函数的设计与使用

2024年4月20日

目录

5.1 函数定义

- 将可能需要反复执行的代码封装为函数，并在需要该功能的地方进行调用，不仅可以实现**代码复用**，更重要的是可以保证代码的**一致性**，只需要修改该函数代码则所有调用均受到影响。
- 设计函数时，应注意**提高模块的内聚性**，同时**降低模块之间的隐式耦合**。
- 在实际项目开发中，往往会把一些通用的函数封装到一个模块中，并把这个通用模块文件放到顶层文件夹中，这样更方便管理。

5.1 函数定义

- 在编写函数时，应尽量减少副作用，尽量不要修改参数本身，不要修改除返回值以外的其他内容。
- 应充分利用Python函数式编程的特点，让自己定义的函数尽量符合纯函数式编程的要求，例如保证线程安全、可以并行运行等等。

5.1 函数定义

❖ 函数定义语法：

```
def 函数名([参数列表]):
    '''注释'''
    函数体
```

❖ 注意事项

- ✓ 函数形参不需要声明其类型，也不需要指定函数返回值类型
- ✓ 即使该函数不需要接收任何参数，也必须保留一对空的圆括号
- ✓ 括号后面的冒号必不可少
- ✓ 函数体相对于def关键字必须保持一定的空格缩进
- ✓ Python允许嵌套定义函数

5.1 函数定义

- 生成斐波那契数列的函数定义和调用

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

fib(1000)
```

5.1 函数定义

- 在定义函数时，开头部分的注释并不是必需的，但是如果为函数的定义加上这段注释的话，可以为用户提供友好的提示和使用帮助。

```
>>> def fib(n):
    '''accept an integer n.
       return the numbers less than n in Fibonacci sequence.'''
    a, b = 1, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

>>> fib(
      (n)
      accept an integer n.
      return the numbers less than n in Fibonacci sequence.
```

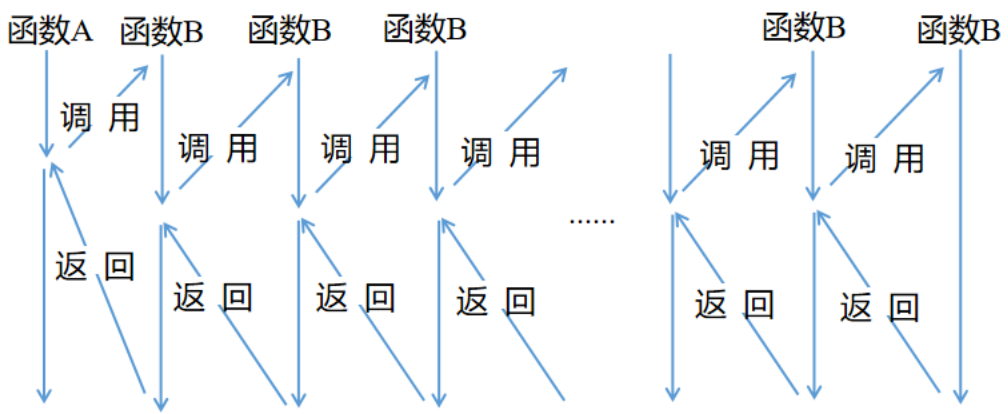
5.1 函数定义

- Python是一种高级动态编程语言，变量类型是随时可以改变的。Python中的函数和自定义对象的成员也是可以随时发生改变的，可以为函数和自定义对象动态增加新成员。

```
>>> def func():
        print(func.x)           #查看函数func的成员x
>>> func()                  #现在函数func还没有成员x，出错
AttributeError: 'function' object has no attribute 'x'
>>> func.x = 3              #动态为函数增加新成员
>>> func()
3
>>> func.x                  #在外部也可以直接访问函数的成员
3
>>> del func.x              #删除函数成员
>>> func()                  #删除之后不可访问
AttributeError: 'function' object has no attribute 'x'
```

5.1 函数定义

- 函数的**递归调用**是函数调用的一种特殊情况，函数调用自己，自己再调用自己，自己再调用自己，…，当某个条件得到满足的时候就不再调用了，然后再一层一层地返回直到该函数的第一次调用。



5.2 形参与实参

- 函数定义时括弧内为形参，一个函数可以没有形参，但是括弧必须要有，表示该函数不接受参数。
- 函数调用时向其传递实参，将实参的值或引用传递给形参。
- 在定义函数时，对参数个数并没有限制，如果有多个形参，需要使用逗号进行分割。

5.2 形参与实参

- 编写函数，接受两个整数，并输出其中最大数。

