第六章函数概述

本文是Python通用编程系列教程，已全部更新完成，实现的目标是从零基础开始到精通Python编程语言。本教程不是对Python的内容进行泛泛而谈，而是精细化，深入化的讲解，共5个阶段，25章内容。所以，需要有耐心的学习，才能真正有所收获。虽不涉及任何框架的使用，但是会对操作系统和网络通信进行全局的讲解，甚至会对一些开源模块和服务器进行重写。学完之后，你所收获的不仅仅是精通一门Python编程语言，而且具备快速学习其他编程语言的能力，无障碍阅读所有Python源码的能力和对计算机与网络的全面认识。对于零基础的小白来说，是入门计算机领域并精通一门编程语言的绝佳教材。对于有一定Python基础的童鞋，相信这套教程会让你的水平更上一层楼。

一 函数介绍

1. 函数的由来

基于第一阶段的项目，我们可以发现存以下几个问题，函数就是用来解决这些问题的。

1. 程序组织结构不清晰，可读性差
2. 代码冗余
3. 管理维护的难度极大，扩展性

2. 函数的定义与调用说明

需要注意的是，函数名本质和变量类似(打印一个变量你直接看到的结果是变量的值，这是龟叔在内部做了转化，为了让你看的更直观，打印函数你直接看到的结果是一个内存地址，从底层上讲，变量名与函数名其实都是与内存地址对应的，因为定义的过程就是在开辟内存空间)，所以函数名定义规则与定义变量名一致。函数就像是一个功能，这个功能就是要执行一个动作，所以约定俗成写成动词或者动词词组。

- 具备某一个功能的工具就是程序的中函数
- 事先准备工具的过程就是函数的定义
- 把准备好的工具拿来就用即为函数的调用

> 所以函数的使用必须遵循：先定义，再调用

二 函数的定义

1. 函数定义说明

```
def 函数名(参数1,参数2,...):
    """
    文档注释
    """
    代码块1
    代码块2
```

```
    代码块2
    return 返回值

# def:定义函数的关键字
# 函数名：是用来调用函数的
# 函数名的命名必须能反映出函数的功能
# 文档注释：描述该函数，来增强函数的可读性
# 代码块：函数的功能实现代码
# return：函数的返回值
```

有了函数之后，如需实现以下打印功能，我们可以使用函数来完成。

```
# print('=========================')
# print('hello Albert')
# print('=========================')

# 1 先定义
def print_sym(sym, count):  # print_sym=<function print_msg at
0x000001B2A33698C8>
    print(sym * count)


def print_msg(msg):
    print('\033[045m%s\033[0m' % msg)


# 2 再调用（函数名加括号就是在调用函数）
print(print_sym)  # 函数名对应一个内存地址
print_sym('#', 30)
print_msg('hello Albert')
print_sym('#', 30)

# 关于函数的参数，现在只需要知道定义的时候有几个参数，调用的时候就传入几个参数。
```

2. 定义函数的三种形式：

```
# 1 有参函数：当函数体的功能依赖于传入的参数时，我们就使用有参函数
def max2(x, y):  # x=100,y=101

    if x > y:
```

```
        print(x)
    else:
        print(y)


max2(100, 101)

# 2 无参函数：当函数体的功能不使用传入的参数时，我们传入参数显然是没有必要的
def func():
    print('----------------------')
    print('---soft run-----------')
    print('----------------------')


def interact():
    name = input('username>>: ').strip()
    pwd = input('password>>: ').strip()
    print(name, pwd)


interact()  # 定义时无参，意味着调用时也无须传入参数
func()  # 定义时无参，意味着调用时也无须传入参数


# 3 空函数:函数体为pass，事先定义功能组织结构，通过调用函数执行某个功能
def auth(username, password):
    """
    这是一个用户认证功能，在Pycharm中，当输入三引号回撤之后，下面的三行代码自动出现
    :param username:
    :param password:
    :return:
    """

def put():
    """
    上传功能
    :return:
    """
    pass


def get():
    """
    下载功能
    :return:
```

```
    """
    pass


def ls():
    """
    list contents
    :return:
    """
    pass
```

三 函数的调用

1. 函数调用说明

```
# 函数的使用必须遵循：先定义，后调用的原则
# 注意：如果没有事先定义函数而直接调用，就相当于在引用一个不存在的变量名
# 定义阶段：在定义阶段只检测语法，不执行函数体代码
# 调用阶段：根据函数名找到函数的内存地址，然后执行函数体代码

# 函数名加括号即调用函数

# 定义阶段
def foo():
    print('from foo')
    bar()


def bar():
    print('from bar')


# 调用阶段
foo()

"""
会报错
# 定义阶段
def foo():
    print('from foo')
    bar()
```

```
# 调用阶段
foo()


def bar():
    print('from bar')
"""
```

