Python 语言程序设计

陈 峦 副教授

13880209111，chluan@uestc.edu.cn

研究院大楼316#

第八章　函数与模块

- 函数（function）是执行特定任务的一段代码。
- 可将反复要用到的程序段封装成函数，并为其指定一个函数名，在需要的时候即可多次调用这段代码。
- 函数是代码复用的重要手段，可提高代码的重复利用率。
- 它也能提高程序的可读性（模块性：单入口，单出口，高内聚，低耦合）。

函数的分类：

1. 系统函数：Python系统提供的函数。

- （1）Python的内置函数：常驻内存，可直接使用，如print()函数；
- （2）标准模块库中的函数：常驻磁盘，需要import，如math模块中的sqrt()函数；
- （3）对象的方法：定义在类中的函数称为方法，它也是一种函数。

2. 用户自定义函数：在Python程序中，用户自己创建的函数。

Python的内置函数

- Python内置函数包含在模块__builtins__中，该模块在启动Python解释器时自动装入内存
- 例：查看Python的全部内置函数

```
>>> dir(__builtins__)
>>> help(id)
>>> help(type)
>>> help(print)
```

标准模块库中的函数

- >>>help()
  - help>modules
  - help>math
  - help>quit
- >>> import math
- >>> dir(math)
- >>> help(math.sqrt)

用户自定义函数

- （1）定义函数
- def sum(x,y):
- 	return x+y
- def f():
- 	print("Goodbye!")
- 	return
- （2）调用函数

- 模块（module）是Python最高级别的程序组织单元，比函数粒度更大，一个模块可以包含若干个函数。
- 模块也分系统模块和用户自定义模块。
- 用户自定义模块就是一个“.py”程序文件。
- 在导入模块之后才可以使用模块中定义的函数。
- 例：要调用sqrt()函数，就必须用import语句导入math模块。

8.1 函数的定义与调用

- 用户自定义函数要先定义，然后才能调用。

8.1.1 函数的定义

- 函数定义包括对函数名、函数的参数与函数功能的描述。
- 一般形式为：
- def 函数名([形式参数表]):
- 　函数体

- 交互式命令窗口：
- **>>> def f(x,y):**
- **　　return x+y**
- **>>> f(2,3)**
- **5**
- 程序代码窗口：**a.py**
- **def f(x,y):**
- **　return x+y**
- **print(f(2,3))**

1. 函数首部

- 即函数的第一行，用于对函数的特征进行定义。
- 函数名：是一个合法的标识符，尽量要见名知义，能反映函数功能，便于记忆。
- 形式参数（formal parameter）：简称形参。形参之间用逗号分隔。

2. 函数体

- 即缩进部分，它描述了函数的功能。
- 格式：return 表达式
- 一个函数中可以有多个return语句。
- 不带参数的return语句或函数体内没有return语句，则函数返回空（None）。
- 若函数需要返回多个值，可以把这些值当成一个元组返回。
- 例：“return 1,2,3 ” 返回元组(1,2,3)。

- 例：
- def f(x):
- 　　return x,x**2,x**3
- def main():
- 　　a,b,c=f(2)
- 　　print("x={},x^2={},x^3={}".format(a,b,c))
- main()

3. 空函数

- 即函数体为空的函数。
- 在编程时常用于“占位置”。
- 格式为:
- def 函数名():
- pass
- 调用空函数时，将什么工作也不做。

4. 为函数提供帮助说明文档

例：a.py

```
import math
def f(z):
    '''
    求一个复数模长的函数
    f(z)
        返回复数z的模
    '''
    return math.sqrt(z.real*z.real+z.imag*z.imag)
def main():
    print(f(3+4j))
main()
```

```
##########################################
>>> import a.py
>>> help(a.f)
>>> print(a.f.__doc__)
```

8.1.2 函数的调用

- 函数的调用格式：函数名(实际参数表)
- 实际参数（actual parameter），简称实参。
- 当有多个实际参数时，实际参数之间用逗号分隔。
- 当没有实参时，也不能省略括号。
- 实参与形参按位置顺序一一对应，它们的参数类型要兼容。

- **Python**函数可以在交互式命令提示符下定义和调用。
- 但通常的做法：是将函数定义和函数调用都放在一个程序文件（***.py**）中，然后运行程序文件。

- 惯例（语法上没有要求）：
- 在Python程序中通常应定义一个主函数，用于完成程序的总体调度功能。
- 由主函数来调用其他函数，使得程序呈现模块化结构。
- 程序的主函数（程序入口）一般命名为main。
- 程序最后一行是调用主函数的语句，它是整个程序的入口。

- 例：
- def f(x,y):
- 　return x+y
- def g(m,n):
- 　return m*n
- def main():
- 　print(f(2,3))
- 　print(g(4,5))
- 　return
- main()

5
20

- def f(x,y):
- 　return x+y
- def g(m,n):
- 　return m*n
- def main():
- 　a=f(2,3)
- 　b=g(4,5)
- 　print(a,b)
- 　return
- main()