```
def printMax(a, b):
    if a>b:
        pirnt(a, 'is the max')
    else:
        print(b, 'is the max')
```

5.2 形参与实参

- 对于绝大多数情况下，在函数内部直接修改形参的值不会影响实参。例如：

```
>>> def addOne(a):
    print(a)
    a += 1
    print(a)
>>> a = 3
>>> addOne(a)
3
4
>>> a
3
```

5.2 形参与实参

- 在有些情况下，可以通过特殊的方式在函数内部修改实参的值，例如下面的代码。

```
>>> def modify(v):          #修改列表元素值
    v[0] = v[0]+1
>>> a = [2]
>>> modify(a)
>>> a
[3]
>>> def modify(v, item):    #为列表增加元素
    v.append(item)
>>> a = [2]
>>> modify(a,3)
>>> a
[2, 3]
```

5.2 形参与实参

- 也就是说，如果传递给函数的是可变序列，并且在函数内部使用下标或可变序列自身的方法增加、删除元素或修改元素时，修改后的结果是可以反映到函数之外的，实参也得到相应的修改。

```
>>> def modify(d): #修改字典元素值或为字典增加元素
    d['age'] = 38
>>> a = {'name':'Dong', 'age':37, 'sex':'Male'}
>>> a
{'age': 37, 'name': 'Dong', 'sex': 'Male'}
>>> modify(a)
>>> a
{'age': 38, 'name': 'Dong', 'sex': 'Male'}
```

5.3 参数类型

- 在Python中，函数参数有很多种：可以为普通参数、默认值参数、关键参数、可变长度参数等等。
- Python在定义函数时**不需要指定形参的类型**，完全由调用者传递的实参类型以及Python解释器的理解和推断来决定，类似于重载和泛型。
- Python函数定义时**也不需要指定函数的类型**，这将由函数中的return语句来决定，如果没有return语句或者return没有得到执行，则认为返回空值None。

5.3 参数类型

- Python支持对函数参数和返回值类型的标注，但**实际上并不起任何作用**，只是看起来方便。

```
>>> def test(x:int, y:int) -> int:
        '''x and y must be integers, return an integer x+y'''
        assert isinstance(x, int), 'x must be integer'
        assert isinstance(y, int), 'y must be integer'
        z = x+y
        assert isinstance(z, int), 'must return an integer'
        return z
>>> test(1, 2)
3
>>> test(2, 3.0)                #参数类型不符合要求，抛出异常
AssertionError: y must be integer
```

5.3 参数类型

- 位置参数（positional arguments）是比较常用的形式，调用函数时实参和形参的顺序必须严格一致，并且实参和形参的数量必须相同。

```
>>> def demo(a, b, c):
        print(a, b, c)

>>> demo(3, 4, 5)              #按位置传递参数
3 4 5
>>> demo(3, 5, 4)
3 5 4
>>> demo(1, 2, 3, 4)           #实参与形参数量必须相同
TypeError: demo() takes 3 positional arguments but 4 were given
```

5.3.1 默认值参数

- 默认值参数必须出现在函数参数列表的最右端，且任何一个默认值参数右边不能有非默认值参数。

```
>>> def f(a=3,b,c=5):
        print a,b,c

SyntaxError: non-default argument follows default argument
>>> def f(a=3,b):
        print a,b

SyntaxError: non-default argument follows default argument
>>> def f(a,b,c=5):
        print a,b,c


>>>
```

5.3.1 默认值参数

- 调用带有默认值参数的函数时，可以不对默认值参数进行赋值，也可以赋值，具有较大的灵活性。

```
>>> def say( message, times =1 ):
        print(message * times)
>>> say('hello')
hello
>>> say('hello',3)
hello hello hello
>>> say('hi',7)
hi hi hi hi hi hi hi
```

5.3.1 默认值参数

- 再例如，下面的函数使用指定分隔符将列表中所有字符串元素连接成一个字符串。

