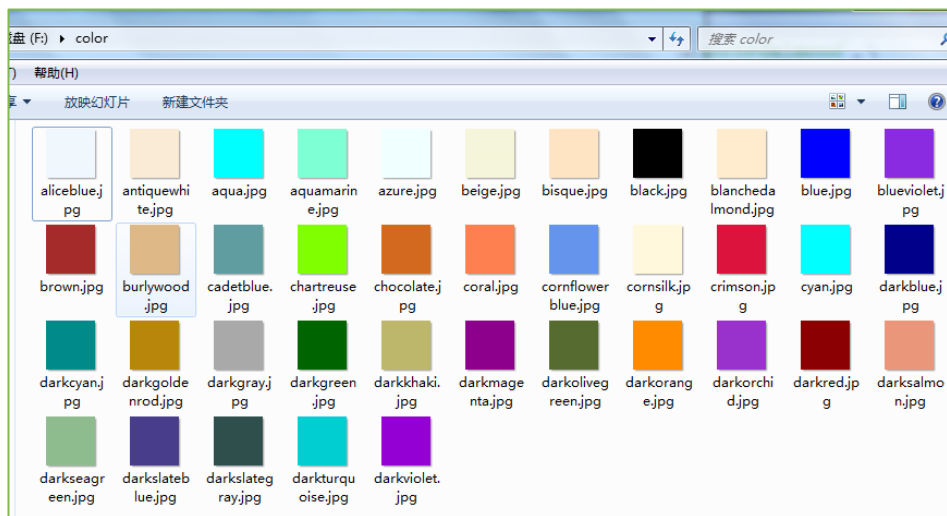
(3) 保存图像

【例7.28】在f:\color文件夹下建立以颜色为文件名，大小为100*100的图像文件

example7.28

```
c=['aliceblue','antiquewhite','aqua','aquamarine','azure','beige','bisque','black','blanchedalmond','\
'blue','blueviolet','brown','burlywood','cadetblue','chartreuse','chocolate','coral','\
'cornflowerblue','cornsilk','crimson','cyan','darkblue','darkcyan','darkgoldenrod','darkgray','\
'darkgreen','darkkhaki','darkmagenta','darkolivegreen','darkorange','darkorchid','darkred','\
'darksalmon','darkseagreen','darkslateblue','darkslategray','darkturquoise','darkviolet']
from PIL import Image
for i in c:
    s=Image.new("RGB", (100, 100), i)
    s.save("f:\\color\\%s.jpg"%i)
```

运行结果如下：



(4) 旋转图像

Image 模块中，函数 rotate() 的功能返回此图像的副本，围绕其中心逆时针旋转给定的度数。具体语法格式如下：

```
Image.rotate (angle, resample = 0, expand = 0, center =  
None, translate = None, fillcolor = None )
```

说明：

angle：逆时针方向。

resample：可选的重采样过滤器。

expand：可选的扩展标志。

center：可选的旋转中心。

translate：可选的后旋转。

fillcolor：旋转像外部区域的可选颜色

```
>>> im=Image.open("f:\\photo1.jpg")
>>> im.rotate(90).show()  #逆时针选择90度并显示
>>>
```



(5) 透明度混合处理图像

函数blend()的语法格式如下：

blend(im1,im2,alpha)

说明：

im1、im2：参与混合的图像1和图像2

alpha：混合的透明度，取值在0-1，具体混合过程为：
 $(im1 * (1 - alpha) + im2 * alpha)$ 。当混合透明度为 0 时，显示 im1
原图。当混合透明度 alpha 取值为 1 时，显示 im2原图片。

【例7.29】将f:\photo1.jpg设置透明度混合

```
#example7.29
from PIL import Image
im1=Image.open("f:\\photo1.jpg").convert(mode="RGB")
im2=Image.new("RGB", im1.size, "blue")
Image.blend(im1, im2, alpha=0.5).show()
```

运行结果如下：



(6) 遮罩混合处理图像

在 Image 模块中使用函数 `composite()` 实现遮罩混合处理。
`composite()` 函数的语法格式如下：

`composite(im1,im2,mask)`

说明：

`im1` 和 `im2` ： 混合处理的图片 1 和图片 2

`mask`： 图像模式，大小要和 `im1`、`im2` 一样。

【例7.30】图像遮罩混合处理效果。

```
#example7.30
from PIL import Image
im1=Image.open('f:\\photo1.jpg')
im2=Image.open('f:\\photo2.jpg')
im2=im2.resize(im1.size)
r, g, b=im2.split()
Image.composite(im1, im2, b).show()
```



【例7.31】将图片切割成九宫图

```
#example7.31
from PIL import Image
# 将图片切割成九宫格
def cut(image):
    width, height = image.size #选取图片的宽度和高度
    item_width = int(width / 3) #一行放3张图
    item_height = int(height / 3) #一列放3张图
    item_list = []
    for i in range(0, 3):
        for j in range(0, 3):
            item = (j*item_width, i*item_height, (j+1)*item_width, (i+1)*item_height)
            item_list.append(item)
    image_list = [image.crop(item) for item in item_list]
    return image_list
#保存图片
def save(image_list):
    index = 1
    for image in image_list:
        image.save(str(index) + '.png', 'PNG')
        index += 1

if __name__ == '__main__':
    image = Image.open("f:\\1.jpeg") #打开d盘下的图片文件1.jpeg
    image_list = cut(image)
    save(image_list)
```



THANK YOU

The user can demonstrate on a projector or computer, or print the presentation and make it into a film to be used in a wider field