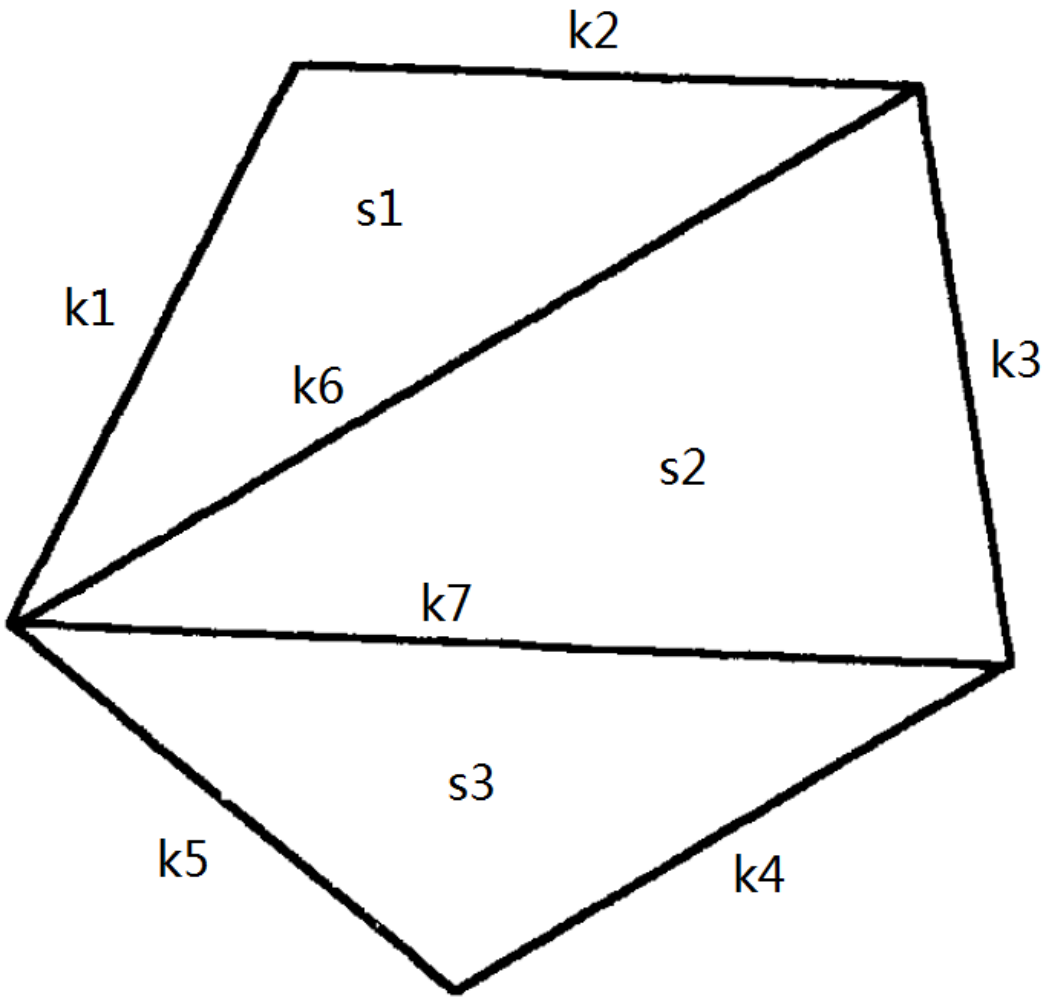
5 20

- 例：
- def f():
- print("AA")
- return
- def g():
- print("BB")
- return
- def main():
- f()
- g()
- return
- main()

- 例：
- def f():
- print("AA")
- def g():
- print("BB")
- def main():
- f()
- g()
- main()

AA
BB

- 例：求五边形的面积，其各边长度从键盘输入。



```
from math import *
def ts(a,b,c):
    s=(a+b+c)/2
    s=sqrt(s*(s-a)*(s-b)*(s-c))
    return s
def main():
    k1,k2,k3,k4,k5,k6,k7=eval(input())
    s=ts(k1,k2,k6)+ts(k6,k3,k7)+ts(k7,k4,k5)
    print("area=",s)
main()
```

8.2 函数的参数传递

- 形参是函数定义时由用户定义的形式上的变量。
- 实参是函数调用时，主调函数为被调用函数提供的原始数据。
- 例：

```
def f(x,y):
    return x*x+y*y
def main():
    print(f(2,3))
main()
```

8.2.1 参数传递方式

- 在Python中，各种数据都是一个对象，通过id()函数可以获得该对象的ID号（可以理解为数据所在内存单元的地址）。
- 在Python中，变量是一个对象的引用，变量与变量之间的赋值是对同一个对象的引用。
- 当给变量重新赋值时，则这个变量指向一个新分配的对象。

- 例：
- >>> x=2
- >>> y=2
- >>> id(x),id(y)
- (1740563232, 1740563232)
- >>> x=3
- >>> id(x),id(y)
- (1740563248, 1740563232)
- >>> x=2
- >>> id(x),id(y)
- (1740563232, 1740563232)

- 例：
- >>> x=2
- >>> y=x
- >>> id(x),id(y)
- (1740563232, 1740563232)
- >>> z=2
- >>> id(x),id(y),id(z)
- (1740563232, 1740563232, 1740563232)

- 例：
- >>> x=20
- >>> y=30
- >>> z=40
- >>> id(x),id(y),id(z)
- (1740563520, 1740563680, 1740563840)

- 在Python中，变量指向一个对象，或者一段内存空间，这段内存空间的内容是可以修改的，但变量的内存起始地址是不能改变的。
- 变量之间的赋值相当于两个变量指向同一块内存区域，在Python中就相当于同一个对象。

- 例：
- >>> s=[1,2,3]
- >>> id(s)
- 35787520
- >>> s[2]=4
- >>> s
- [1, 2, 4]
- >>> id(s)
- 35787520