2. 调用函数的三种形式

```
# 1 基本的调用
def func():
    print('from func')


func()


# 2 调用并把返回结果赋值给变量
def max2(x, y):
    if x > y:
        return x
    else:
        return y


res = max2(10, 3)
print(res)
res = max2(10, 3) * 100  # 和上面类似，对返回结果再计算
print(res)

# 4 把返回结果再当做参数传入
res = max2(max2(10, 3), 11)
print(res)
```

四 函数的返回值

1. 函数返回值说明

```
# 什么时候应该有返回值？
# 函数体代码运行完毕后需要有一个返回结果给调用者
```

2. 函数返回值的三种形式

```
# 1 没有return，或者return后面什么都不写，返回值None
def func():
    pass


def func1():
    return


def func2():
    return None


res = func()
res1 = func1()
res2 = func2()
print(res)
print(res1)
print(res2)


# 2 return后跟一个值,返回该值本身
def func3():
    return 1


res3 = func3()
print(res3)


# 3 return可以逗号分隔，返回多个值,会返回一个元组给调用者
def func4():
    return 1, 2, [1, 2, 3]
```

```
res4 = func4()
print(res4)  # (1, 2, [1, 2, 3])
```

3. return两点注意事项

1. return返回值没有类型限制
2. return是函数结束的标志，函数内可以写多个return，但执行一次，函数就立刻结束，并把return后的值作为本次调用的返回值

```
def func5():
    print('first')
    return 1
    print('second')
    return 2
    print('third')
    return 3


res5 = func5()
print(res5)
```

五 函数的参数

1. 形参与实参

```
"""
形参（形式参数）：指的是在定义函数时，括号内定义的参数，形参其实就变量名
实参（实际参数）：指的是在调用函数时，括号内传入的值，实参其实就是变量的值
"""
# x,y是形参
def func(x, y):  # x=10, y=11
    print(x)
    print(y)


# 10,11是实参
```

```
func(10, 11)

"""
注意：
实参值（变量的值）与形参（变量名）的绑定关系只在函数调用时才会生效/绑定
在函数调用结束后就立刻解除绑定
"""
```

2. 有参函数的五种传参方式

(1) 位置参数

以上所讲的形参与实参是有参函数的两个概念，接下是传参方式，位置参数就是最基本的传参方式。位置即顺序，位置参数指的就是按照从左到右的顺序依次定义的参数。

```
# 在定义函数时，按照位置定义的形参，称为位置形参
def foo(x, y, z):
    print(x, y, z)


"""
注意：
位置形参的特性是：在调用函数时必须为其传值，而且多一个不行，少一个也不行
"""

# 在调用函数时，按照位置定义的实参，称为位置实参

# foo(1,2) # 报错
# foo(1,2,3,4) #报错
foo(1, 3, 2)  # x = 1, y = 3, z = 2
"""
注意：位置实参会与形参一一对应
"""
```

(2) 关键字参数

在调用函数时，按照key=value的形式定义的实参，称为关键字参数。关键字参数是指在位置形参的前提下，以关键字的形式为形参传值，所以它与位置参数的区别主要是体现在实参的传值上面。

```
def foo(x, y, z):
    print(x, y, z)


"""
注意：
1 相当于直呼其名地为形参传值，意味着即便是不按照顺序定义，仍然能为指定的参数传值
foo(2,1,3)   # x=2,y=1,z=3
foo(y=2,x=1,z=3)   # x=1,y=2,z=3

2 在调用函数时，位置实参与关键字实参可以混合使用，但必须遵循形参的规则
foo(1,z=3)   # 报错

3 不能为同一个形参重复传值
foo(1,x=1,y=3,z=2)   # 报错

4 位置实参必须放到关键字实参的前面
foo(y=3,z=2,1)   # 报错
"""

foo(1, z=3, y=2)
```

(3) 默认参数

到目前形参只讲了一种就是位置形参，实参讲了两种分别是位置实参和关键字实参，接下来我们再来讲解一种形参，叫做默认参数。它指的是在定义阶段已经为某个形参赋值,那么该形参就称为默认参数。

```
# 1 定义阶段已经有值,意味着调用阶段可以不传值
def register(name, age, sex='male'):
    print(name, age, sex)


register('Albert', 18, )
register('James', 34, )
register('林志玲', 20, 'female')
register('周星驰', 50)

# 2 位置形参必须在默认参数的前面
# def func(y=1,x): #报错
#     pass
```

```
# 3 默认参数的值只在定义阶段赋值一次,也就是说默认参数的值再定义阶段就固定死了
m = 10


def foo(x, y=m):
    print(x, y)


m = 'a'  # foo内的默认参数不会发生改变
foo(1)
foo(1, 11)


# 4 默认参数的值应该设置为不可变类型(重要)

# 假如默认参数不是不可变类型，我们以列表为例
def register(name, hobby, l=[]):
    l.append(hobby)
    print(name, l)


register('Kobe', 'play')  # Kobe ['play'] 一切正常~
register('James', 'read')  # James ['play', 'read'] what?!
register('Albert', 'music')  # Albert ['play', 'read', 'music'] 这就是未设置为不可变类型出现的BUG


# 数据出错的原因就是每次调用都会在同一个列表上作修改


# 为了实现同样的功能，修正后如下
def register(name, hobby, l=None):
    if l is None:
        l = []
    l.append(hobby)
    print(name, l)


register('Kobe', 'play')
register('James', 'read')
register('Albert', 'music')

# 应用场景:
# 对于经常需要变化的值,需要将对应的形参定义成位置形参
# 对于大多数情况值都一样的情况,需要将对应的形参定义成默认参数
```

