第5章 函数

第5章 函数

- 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。

5.1.1 函数定义与调用基本语法

❖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

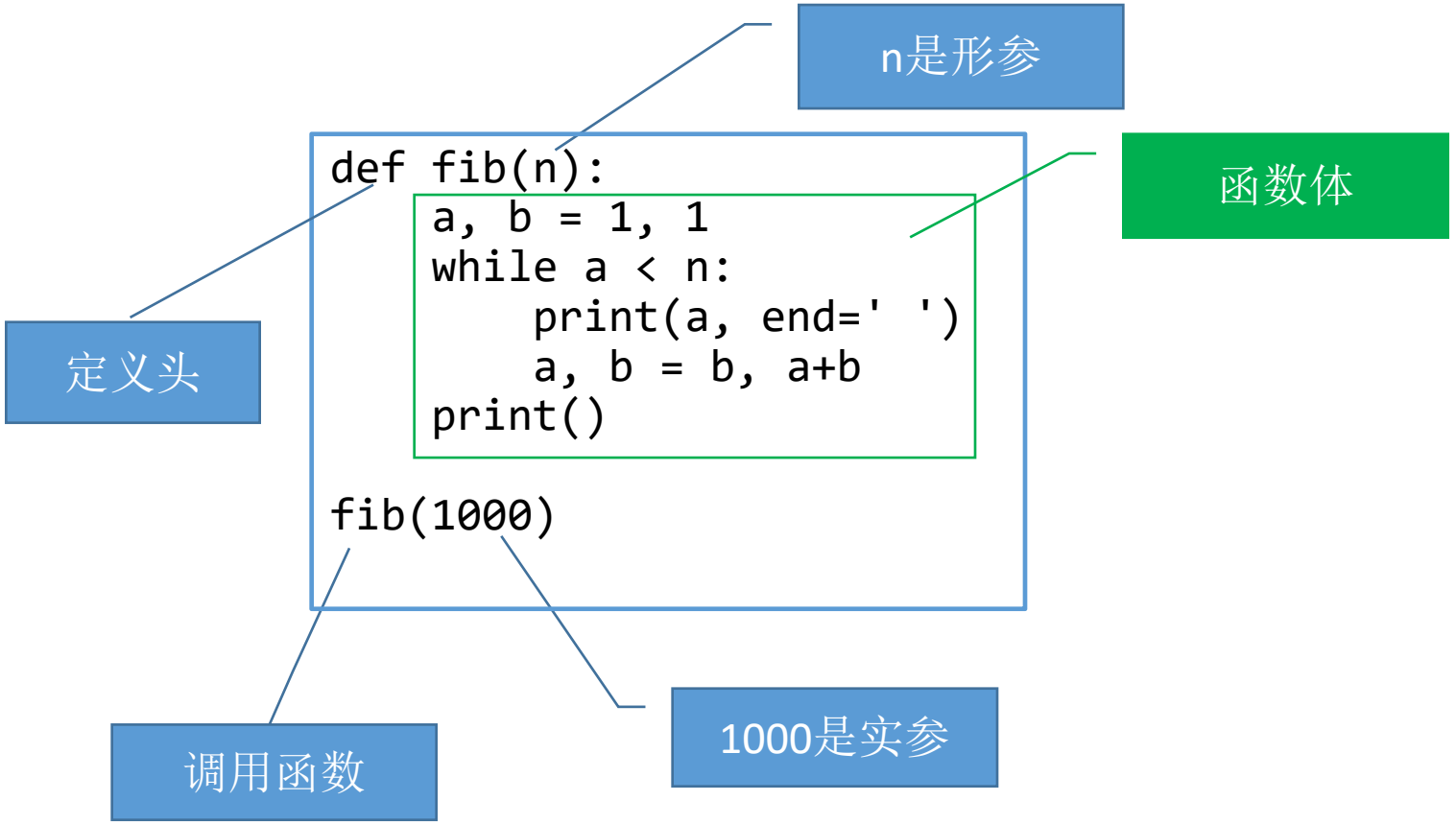
✓函数体相对于def关键字必须保持一定的空格缩进

5.1.1 函数定义与调用基本语法

- 在Python中，定义函数时也不需要声明函数的返回值类型，而是使用return语句结束函数执行的同时返回任意类型的值，函数返回值类型与return语句返回表达式的类型一致。
- 不论return语句出现在函数的什么位置，一旦得到执行将直接结束函数的执行。
- 如果函数没有return语句、有return语句但是没有执行到或者执行了不返回任何值的return语句，解释器都会认为该函数以return None结束，即返回空值。