- 例：
- >>> s=[1,2,3]
- >>> t=[1,2,3]
- >>> id(s),id(t)
- (35568816, 35568856)
- >>> t=s
- >>> id(s),id(t)
- (35568816, 35568816)
- >>> t[2]=4
- >>> s,t
- ([1, 2, 4], [1, 2, 4])

例：

```
>>> d={"aa":1,"bb":2}
>>> id(d)
34895168
>>> d["cc"]=3
>>> d
{'aa': 1, 'bb': 2, 'c': 'c', 'cc': 3}
>>> id(d)
34895168
>>> d.clear()
>>> d
{}
>>> id(d)
34895168
```

- 在Python中，实参向形参传送数据的方式是“值传递”，即“拷贝”或“复制”。
- 实参的值传给形参，是对象间整体赋值，是一种单向传递方式，不能由形参传回给实参。

- 例：
- def f(x):
- print("x1=",id(x))
- x=3
- print("x2=",id(x))
- def main():
- a=2
- print("a1=",id(a))
- f(a)
- print("a=",a,"a2=",id(a))
- main()

```
a1= 500425504
x1= 500425504
x2= 500425520
a= 2 a2= 500425504
```

- 当在函数内部修改列表、字典的元素时，形参的改变会影响实参，即双向传递，类似于“传地址”、“共享内存”、“借”。
- 即，在函数内没有对传递进来的形参变量重新赋值，而是修改可变形参变量（列表和字典）的局部元素，此时就会导致外部实参（所指向对象的内容）的修改。

- 例：
- def f(m,n):
- 　　m[2]=44
- 　　n["bb"]=3
- 　　return
- s=[11,22,33]
- t={"aa":1,"bb":2}
- f(s,t)
- print("s={},t={}".format(s,t))

s=[11, 22, 44], t={'aa': 1, 'bb': 3}

- 例：
- def f(t):
- t[0],t[1]=t[1],t[0]
- return
- s=list(eval(input("请输入：")))
- f(s)
- print("s={}".format(s))

```
请输入：11,22
s=[22, 11]
```

8.2.2 参数的类型

- 参数类型包括位置参数、关键字参数、默认值参数和可变长度参数。

1. 位置参数

- 函数调用时的参数通常采用按位置匹配的方式，即实参按顺序传递给相应位置的形参。
- 此时，实参的个数应与形参个数必须完全相等。

- 例：位置参数
- def f(m,n):
- 　　print(m,n)
- 　　return
- f(2,3)

2 3

- 例：
- def f(m,n):
- 　　print(m,n)
- 　　return
- f(4)

TypeError: f() missing 1 required positional argument: 'n'

2.关键字参数

- 关键字参数的形式为：
- 形参名=实参值
- 在函数调用中使用关键字参数，是指通过形式参数的名称来指示为哪个形参传递什么值，这可以跳过某些参数或脱离参数的顺序。

- 例：位置参数
- def f(y,x):
- 　　print("2:x={},y={}".format(x,y))
- 　　return
- x=2
- y=3
- print("1:x={},y={}".format(x,y))
- f(x,y)

```
1:x=2, y=3
2:x=3, y=2
```

- 例：关键字参数
- def f(y,x):
- 　　print("x={},y={}".format(x,y))
- 　　return
- f(x=2,y=3)

x=2, y=3

- 注意：关键字参数必须位于位置参数之后。
- 即，在关键字参数之后不能有位置参数。
- 例：
- def f(x,y,z):
- 　　print("x={},y={},z={}".format(x,y,z))
- def main():
- 　　f(2,z=3,y=4)
- main()

3. 默认值参数

- 默认值参数是指定义函数时，假设一个默认值，如果不提供参数的值，则取默认值。
- 默认值参数的格式为：
- 形参名=默认值
- 注意：默认值参数必须出现在形参表的最右端。即第一个形参使用默认值参数后，它后面（右侧）的所有形参也必须使用默认值参数，否则会出错。

- 例：
- def f(x,y=4,z=5):
- 　print("x={},y={},z={}".format(x,y,z))
- 　return
- f(2,3)

x=2, y=3, z=5

- 默认值参数等效为函数重载，提高了程序的灵活性
- 例：
- def f(x=1,y=2,z=3):
- 　　print("x={},y={},z={}".format(x,y,z))
- def main():
- 　　f()
- 　　f(4)
- 　　f(5,6)
- 　　f(7,8,9)
- main()

4. 可变长度参数（个数可变的参数）

- 即函数参数个数不固定，也称参数收集。
- 在Python中，有两种可变长度参数，分别是元组（非关键字参数）和字典（关键字参数）。

（1）元组可变长度参数

- 元组可变长度参数在参数名前面加*，用来接受任意多个实参并将其放在一个元组中。

- 例：
- def f(*t):
- print("t={}".format(t))
- return
- f(1,2,3)
- f(1,2,3,4,5)

```
t=(1, 2, 3)
t=(1, 2, 3, 4, 5)
```

- 注意：一个函数最多只能带一个支持“普通”参数收集的形参（元组可变长度参数）。
- 例：
- def f(x,*y,z):
- print("x={},y={},z={}".format(x,y,z))
- def main():
- f(1,2,3,4,5,6,7,8,z=9)
- main()

（2）字典可变长度参数

- 在函数的字典可变长度参数名前面加**，该字典参数即可接受任意多个实参。
- 字典可变长度参数的实参格式：
- 关键字=实参值
- 实参关键字和实参值将组合成一组“关键字：值”字典元素，该元素即为可变长度参数字典的元素。
- 所有其他类型的形式参数，必须放在可变长度参数之前（左侧）。