```
>>> def Join(List,sep=None):
    return (sep or ' ').join(List)
>>> aList = ['a', 'b', 'c']
>>> Join(aList)
'a b c'
>>> Join(aList, ',')
'a,b,c'
```

5.3.1 默认值参数

- 默认值参数如果使用不当，会导致很难发现的逻辑错误，例如：

```
def demo(newitem,old_list=[]):
    old_list.append(newitem)
    return old_list
print(demo('5',[1,2,3,4]))   #right
print(demo('aaa',['a','b'])) #right
print(demo('a'))             #right
print(demo('b'))             #wrong
```

想一想，这段代码会输出什么呢？

5.3.1 默认值参数

- 上面的代码输出结果如下，最后一个结果是错的。

```
[1, 2, 3, 4, '5']
['a', 'b', 'aaa']
['a']
['a', 'b']
```

为什么会这样呢?

5.3.1 默认值参数

- 原因在于默认值参数的赋值只会在函数定义时被解释一次。当使用可变序列作为参数默认值时，一定要谨慎操作。
- 正确的代码该怎么写呢?

5.3.1 默认值参数

- 终极解决方案：改成下面的样子就不会有问题了

```
def demo(newitem,old_list=None):
    if old_list is None:
        old_list=[]
    old_list.append(newitem)
    return old_list
print(demo('5',[1,2,3,4]))
print(demo('aaa',['a','b']))
print(demo('a'))
print(demo('b') )
```

5.3.1 默认值参数

- 注意：

✓ 默认值参数只在函数定义时被解释一次

✓ 可以使用“函数名.__defaults__”查看所有默认参数的当前值

```
>>> i = 3
>>> def f(n=i):                #参数n的值仅取决于i的当前值
        print(n)
>>> f()
3
>>> i = 5                      #函数定义后修改i的值不影响参数n的默认值
>>> f()
3
```

5.3.2 关键字参数

- 关键字参数主要指实参，即调用函数时的参数传递方式。
- 通过关键字参数，实参顺序可以和形参顺序不一致，但不影响传递结果，避免了用户需要牢记位置参数顺序的麻烦。

```
>>> def demo(a,b,c=5):
        print(a,b,c)
>>> demo(3,7)
3 7 5
>>> demo(a=7,b=3,c=6)
7 3 6
>>> demo(c=8,a=9,b=0)
9 0 8
```

5.3.3 可变长度参数

- 可变长度参数主要有两种形式：

  - *parameter用来接受多个实参并将其放在一个元组中
  - **parameter接受关键字参数并存放到字典中

5.3.3 可变长度参数

- *parameter的用法

```
>>> def demo(*p):
    print(p)

>>> demo(1,2,3)
(1, 2, 3)
>>> demo(1,2)
(1, 2)
>>> demo(1,2,3,4,5,6,7)
(1, 2, 3, 4, 5, 6, 7)
```

5.3.3 可变长度参数

- *parameter的用法

```
>>> def demo(**p):
        for item in p.items():
                print(item)

>>> demo(x=1,y=2,z=3)
('y', 2)
('x', 1)
('z', 3)
```

5.3.3 可变长度参数

- 几种不同类型的参数可以混合使用，但是不建议这样做

```
>>> def func_4(a,b,c=4,*aa,**bb):
    print(a,b,c)
    print(aa)
    print(bb)

>>> func_4(1,2,3,4,5,6,7,8,9,xx='1',yy='2',zz=3)
(1, 2, 3)
(4, 5, 6, 7, 8, 9)
{'yy': '2', 'xx': '1', 'zz': 3}
>>> func_4(1,2,3,4,5,6,7,xx='1',yy='2',zz=3)
(1, 2, 3)
(4, 5, 6, 7)
{'yy': '2', 'xx': '1', 'zz': 3}
```

5.3.4 参数传递的序列解包

- 传递参数时，可以通过在实参序列前加星号将其解包，然后传递给多个单变量形参。