5.1.1 函数定义与调用基本语法

- **问题解决：**编写生成斐波那契数列的函数并调用。



5.1.1 函数定义与调用基本语法

- 在定义函数时，开头部分的注释并不是必需的，但如果为函数的定义加上注释的话，可以为用户提供友好的提示。

```
>>> 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.
```

5.2 函数参数

- 函数定义时圆括弧内是使用逗号分隔开的形参列表（parameters），函数可以有多个参数，也可以没有参数，但定义和调用时一对圆括弧必须要有，表示这是一个函数并且不接收参数。
- 调用函数时向其传递实参（arguments），根据不同的参数类型，将实参的引用传递给形参。
- 定义函数时不需要声明参数类型，解释器会根据实参的类型自动推断形参类型，在一定程度上类似于函数重载和泛型函数的功能。

5.2 函数参数

- 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，而是创建一个新变量。例如：

```
>>> def addOne(a):
    print(id(a), ':', a)
    a += 1
    print(id(a), ':', a)

>>> v = 3
>>> id(v)
1599055008
>>> addOne(v)
1599055008 : 3
1599055040 : 4
>>> v
3
>>> id(v)
1599055008
```

注意：此时a的地址与v的地址相同

现在a的地址和v的地址不一样了

5.2 函数参数

- 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。

```
>>> def modify(v):                  # 使用下标修改列表元素值
    v[0] = v[0]+1
>>> a = [2]
>>> modify(a)
>>> a
[3]
>>> def modify(v, item):            # 使用列表的方法为列表增加元素
    v.append(item)
>>> a = [2]
>>> modify(a,3)
>>> a
[2, 3]
```

5.2 函数参数

- 也就是说，如果传递给函数的实参是可变序列，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，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'}
```

5.2.1 位置参数

- 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
```

5.2.2 默认值参数

- 在调用带有默认值参数的函数时，可以不用为设置了默认值的形参进行传值，此时函数将会直接使用函数定义时设置的默认值
- 当然也可以通过显式赋值来替换其默认值。在调用函数时是否为默认值参数传递实参是可选的。
- 需要注意的是，在定义带有默认值参数的函数时，任何一个默认值参数右边都不能再出现没有默认值的普通位置参数，否则会提示语法错误。

5.2.2 默认值参数

- 带有默认值参数的函数定义语法如下：

```
def 函数名(……，形参名=默认值):
    函数体
```

5.2.2 默认值参数

- 可以使用“函数名.__defaults__”随时查看函数所有默认值参数的当前值，其返回值为一个元组，其中的元素依次表示每个默认值参数的当前值。

```
>>> def say( message, times =1 ):
    print((message+' ') * times)
>>> print(say.__defaults__)
(1,)
```

5.2.2 默认值参数

- 多次调用函数并且不为默认值参数传递值时，默认值参数只在定义时进行一次解释和初始化，对于列表、字典这样可变类型的默认值参数，这一点可能会导致很严重的逻辑错误。例如：

```
>>> def demo(newitem, old_list=[]):
    old_list.append(newitem)
    return old_list

>>> print(demo('5', [1, 2, 3, 4]))
[1, 2, 3, 4, '5']
>>> print(demo('aaa', ['a', 'b']))
['a', 'b', 'aaa']
>>> print(demo('a'))
['a']
>>> print(demo('b'))                    #注意这里的输出结果
['a', 'b']
```

5.2.2 默认值参数

- 一般来说，要避免使用列表、字典、集合或其他可变序列作为函数参数默认值，对于上面的函数，更建议使用下面的写法。

```
def demo(newitem, old_list=None):
    if old_list is None:
        old_list = []
    old_list.append(newitem)
    return old_list
```

5.2.2 默认值参数

- 函数的默认值参数是在函数定义时确定值的，所以只会被初始化一次。

```
>>> 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
>>> i = 7
>>> f()
3
>>> def f(n=i):                    #重新定义函数
    print(n)
>>> f()
7
```