- 例：
- def f(**t):
- print("t={}".format(t))
- return
- f(x=1,y=2,z=3)
- f(m="abc",n=4)

```
t={'x': 1, 'y': 2, 'z': 3}
t={'m': 'abc', 'n': 4}
```

- 例：
- def f(x,*y,**z):
- print("x={},y={},z={}".format(x,y,z))
- def main():
- f(1,2,3,4,5,6,a=7,b=8,c=9)
- main()

- 例：
- def f(x,y,**z):
- print("x={},y={},z={}".format(x,y,z))
- def main():
- f(1,a=2,b=3,c=4,y=5)
- f(1,y=2,a=3,b=4,c=5)
- main()

- 例：写出下列程序的执行结果。

```
def f(x,y=30,*m,**n):
    t=[0,0,0]
    t[0]=x+y
    for i in range(0,len(m)):
        t[1]+=m[i]
    for j in n.values():
        t[2]+=j
    return t
s=f(1,2,3,4,5,6,a=7,b=8)
print("s=",s)
```

s= [3, 18, 15]

5. 逆向参数收集

- 即把列表、元组、字典等对象的元素“拆开”后传给函数的形式参数。
- 列表、元组对象的“拆开”是在其前面加“*”；
- 字典对象的“拆开”是在其前面加“**”；
- 字典“拆开”后将变成关键字参数的形式传递给函数的形参。

- 例：

```
def f(x,y,z):
    print("x={},y={},z={}".format(x,y,z))
def g(x,y,*z):
    print("x={},y={},z={}".format(x,y,z))
def h(x,*y):
    print("x={},y={}".format(x,y))
def main():
    a=[1,2,3]
    b={"x":22,"y":33,"z":44}
    c=(4,5,6,7,8)
    f(*a)
    f(**b)
    g(*c)
    h(c)
    h(*c)
main()
```

```
x=1, y=2, z=3
x=22, y=33, z=44
x=4, y=5, z=(6,  7,  8)
x=(4,  5,  6,  7,  8), y=()
x=4, y=(5,  6,  7,  8)
```

8.3 两类特殊函数

- Python有两类特殊函数：匿名函数和递归函数。
- 匿名函数是指没有函数名的简单函数，只可以包含一个表达式，不允许包含其他复杂的语句，表达式的结果是函数的返回值。

- 递归函数是指直接或间接调用函数本身的函数。
- 递归函数反映了一种逻辑思想，用它来解决某些问题时显得很简练。

1. 匿名函数的定义

- 在Python中，可以使用lambda关键字（即λ）来在同一行内定义函数，因为不用指定函数名，所以这个函数被称为匿名函数，也称为lambda函数。
- 用匿名函数有个好处，因为函数没有名字，所以不必担心函数名冲突。

- 定义格式为：
- lambda [参数1[,参数2,……,参数n]]:表达式
- 关键字lambda表示匿名函数，冒号前面是函数参数，可以有多个函数参数，但只有一个返回值，所以只能有一个表达式，返回值就是该表达式的结果。
- 匿名函数不能包含语句或多个表达式、不用写return语句。

2. 匿名函数的调用

- 匿名函数也是一个函数对象，也可以把匿名函数赋值给一个变量，再利用变量来调用该函数。
- 定义或调用匿名函数时也可以指定默认值参数和关键字参数。

- 例：
- >>> f=lambda x,y:x+y
- >>> f(2,3)
- 5
- >>> m=lambda x=2,y=3:x+y
- >>> m
- <function <lambda> at 0x01384348>
- >>> lambda x=2,y=3:x+y
- <function <lambda> at 0x021534B0>

- 例：lambda函数的定义与调用。
- f=lambda a,b=1,c=2:a+b+c #使用默认值参数
- print("1:",f(3))
- print("2:",f(4,5))
- print("3:",f(6,7,8))
- print("4:",f(c=10,a=20)) #使用关键字实参

```
1: 6
2: 11
3: 21
4: 31
```

3. 把匿名函数作为函数的返回值

- 可以把匿名函数作为普通函数的返回值返回。
- 例：
- def f():
- return lambda x,y:x+y
- g=f()
- print(g(2,3))

5

4. 把匿名函数作为序列或字典的元素

- 可以将匿名函数作为序列或字典的元素。一般格式（以列表为例）：
- 列表名=[匿名函数1,匿名函数2,……,匿名函数n]
- 这时可以以序列或字典元素引用作为函数名来调用匿名函数，一般格式为:
- 列表或字典元素引用(匿名函数实参)

- 例：
- >>> f=[lambda x,y:x+y,lambda x,y:x*y]
- >>> print(f[0](2,3),f[1](4,5))
- 5 20
- >>> g={"aa":lambda x,y:x+y,"bb":lambda x,y:x*y}
- >>> print(g["aa"](2,3),g["bb"](4,5))
- 5 20

8.3.2 递归函数

1. 递归的基本概念

- 递归（recursion）是指在连续执行某一处理过程时，该过程中的某一步要用到它自身的上一步或上几步的结果。
- 在一个程序中，若存在程序自己调用自己的现象就是构成了递归。

- 递归是一种常用的程序设计技术。
- 在实际应用中，许多问题的求解方法具有递归特征，可以利用递归描述这种求解算法。
- 通常，函数用递归描述比用循环控控制结构构描述更自然、更简洁，思路也更清晰。