```
>>> def demo(a, b, c):
    print(a+b+c)

>>> seq = [1, 2, 3]
>>> demo(*seq)
6
>>> tup = (1, 2, 3)
>>> demo(*tup)
6
```

```
>>> dic = {1:'a', 2:'b', 3:'c'}
>>> demo(*dic)
6
>>> Set = {1, 2, 3}
>>> demo(*Set)
6
>>> demo(*dic.values())
abc
```

5.3.4 参数传递的序列解包

- 注意：调用函数时如果对实参使用一个星号*进行序列解包，这么这些解包后的实参将会被当做普通位置参数对待，并且会在关键参数和使用两个星号**进行序列解包的参数之前进行处理。

5.3.4 参数传递的序列解包

```
>>> def demo(a, b, c):                #定义函数
        print(a, b, c)
>>> demo(*(1, 2, 3))                  #调用，序列解包
1 2 3
>>> demo(1, *(2, 3))                  #位置参数和序列解包同时使用
1 2 3
>>> demo(1, *(2,), 3)
1 2 3
```

5.3.4 参数传递的序列解包

```
>>> demo(a=1, *(2, 3))        #序列解包相当于位置参数，优先处理
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    demo(a=1, *(2, 3))
TypeError: demo() got multiple values for argument 'a'

>>> demo(b=1, *(2, 3))
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    demo(b=1, *(2, 3))
TypeError: demo() got multiple values for argument 'b'
>>> demo(c=1, *(2, 3))
2 3 1
```

5.3.4 参数传递的序列解包

```
>>> demo(**{'a':1, 'b':2}, *(3,)) #序列解包不能在关键参数解包之后
SyntaxError: iterable argument unpacking follows keyword argument unpacking

>>> demo(*(3,), **{'a':1, 'b':2})
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    demo(*(3,), **{'a':1, 'b':2})
TypeError: demo() got multiple values for argument 'a'
>>> demo(*(3,), **{'c':1, 'b':2})
3 2 1
```

5.4 return语句

- return语句用来从一个函数中返回一个值，同时结束函数。
- 如果函数没有return语句，或者有return语句但是没有执行到，或者只有return而没有返回值，Python将认为该函数以return None结束。

```
def maximum( x, y ):
    if x>y:
            return x
    else:
            return y
```

5.4 return语句

- 在调用函数或对象方法时，一定要注意有没有返回值，这决定了该函数或方法的用法。

```
>>> a_list = [1, 2, 3, 4, 9, 5, 7]
>>> print(sorted(a_list))
[1, 2, 3, 4, 5, 7, 9]
>>> print(a_list)
[1, 2, 3, 4, 9, 5, 7]
>>> print(a_list.sort())
None
>>> print(a_list)
[1, 2, 3, 4, 5, 7, 9]
```

5.5 变量作用域

- 变量起作用的代码范围称为变量的作用域，不同作用域内变量名可以相同，互不影响。
- 一个变量在函数外部定义和在函数内部定义，其作用域是不同的。
- 在函数内部定义的普通变量只在函数内部起作用，称为局部变量。当函数执行结束后，局部变量自动删除，不再可以使用。
- 局部变量的引用比全局变量速度快，应优先考虑使用。

5.5 变量作用域

- 如果想要在函数内部给一个定义在函数外的变量赋值，那么这个变量就不能是局部的，其作用域必须为全局的，能够同时作用于函数内外，称为全局变量，可以通过global来定义。这分为两种情况：

✓ 一个变量已在函数外定义，如果在函数内需要为这个变量赋值，并要将这个赋值结果反映到函数外，可以在函数内用global声明这个变量，将其声明为全局变量。

✓ 在函数内部直接将一个变量声明为全局变量，在函数外没有声明，该函数执行后，将增加为新的全局变量。

5.5 变量作用域

1. 修改全局作用域中已定义的变量

```
# 全局变量的定义
global_var = 10

def modify_global_var():
    global global_var  # 声明global_var为全局变量
    global_var = 20  # 修改全局变量的值

modify_global_var()
print(global_var)  # 输出 20，全局变量的值被修改
```

5.5 变量作用域

2. 在函数内部创建新的全局变量