5.2.3 关键参数

- 关键参数主要指调用函数时的参数传递方式，与函数定义无关。通过关键参数可以按参数名字传递值，明确指定哪个值传递给哪个参数，实参顺序可以和形参顺序不一致，但不影响参数值的传递结果，避免了用户需要牢记参数位置和顺序的麻烦，使得函数的调用和参数传递更加灵活方便。

```
>>> 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
```

5.2.4 可变长度参数

- 可变长度参数主要有两种形式：在参数名前加1个*或2个**
  - *parameter用来接受多个位置参数并将其放在一个元组中
  - **parameter接受多个关键参数并存放到字典中

5.2.4 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)
```

5.2.4 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)
```

5.2.4 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 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。
- 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。

5.3 变量作用域

- 全局变量可以通过关键字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，可以在函数内使用global将其声明为全局变量。

✓ 如果一个变量在函数外没有定义，在函数内部也可以直接将一个变量定义为全局变量，该函数执行后，将增加一个新的全局变量。

5.3 变量作用域

```
>>> 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
```

5.3 变量作用域

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

5.3 变量作用域

- 注意：在某个作用域内任意位置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
```

5.3 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
```

5.4 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。也可以定义具名函数。

- 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，不允许包含复合语句，但在表达式中可以调用其他函数。

5.4 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
```

5.4 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]
```

5.4 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]
```

5.4 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]
```

5.4 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]
```

5.4 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]
```

5.4 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]
```

5.6 精彩案例赏析

示例5-1 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)                #平均值
    g = [i for i in para if i>avg]             #列表推导式
    return (avg,) + tuple(g)
```

5.6 精彩案例赏析

示例5-2 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 ch.islower():    #islower参见P162
            result[1] += 1
        elif ch.isupper():          #isupper参见P162
            result[0] += 1
    return tuple(result)
```

5.6 精彩案例赏析

示例5-3 编写函数，接收包含n个整数的列表lst和一个整数k（0<=k<n）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中的所有元素逆序。

```
def demo(lst, k):
    x = lst[k-1::-1]
    y = lst[:k-1:-1]
    return list(reversed(x+y))
```

5.6 精彩案例赏析

示例5-4 编写函数，接收一个整数t为参数，打印杨辉三角前t行。

```
def yanghui(t):
    print([1])
    line = [1, 1]
    print(line)
    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)
```

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
```

5.6 精彩案例赏析

- **示例5-6** 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。

```
def demo(n):
    def IsPrime(p):
        if p == 2:
            return True
        if p%2 == 0:
            return False
        for i in range(3, int(p**0.5)+1, 2):
            if p%i==0:
                return False
        return True

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

5.6 精彩案例赏析

示例5-7 编写函数，接收两个正整数作为参数，返回一个元组，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):
    p = m*n
    while m%n!=0:
        m, n = n, m%n
    return (n, p//n)
```

```
def demo(m,n):
    p=min(m,n)
    while m%p!=0 or n%p!=0:
        p=p-1
    q=max(m,n)
    while q%m!=0 or q%n!=0:
        q=q+1
    return p,q


print(demo(24,36))
```

5.6 精彩案例赏析

示例5-8 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。

```
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
```

5.6 精彩案例赏析

- **示例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。

5.6 精彩案例赏析

```
from random import randint

def guess(maxValue=100, maxTimes=5):
    #随机生成一个整数
    value = randint(1,maxValue)
    for i in range(maxTimes):
        prompt = 'Start to GUESS:' if i==0 else 'Guess again:'
       x = int(input(prompt))
       if x == value:
              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)
```

5.6 精彩案例赏析

示例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):
    result, t = 0, 0
    for i in range(n):
        t = t*10 + v
        result += t
    return result

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

5.6 精彩案例赏析

示例5-17 编写函数，查找序列元素的最大值和最小值。给定一个序列，返回一个元组，其中元组第一个元素为序列最大值，第二个元素为序列最小值。

```
def myMaxMin(iterable):
    '''返回序列的最大值和最小值'''
    tMax = tMin = iterable[0]
    for item in iterable[1:]:
        if item > tMax:
            tMax = item
        elif item < tMin:
            tMin = item

    return (tMax, tMin)
```