- 递归函数是指一个函数的函数体中又直接或间接地调用该函数本身的函数。
- 如果函数a中又调用函数a自己，则称函数a为直接递归。
- 如果函数a中先调用函数b，函数b中又调用函数a，则称函数a为间接递归。
- 程序设计中常用的是直接递归。

- 数学上递归定义的函数是非常多的。
- 例：当n为自然数时，求n的阶乘n!。

$$n! = \begin{cases} 1 & n \leqslant 1 \\ n(n-1)! & n > 1 \end{cases}$$

2. 递归函数的调用过程

- 例：求n!的递归函数。

```
def f(n):
    if n<=1:
        return 1
    else:
        return n*f(n-1)
m=f(3)
print("3!=",m)
```

- 编写递归程序要注意两点：
- （1）要找出正确的递归算法，这是编写递归程序的基础;
- （2）要确定算法的递归结束条件，这是决定递归程序能否正常结束的关键。否则，将“死递归”（类似“死循环”、“无限循环”）。

- 例：写出下面程序的运行结果。

```
def f(n):
    if(n==0 or n==1):
        return 1
    else:
        return n+f(n-1)+f(n-2)
def main():
    x=f(3)
    print(x)
    return
main()
```

- 例：写出下面程序的运行结果。

```
def f(n):
    print(n)
    if(n==1):
        return
    else:
        f(n-1)
    print(n)
def main():
    f(3)
    return
main()
```

- 例：分析下面程序的功能。
- def f(n):
- 　if(n>=2):
- 　　f(n//2)
- 　print(n%2,end="")
- def main():
- 　f(23)
- 　return
- main()

- 例：分析下面程序的功能。

```
def f(n):
    if(n>=8):
        f(n//8)
    print(n%8,end="")
def main():
    f(123)
    return
main()
```

- 例：分析下面程序的功能。

```
def f(n):
    if(n>=16):
        f(n//16)
    t=n%16
    if(0<=t<=9):
        print(t,end="")
    else:
        print(chr(ord('A')+t-10),end="")
def main():
    f(123)
    return
main()
```

- 例：分析下面程序的功能。
- def f(n):
- 　if(n>=16):
- 　　f(n//16)
- 　t=n%16
- 　print(t if 0<=t<=9 else chr(ord('A')+t-10),end="")
- def main():
- 　f(123)
- 　return
- main()

- 例：用递归方法计算下列多项式函数的值。

$$p(x,n)=x-x^2+x^3-x^4+\cdots+(-1)^{n-1}x^n(n>0)$$

数学变换：$p(x,n)=x-x^2+x^3-x^4+\cdots(-1)^{n-1}x^n$

$$=x[1-(x-x^2+x^3-\cdots+(-1)n-2x^{n-1})]$$

$$=x[1-p(x,n-1)]$$

递归定义：$$p(x,n)=\begin{cases}x & n=1\\ x[1-p(x,n-1)] & n>1\end{cases}$$

```
def p(x,n):
    if n==1:
        return x
    else:
        return x*(1-p(x,n-1))
print(p(2,4))
```

$$p(x,n) = \begin{cases} x & n = 1 \\ x\left[1 - p(x,n-1)\right] & n > 1 \end{cases}$$

3. 递归函数的特点

优点：

- 当一个问题蕴含了递归关系且结构比较复杂时，采用递归函数可以使程序变得简洁、紧凑，能够很容易地解决一些用非递归算法很难解决的问题。

缺点：

- 递归函数是以牺牲存储空间为代价的，因为每一次递归调用都要保存相关的参数和变量。
- 递归函数也会影响程序执行速度，由于反复调用函数，会增加时间开销。

- 所有的递归问题都可以用非递归的算法实现，并且已经有了固定的算法。（略，自学）

8.4 函数的参数传递机制

- 值传递：单向传递，把函数的实在参数拷贝给形式参数，形参的改变不影响实参
- 例：

```
def f(x,y):
    x=4
    y=5
def main():
    x=2
    y=3
    f(x,y)
    print("x={},y={}".format(x,y))
main()
```

- 引用（地址）传递：双向传递，函数的实在参数与形式参数共享内存，形参的改变要影响实参
- 当列表、字典等可变的复合数据类型作函数的形式参数时，将采用引用传递。
- 例：

```
def f(s):
    s[0]=4
    s[1]=5
def main():
    t=[2,3]
    f(t)
    print("t={}".format(t))
main()
```

- 例：
- def f(s):
- 　　s["x"]=4
- 　　s["y"]=5
- def main():
- 　　t={"x":2,"y":3}
- 　　f(t)
- 　　print("t={}".format(t))
- main()

8.5 变量的作用域

- Python程序可以由若干函数组成，每个函数都要用到一些变量。
- 在程序中能对变量进行存取操作的范围称为变量的作用域。
- 作用域也可理解为一个变量的命名空间。

- 程序中变量被赋值的位置，就决定了哪些范围的对象可以访问这个变量，这个范围就是命名空间。
- **Python**在给变量赋值时生成了变量名，当然作用域也就确定了。
- 根据变量的作用域不同，变量分为局部变量和全局变量。

8.5.1 局部变量

- 在一个函数体内或语句块内定义的变量称为局部变量。
- 局部变量只在定义它的函数体或语句块内有效，即只能在定义它的函数体或语句块内部使用它，而在定义它的函数体或语句块之外不能使用它。