```
>>> def demo():
    global x
    x = 3
    y =4
    print(x,y)

>>> x = 5
>>> demo()
3  4
>>> x
3
>>> y
NameError: name 'y' is not defined
```

```
>>> del x
>>> x
NameError: name 'x' is not defined
>>> demo()
3  4
>>> x
3
>>> y
NameError: name 'y' is not defined
```

5.5 变量作用域

- 注意：在某个作用域内只要有为变量赋值的操作，该变量在这个作用域内就是局部变量，除非使用global进行了声明。

```
>>> x = 3
>>> def f():
        print(x)        #本意是先输出全局变量x的值，但是不允许这样做
        x = 5           #有赋值操作，因此在整个作用域内x都是局部变量
        print(x)
>>> f()
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    f()
  File "<pyshell#9>", line 2, in f
    print(x)
UnboundLocalError: local variable 'x' referenced before assignment
```

5.5 变量作用域

- 如果局部变量与全局变量具有相同的名字，那么该局部变量会在自己的作用域内隐藏同名的全局变量。

```
>>> def demo():
        x = 3      #创建了局部变量，并自动隐藏了同名的全局变量
>>> x = 5
>>> x
5
>>> demo()
>>> x            #函数执行不影响外面全局变量的值
5
```

5.5 变量作用域

- 如果需要在同一个程序的不同模块之间共享全局变量的话，可以编写一个专门的模块来实现这一目的。例如，假设在模块A.py中有如下变量定义：

```
global_variable = 0
```

- 而在模块B.py中包含以下用来设置全局变量的语句：

```
import A
A.global_variable = 1
```

- 在模块C.py中有以下语句来访问全局变量的值：

```
import A
print(A.global_variable)
```

5.5 变量作用域

- 除了局部变量和全局变量，Python还支持使用nonlocal关键字定义一种介于二者之间的变量。关键字nonlocal声明的变量会引用距离最近的非全局作用域的变量，**要求声明的变量已经存在**，关键字nonlocal不会创建新变量。

5.5 变量作用域

```
def scope_test():
    def do_local():
        spam = "我是局部变量"

    def do_nonlocal():
        nonlocal spam        #这时要求spam必须是已存在的变量
        spam = "我不是局部变量，也不是全局变量"

    def do_global():
        global spam          #如果全局作用域内没有spam，就自动新建一个
        spam = "我是全局变量"

    spam = "原来的值"
    do_local()
    print("局部变量赋值后：", spam)
    do_nonlocal()
    print("nonlocal变量赋值后：", spam)
    do_global()
    print("全局变量赋值后：", spam)

scope_test()
print("全局变量：", spam)
```

5.6 lambda表达式

- lambda表达式可以用来声明匿名函数，也就是没有函数名字的临时使用的小函数，尤其适合需要一个函数作为另一个函数参数的场合。
- lambda表达式只可以包含一个表达式，该表达式的计算结果可以看作是函数的返回值，不允许包含其他复杂的语句，但在表达式中可以调用其他函数。

5.6 lambda表达式

```
>>> f = lambda x,y,z:x+y+z        #可以给lambda表达式起名字
>>> f(1,2,3)                      #像函数一样调用
6
>>> g = lambda x, y=2,z=3: x+y+z   #参数默认值
>>> g(1)
6
>>> g(2, z=4, y=5)                #关键参数
11
```

5.6 lambda表达式

```
>>> L = [(lambda x: x**2), (lambda x: x**3), (lambda x: x**4)]
>>> print(L[0](2),L[1](2),L[2](2))
4 8 16
>>> D = {'f1':(lambda:2+3), 'f2':(lambda:2*3),
       'f3':(lambda:2**3)}
>>> print(D['f1'](), D['f2'](), D['f3']())
5 6 8
>>> L = [1,2,3,4,5]
>>> print(list(map(lambda x: x+10, L)))        #模拟向量运算
[11, 12, 13, 14, 15]
>>> L
[1, 2, 3, 4, 5]
```

5.6 lambda表达式

```
>>> def demo(n):
        return n*n

>>> demo(5)
25
>>> a_list = [1,2,3,4,5]
>>> list(map(lambda x: demo(x), a_list))  #在lambda表达式中调用函数
[1, 4, 9, 16, 25]
```

5.6 lambda表达式

```
>>> data = list(range(20))          #创建列表
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> import random
>>> random.shuffle(data)            #打乱顺序
>>> data
[4, 3, 11, 13, 12, 15, 9, 2, 10, 6, 19, 18, 14, 8, 0, 7, 5, 17, 1, 16]
>>> data.sort(key=lambda x: x)      #和不指定规则效果一样
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

5.6 lambda表达式

```
>>> data.sort(key=lambda x: len(str(x))) #按转换成字符串以后的长度排序
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> data.sort(key=lambda x: len(str(x)), reverse=True)
                                #降序排序
>>> data
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

5.6 lambda表达式

```
>>> import random
>>> x = [[random.randint(1,10) for j in range(5)] for i in range(5)]
                        #使用列表推导式创建列表
                        #包含5个子列表的列表
                        #每个子列表中包含5个1到10之间的随机数
>>> for item in x:
        print(item)

[5, 6, 8, 7, 4]
[1, 5, 3, 9, 4]
[9, 6, 10, 7, 6]
[8, 2, 7, 1, 6]
[1, 7, 5, 3, 5]
```

5.6 lambda表达式

```
>>> y = sorted(x, key=lambda item: (item[1], item[4]))
                    #按子列表中第2个元素升序、第5个元素升序排序
>>> for item in y:
            print(item)

[8, 2, 7, 1, 6]
[1, 5, 3, 9, 4]
[5, 6, 8, 7, 4]
[9, 6, 10, 7, 6]
[1, 7, 5, 3, 5]
```

5.7 案例精选

- 例5-1：编写函数计算圆的面积。

```
from math import pi as PI

def CircleArea(r):
    if isinstance(r, (int,float)):      #确保接收的参数为数值
        return PI*r*r
    else:
        print('You must give me an integer or float as radius.')

print(CircleArea(3))
```

5.7 案例精选

- 例5-2：编写函数，接收任意多个实数，返回一个元组，其中第一个元素为所有参数的平均值，其他元素为所有参数中大于平均值的实数。

```
def demo(*para):
    avg = sum(para)/len(para)
    g = [i for i in para if i>avg]
    return (avg,)+tuple(g)

print(demo(1,2,3,4))
```

5.7 案例精选

- 例5-3：编写函数，接收字符串参数，返回一个元组，其中第一个元素为大写字母个数，第二个元素为小写字母个数。

```
def demo(s):
    result = [0,0]
    for ch in s:
        if 'a'<=ch<='z':
            result[1] += 1
        elif 'A'<=ch<='Z':
            result[0] += 1
    return result

print(demo('aaaabbbbC'))
```

5.7 案例精选

- 例5-4：编写函数，接收包含20个整数的列表lst和一个整数k作为参数，返回新列表。处理规则为：将列表lst中下标k之前的元素逆序，下标k之后的元素逆序，然后将整个列表lst中的所有元素再逆序。

```
def demo(lst,k):
    x = lst[:k]
    x.reverse()
    y = lst[k:]
    y.reverse()
    r = x+y
    r.reverse()
    return r

lst = list(range(1,21))
print(demo(lst,5))
```

5.7 案例精选

- 本例的执行结果实际上是把列表中所有元素循环左移k位。在collections标准库的deque对象已经实现了该功能，直接调用即可。

```
>>> import collections
>>> x = list(range(20))
>>> x = collections.deque(x)
>>> x
deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
>>> x.rotate(-3)
>>> x
deque([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2])
>>> x = list(x)
>>> x
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2]
```

5.7 案例精选

❖ 要是把代码写成这样，会不会眼前一亮呢？

```
>>> def shift(lst, k):
        return lst[k:]+lst[:k]

>>> x = list(range(20))
>>> shift(x, 3)
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2]
>>> shift(x, -3)
[17, 18, 19, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