- 例：
- def fun1(x):
- 　　m,n=1,2
- 　　print(m,n,x) #此处可使用形参x和局部变量m,n
- 　　return
- def fun2(x,y):
- 　　m,n=3,4
- 　　print(x,y,m,n) #此处可使用形参x,y和局部变量m,n
- 　　return
- def main():
- 　　a,b=5,6
- 　　print(a,b) #此处可使用a,b
- 　　fun1(a)
- 　　fun2(a,b)
- 　　return
- main()

- 不同的函数可以使用相同的标识符命名各自的变量。
- 同一名字在不同函数中代表不同对象，互不干扰。
- 对于带参数的函数来说，形式参数的有效范围也局限于函数体。
- 同理，同一标识符可作为不同函数的形参名，它们也被作为不同对象。

8.5.2 全局变量

- 在函数定义之外定义的变量称为全局变量，它可以被多个函数引用。
- 在函数体中，如果要为在函数外的全局变量重新赋值，可以使用global语句，表明变量是全局变量。

- 例：
- x=22　#定义全局变量
- def f():
- 　print("1:x=",x)
- def g():
- 　print("2:x=",x)
- f()
- g()
- x=33
- f()
- g()

```
1:x= 22
2:x= 22
1:x= 33
2:x= 33
```

- 例：
- def f():
- 　　x=44
- 　　print("1:x=",x)
- 　　x=55
- def g():
- 　　print("2:x=",x)
- x=22
- f()
- g()
- x=33
- f()
- g()

```
1:x= 44
2:x= 22
1:x= 44
2:x= 33
```

- 例：写出以下程序输出结果。
- def f():
- 　　global x　#说明x为全局变量
- 　　x=30
- 　　y=40　　#定义局部变量y
- 　　print("2:",x,y)
- x=10　　#定义全局变量x
- y=20　　#定义全局变量y
- print("1:",x,y)
- f()
- print("3:",x,y)

- 在同一程序文件中，如果全局变量与局部变量同名，则在局部变量的作用范围内，全局变量不起作用。

- 例： 写出程序的输出结果。

```
def f():
    global x
    x='ABC'
    def g():
        global x
        x+='abc'
        return x
    return g()
print(f())
```

ABCabc

- 引入nonlocal关键字：只要在内层函数中用nonlocal语句说明变量，就可以让解释器在外层函数中修改变量的值。

- 例：
- def f():
- x='ABC'
- def g():
- nonlocal x
- x+='abc'
- return x
- return g()
- print(f())

ABCabc

- 在程序中定义全局变量的主要目的是，为函数间的数据联系提供一个直接传递的通道。
- 在某些应用中，函数将执行结果保留在全局变量中，使函数能返回多个值。
- 在另一些应用中，将部分参数信息放在全局变量中，以减少函数调用时的参数传递。

- 因程序中的多个函数能使用全局变量，其中某个函数改变全局变量的值就可能影响其他函数的执行，产生副作用。
- 因此，不宜过多使用全局变量。

8.6 模块

- Python模块可以在逻辑上组织Python程序，将相关的程序组织到一个模块中，使程序具有良好的结构，增加程序的重用性。
- 模块可以被别的程序导入，以调用该模块中的函数，这也是使用Python标准库模块的方法。

8.6.1 模块的定义与使用

- Python模块是比函数更高级别的程序组织单元，一个模块可以包含若干个函数。
- 与函数相似，模块也分标准库模块和用户自定义模块。

1. 标准库模块

- 标准库模块是Python自带的函数模块，也称为标准链接库。
- Python提供了大量的标准库模块，实现了很多常见功能，包括数学运算、字符串处理、操作系统功能、网络和Internet编程、图形绘制、图形用户界面创建，等等，这些为应用程序开发提供了强大支持。

- 标准库模块并不是Python语言的组成部分，而是由专业开发人员预先设计好并随语言提供给用户使用的。
- 用户可以在安装了标准Python系统的情况下，通过导入命令来使用所需要的模块。
- 标准库模块种类繁多，可以使用Python的联机帮助命令来熟悉和了解标准库模块。

2. 用户自定义模块

- 用户自定义一个模块就是建立一个Python程序文件，其中包括变量、函数的定义。
- 例：
- #aa.py
- x=2
- def f(y):
- 　　print("y=",y)
- 　　return

- 通过执行import语句来导入Python模块。
- 语句格式：
- import 模块名1[,模块名2[,……,模块名n]]
- 当Python解释器执行import语句时，如果模块文件出现在搜索路径中，则导入相应的模块。

- 例：
- # a.py
- def f(m,n):
- print("m+n=",m+n);
- ##################################
- >>> import a
- >>> a.f(2,3)
- m+n= 5

- from语句可以从一个模块中导入特定的项目（函数、变量）到当前的命名空间。
- 语句格式：
- from 模块名 import 项目名1[,项目名2[,……,项目名n]]
- 此语句不导入整个模块到当前的命名空间，而只是导入指定的项目，这时在调用函数时不需要加模块名作为限制。

- 例：
- # a.py
- def f(m,n):
- print("m+n=",m+n);
- ##################################
- >>> from a import f
- >>> f(4,5)
- m+n= 9

- 例：
- #a.py
- def f(x,y):
- 　　return x+y
- def g(x,y):
- 　　return x*y

- 例：
- >>> from a import f,g
- >>> print(f(2,3),g(2,3))
- 5 6

- 也可通过import语句一次性导入模块的所有项目（函数、变量）到当前的命名空间。
- 语句格式：
- from import 模块名 *
- 注意：这将把所有的名字都导入进来，但是那些名称由单一下画线“_”开头的项目不在此列。
- 大多数情况下，Python程序员不使用这种方法，因为导入的其他来源的项目名称，很可能覆盖已有的定义。