5.7 案例精选

- 例5-5：编写函数，接收整数参数t，返回斐波那契数列中大于t的第一个数。

```
def demo(t):
    a, b = 1, 1
    while b<t:
        a, b = b, a+b
    else:
        return b

print(demo(50))
```

5.7 案例精选

- 例5-6：编写函数，接收一个包含若干整数的列表参数lst，返回一个元组，其中第一个元素为列表lst中的最小值，其余元素为最小值在列表lst中的下标。

```
import random

def demo(lst):
    m = min(lst)
    result = (m,)
    for index, value in enumerate(lst):
        if value==m:
            result = result+(index,)
    return result

x = [random.randint(1,20) for i in range(50)]
print(x)
print(demo(x))
```

5.7 案例精选

- 例5-7：编写函数，接收一个整数t为参数，打印杨辉三角前t行。

```
def demo(t):
    print([1])
    print([1,1])
    line = [1,1]
    for i in range(2,t):
        r = []
        for j in range(0,len(line)-1):
            r.append(line[j]+line[j+1])
        line = [1]+r+[1]
        print(line)

demo(10)
```

5.7 案例精选

- 例5-8：编写函数，接收一个正偶数为参数，输出两个素数，并且这两个素数之和等于原来的正偶数。如果存在多组符合条件的素数，则全部输出。

5.7 案例精选

```
import math

def IsPrime(n):
    m = int(math.sqrt(n))+1
    for i in range(2, m):
        if n%i==0:
            return False
    return True

def demo(n):
    if isinstance(n,int) and n>0 and n%2==0:
        for i in range(3, n//2 +1):
            if IsPrime(i) and IsPrime(n-i):
                print(i, '+', n-i, '=', n)

demo(60)
```

5.7 案例精选

- 例9：编写函数，接收两个正整数作为参数，返回一个数组，其中第一个元素为最大公约数，第二个元素为最小公倍数。

```
def demo(m,n):
    if m>n:
        m, n = n, m
    p = m*n
    while m!=0:
        r = n%m
        n = m
        m = r
    return (n, p//n)

print(demo(20,30))
```

5.7 案例精选

- Python标准库已经提供了计算最大公约数的方法。

```
>>> import fractions
>>> fractions.gcd(36, 39)
3
>>> fractions.gcd(30, 20)
10
>>> 30*20/fractions.gcd(30, 20)
60.0

>>> import math
>>> math.gcd(36, 39)
3
```

5.7 案例精选

- 例5-10：编写函数，接收一个所有元素值都不相等的整数列表x和一个整数n，要求将值为n的元素作为支点，将列表中所有值小于n的元素全部放到n的前面，所有值大于n的元素放到n的后面。

5.7 案例精选

```
import random
def demo(x, n):
    if n not in x:
        print(n, ' is not an element of ', x)
        return
    i = x.index(n) #获取指定元素在列表中的索引
    x[0], x[i] = x[i], x[0] #将指定元素与第0个元素交换
    key = x[0]
    i = 0
    j = len(x) - 1
    while i<j:
        while i<j and x[j]>=key: #从后向前寻找第一个比指定元素小的元素
            j -= 1
        x[i] = x[j]
        while i<j and x[i]<=key: #从前向后寻找第一个比指定元素大的元素
            i += 1
        x[j] = x[i]
    x[i] = key
x =list(range(1, 10))
random.shuffle(x)
print(x)
demo(x, 4)
print(x)
```

5.7 案例精选

- 上面的代码演示的是快速排序算法中非常重要的一个步骤，当然也可以使用下面更加简洁的代码来实现。

```
>>> import random
>>> def demo(x, n):
        t1 = [i for i in x if i<n]
        t2 = [i for i in x if i>n]
        return t1+[n]+t2
>>> x = list(range(1,10))
>>> random.shuffle(x)
>>> x
[1, 9, 3, 6, 5, 2, 4, 7, 8]
>>> demo(x, 4)
[1, 3, 2, 4, 9, 6, 5, 7, 8]
```

5.7 案例精选

- 例5-11 编写函数，计算字符串匹配的准确率。以打字练习程序为例，假设origin为原始内容，userInput为用户输入的内容，下面的代码用来测试用户输入的准确率。