- 例：
- # a.py
- def f(m,n):
- print("m+n=",m+n);
- ##################################
- >>> from a import *
- >>> f(6,7)
- m+n= 13

- 例：创建一个fibo.py模块，其中包含两个求Fibonacci数列的函数。
- def fib1(n):
- a,b=0,1
- while b<n:
- print(b,end=' ')
- a,b=b,a+b
- print()
- def fib2(n):
- result=[]
- a,b=0,1
- while b<n:
- result.append(b)
- a,b=b,a+b
- return result

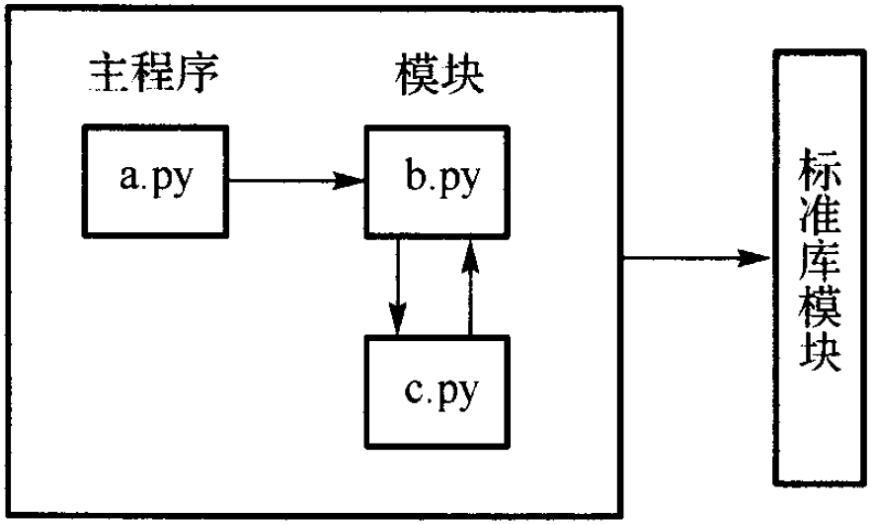
- 例：导入fibo.py模块，并调用其中的函数。
- >>>import fibo
- >>>fibo.fib1(1000)
- >>>fibo.fib2(100)
- >>>from fibo import * #把所有项目都导入进来
- >>>fib1(500)

- 注意：
- from 模块名 import *
- 该语句一次性把模块中的所有函数、变量都导入到当前名称空间，这样就可以直接调用函数了。
- 但该语句导入的项目中并不包括那些名称由单一下画线(_)开头的项目。
- 通常不提倡使用这种方法：因为导入的其他模块中的同名项目，会覆盖当前名称空间中自己定义的同名项目。

8.6.2 Python程序结构

- 简单的程序可以只用一个程序文件实现，但对于绝大多数Python程序，一般都是由多个程序文件组成的，其中每个程序文件就是一个.py源程序文件。
- Python程序的结构是指将一个求解问题的程序分解为若干源程序文件的集合以及将这些文件连接在一起的方法。

- Python程序通常由一个主程序以及多个模块组成。
- 主程序定义了程序的主控流程，是执行程序时的启动文件，属于顶层文件。
- 模块则是函数库，相当于子程序。
- 模块是用户自定义函数的集合体，主程序可以调用模块中定义的函数来完成应用程序的功能，还可以调用标准库模块，同时模块也可以调用其他模块或标准库模块定义的函数。



Python 程序结构

```
#文件a.py
import b,c
b.hello("AA")
b.bye("BB")
c.show(2)
```

```
#文件b.py
import math
def hello(person):
    print("Hello",person)
def bye(person):
    print("Bye",person)
def disp(r):
    print(math.pi*r*r)
```

```
#文件c.py
import b
def show(n):
    b.disp(n)
```

8.6.3 模块的有条件执行

- 每一个Python程序文件都可以当成一个模块，模块以磁盘文件的形式存在。
- 模块中可以是一段可以直接执行的程序（也称为脚本），也可以定义一些变量、类或函数，让别的模块导入和调用，类似于库函数。

- 模块中的定义部分，例如全局变量定义、类定义、函数定义等，因为没有程序执行入口，所以不能直接运行，但对主程序代码部分有时希望只让它在模块直接执行的时候才执行，被其他模块加载时就不执行。
- 在Python中，可以通过系统变量“__name__”（注意前后都是两个下画线）的值来区分这两种情况。

- __name__是一个全局变量，在模块内部是用来标识模块名称的。
- 如果模块是被其他模块导入的，__name__的值是模块的名称，主动执行时它的值就是字符串“__main__”。
- 例：建立模块m.py
- def test():
- print(__name__)
- test()

- 在Python交互方式下第一次执行import导入命令，打印的__name__值就是模块的名称：
- >>>import m
- m
- 如果通过Python解释器直接执行模块，则__name__会被设置为“__main__”这个字符串值：
- __main__
- 通过__name__变量的这个特性，可以将一个模块文件既作为普通的模块库供其他模块使用，又可以作为一个可执行文件进行执行。

- 具体做法是在程序执行入口之前加上if判断语句，即模块m.py写成：
- def test():
- 　print(__name__)
- if __name__=='__main__':
- 　test()
- 当使用import命令导入m.py时，__name__变量的值是模块名“m”，所以不执行test()函数调用。
- 当运行m.py时，__name__变量的值是“__main__”，所以执行test()函数调用。

8.7 函数应用举例

- 应用计算机求解复杂的实际问题，总是把一个任务按功能分成若干个子任务，每个子任务还可再进一步分解。
- 一个子任务称为一个功能模块，在Python中用函数实现。

- 一个大型程序往往由许多函数组成，这样便于程序的调试和维护，所以设计功能和数据独立的函数是软件开发中的最基本的工作。

例： 求 $y = e^2 + \sum_{n=1}^{100} \frac{1+\ln n}{2\pi}$。

- from math import *
- f=lambda n:(1+log(n))/(2*pi)
- y=exp(2.0)
- for n in range(1,101):
- 　　y+=f(n)
- print('y=',y)

例：先定义函数求 $\sum_{i=1}^{n} i^m$，然后调用该函数求 $s=\sum_{k=1}^{100} k+\sum_{k=1}^{50} k^2+\sum_{k=1}^{10} \frac{1}{k}$。

```
def mysum(n,m):
    s=0
    for i in range(1,n+1):
        s+=i**m
    return s
def main():
    s=mysum(100,1)+mysum(50,2)+mysum(10,-1)
    print("s=",s)
main()
```

自测题

- 一、选择题
- 1. 下列选项中不属于函数优点的是（　　）。D
- A. 减少代码重复
- B. 使程序模块化
- C. 使程序便于阅读
- D. 便于发挥程序员的创造力

- 2．关于函数的说法中正确的是（　　）。B
- A．函数定义时必须有形参
- B．函数中定义的变量只在该函数体中起作用
- C．函数定义时必须带return语句
- D．实参与形参的个数可以不相同，类型可以任意

- 3．以下关于函数说法正确的是（　　）。D
- A．函数的实际参数和形式参数必须同名
- B．函数的形式参数既可以是变量也可以是常量
- C．函数的实际参数不可以是表达式
- D．函数的实际参数可以是其他函数的调用

- 4．有以下两个程序。
- 程序一：
- x=[1,2,3]
- def f(x):
- 　　x=x+[4]
- f(x)
- print(x)
- 程序二：
- x=[1,2,3]
- def f(x):
- 　　x+=[4]
- f(x)
- print(x)

- 下列说法正确的是（　　）。A
- A．两个程序均能正确运行，但结果不同
- B．两个程序的运行结果相同
- C．程序一能正确运行，程序二不能
- D．程序一不能正确运行，程序二能

- 5．已知f=lambda x,y:x+y，则f([4],[1,2,3])的值是（　　）。C
- A．[1, 2, 3, 4]　　B．10
- C．[4, 1, 2, 3]　　D．{1, 2, 3, 4}
- 6．下列程序的运行结果是（　　）。D
- f=[lambda x=1:x*2,lambda x:x**2]
- print(f[1](f[0](3)))
- A．1　　B．6
- C．9　　D．36

- 7．下列程序的运行结果是（　　）。B
- def f(x=2,y=0):
- 　　return x-y
- y=f(y=f(),x=5)
- print(y)
- A．-3　　B．3　　C．2　　D．5

- 8．output.py文件和test.py文件内容如下，且output.py和test.py位于同一文件夹中，那么运行test.py的输出结果是（　　）。A
- #output.py
- def show():
- 　print(__name__)
- #test.py
- import output
- if __name__=='__main__':
- 　output.show()
- A．output　　B．__name__
- C．test　　D．__main__

- 二、填空题
- 1．函数首部以关键字（　）开始，最后以（　）结束。def，冒号
- 2．函数执行语句“return [1,2,3],4”后，返回值是（　）；没有return语句的函数将返回（　）。([1, 2, 3], 4)，None
- 3．使用关键字（　）可以在一个函数中设置一个全局变量。global

- 4．下列程序的输出结果是（　）。40
- counter=1
- num=0
- def TestVariable():
- 　global counter
- 　for i in (1,2,3):counter+=1
- 　num=10
- TestVariable()
- print(counter,num)

- 5．设有f=lambda x,y:{x:y}，则f(5,10)的值是（　）。{5: 10}
- 6．Python包含了数量众多的模块，通过（　）语句，可以导入模块，并使用其定义的功能。

  import
- 7．设Python中有模块m，如果希望同时导入m中的所有成员，则可以采用（　）的导入形式。

  from m import *

- 8．Python中每个模块都有一个名称，通过特殊变量（　）可以获取模块的名称。特别地，当一个模块被用户单独运行时，模块名称为（　）。
- __name__,__main__

- 9．建立模块a.py，模块内容如下。
- def B():
- 　print('BBB')
- def A():
- 　print('AAA')
- 为了调用模块中的A()函数，应先使用语句（　）。

  from a import A

- 三、简答题
- 1．什么叫递归函数？举例说明。
- 2．什么叫lambda函数？举例说明。
- 3．什么叫模块？如何导入模块？

- 4．写出下列程序的输出结果。
- def ff(x,y=100):
- 　　return {x:y}
- print(ff(y=10,x=20))
- ##################################
- {20: 10}

- 5．分析下面的程序。
- x=10
- def f():
- 　　#y=x
- 　　x=0
- 　　print(x)
- print(x)
- f()

- （1）函数f()中的x和程序中的x是同一个变量吗？程序的输出结果是什么？
- （2）删除函数f()中第一个语句前面的“#”，此时运行程序会出错，为什么？
- （3）删除函数f()中第一个语句前面的“#”，同时在函数f()中第二个语句前面加“#”，此时程序能正确运行，为什么？写出运行结果。