5.7 案例精选

```
def Rate(origin, userInput):
    if not (isinstance(origin, str) and isinstance(userInput, str)):
        print('The two parameters must be strings.')
        return
    if len(origin)<len(userInput):
        print('Sorry. I suppose the second parameter string is shorter.')
        return
    right = 0                    #精确匹配的字符个数
    for origin_char, user_char in zip(origin, userInput):
        if origin_char==user_char:
            right += 1
    return right/len(origin)

origin = 'Shandong Institute of Business and Technology'
userInput = 'ShanDong institute of business and technolog'
print(Rate(origin, userInput))   #输出测试结果
```

5.7 案例精选

- 例5-12 编写函数，使用非递归方法对整数进行因数分解。

```
from random import randint
from math import sqrt
```

5.7 案例精选

```
def factoring(n):
    '''对大数进行因数分解'''
    if not isinstance(n, int):
        print('You must give me an integer')
        return
    #开始分解，把所有因数都添加到result列表中
    result = []
    for p in primes:
        while n!=1:
            if n%p == 0:
                n = n/p
                result.append(p)
            else:
                break
        else:
            result = '*'.join(map(str, result))
            return result
    #考虑参数本身就是素数的情况
    if not result:
        return n
```

5.7 案例精选

```
testData = [randint(10, 100000) for i in range(50)]
#随机数中的最大数
maxData = max(testData)
#小于maxData的所有素数
primes = [ p for p in range(2, maxData) if 0 not in
      [ p% d for d in range(2, int(sqrt(p))+1)] ]

for data in testData:
   r = factoring(data)
   print(data, '=', r)
   #测试分解结果是否正确
   print(data==eval(r))
```

5.7 案例精选

- 例5-11 编写函数模拟猜数游戏。系统随机产生一个数，玩家最多可以猜5次，系统会根据玩家的猜测进行提示，玩家则可以根据系统的提示对下一次的猜测进行适当调整。

5.7 案例精选

```
from random import randint

def guess(maxValue=100, maxTimes=5):
    value = randint(1,maxValue)    #随机生成一个整数
    for i in range(maxTimes):
        prompt = 'Start to GUESS:' if i==0 else 'Guess again:'
        #使用异常处理结构，防止输入不是数字的情况
        try:
            x = int(input(prompt))
        except:
            print('Must input an integer between 1 and ', maxValue)
        else:
            if x == value:                #猜对了
                print('Congratulations!')
                break
            elif x > value:
                print('Too big')
            else:
                print('Too little')
    else:                                 #次数用完还没猜对，游戏结束，提示正确答案
        print('Game over. FAIL.')
        print('The value is ', value)
```

5.7 案例精选

例5-12 编写函数，计算形式如a + aa + aaa + aaaa + ... + aaa...aaa的表达式的值，其中a为小于10的自然数。

```
def demo(v, n):
    assert type(n)==int and 0<v<10, 'v must be integer between 1 and 9'
    result, t = 0, 0
    for i in range(n):
        t = t*10 + v
        result += t
    return result

print(demo(3, 4))
```

5.7 案例精选

- 例5-13 编写函数，模拟轮盘抽奖游戏。

✓ 轮盘抽奖是比较常见的一种游戏，在轮盘上有一个指针和一些不同颜色、不同面积的扇形，用力转动轮盘，轮盘慢慢停下后依靠指针所处的位置来判定是否中奖以及奖项等级。本例中的函数名和很多变量名使用了中文，这在Python 3.x中是完全允许的。

5.7 案例精选

```
from random import random

def 轮盘赌(奖项分布):
    本次转盘读数 = random()
    for k, v in 奖项分布.items():
        if v[0]<=本次转盘读数<v[1]:
            return k
#各奖项在轮盘上所占比例
奖项分布 = {'一等奖':(0, 0.08),
          '二等奖':(0.08, 0.3),
          '三等奖':(0.3, 1.0)}

中奖情况 = dict()

for i in range(10000):
    本次战况 = 轮盘赌(奖项分布)
    中奖情况[本次战况] = 中奖情况.get(本次战况, 0) + 1

for item in 中奖情况.items():
    print(item)
```