(4) 可变长参数

<1> 可变长参数基本使用

可变长度指的参数的个数可以不固定，实参有按位置定义的实参和按关键字定义的实参,所以可变长的实参指的就是按照这两种形式定义的实参个数可以不固定，然而实参终究是要给形参传值的，所以形参必须有两种对应的解决方案来分别处理以上两种形式可变长度的实参。

```
# *会将溢出的位置实参全部接收,然后保存成元组的形式赋值给一个变量args(可以任意命名，约定俗成args)
def foo(x, y, z, *args):  # args=(4,5,6,7,8)
    print(x, y, z)
    print(args)


foo(1, 2, 3, 4, 5, 6, 7, 8, )


# **会将溢出的关键字实参全部接收,然后保存成字典的形式赋值给kwargs
def foo(x, y, z, **kwargs):  # kwargs={'c':3,'a':1,'b':2}
    print(x, y, z)
    print(kwargs)


foo(x=1, y=2, z=3, a=1, b=2, c=3)
```

<2> 星与星星(打散)

很多时候【*】的作用就是打散，在讲列表的方法append与extend的区别时，也做了一个简单的说明。

```
# 一旦碰到实参加*,就把该实参的值打散
def foo(x, y, z, *args):  # args=([4,5,6,7,8],)
    print(x, y, z)
    print(args)


foo(1, 2, 3, *[4, 5, 6, 7, 8])  # foo(1,2,3,4,5,6,7,8)
foo(1, 2, 3, *(4, 5, 6, 7, 8))  # foo(1,2,3,4,5,6,7,8)
```

```python
foo(1, 2, 3, *'hello')  # foo(1,2,3,'h','e','l','l','o')


def foo(x, y, z):
    print(x, y, z)


# foo(*[1, 2, 3, 4])  # foo(1,2,3,4) #报错
# foo(*[1, 2, ])  # foo(1,2,) #报错
foo(*[1, 2, 3])  # foo(1,2,3)


# 一旦碰到实参加**,就把该实参的值打散
def bar(x, y, z, **kwargs):
    print(x, y, z)
    print(kwargs)


bar(1, 2, 3, **{'a': 1, 'b': 2})  # foo(1,2,3,b=2,a=1)


def boo(x, y, z):
    print(x, y, z)


# boo(1, **{'z': 3, 'y': 2, 'x': 111})  # 报错 boo(1,z=3,y=2,x=111)
boo(1, **{'z': 3, 'y': 2})  # foo(1,z=3,y=2)

# *的应用场景
def sum2(*args):
    res = 0
    for num in args:
        res += num
    return res


print(sum2(1, 2, 3, 4, 5, 6, 7))


# ** 的应用场景
def auth(name, pwd, **kwargs):
    print(name)
    print(pwd)
    print(kwargs)
```

```
auth(name='Albert', pwd='123')
auth(name='Albert', pwd='123', group='group1')
```

<3> 组合使用(重点)

```
def index(name, age, gender):
    print('welcome %s %s %s' % (name, age, gender))


def wrapper(*args, **kwargs):  # args=(1,2,3),kwargs={'x':1,'y':2,'z':3}
    # print(args)
    # print(kwargs)
    index(*args, **kwargs)  # index(*(1,2,3),**{'x':1,'y':2,'z':3}) #
index(1,2,3,z=3,y=2,x=2)


# wrapper(1,2,3,x=1,y=2,z=3)  # 报错

wrapper(name='Albert',age=18,gender='male')
wrapper('Albert', age=18, gender='male')
wrapper('Albert', 18, gender='male')
wrapper('Albert', 18, 'male')

"""
执行过程：
wrapper的所有参数都原封不动地传给index，而index函数只接收三个位置参数
星与星星的组合使用在源码中非常常见，这也是装饰器的核心之一，这非常重要。
"""
```

(5) 命名关键字参数

<1> 命名关键字参数导入

在【星】后面参数都是命名关键字参数，它的特点是必须被传值，约束函数的调用者必须按照key=value的形式传值，约束函数的调用者必须用指定的key名。
如果没有明明关键字参数，当我们需要在做上述约束时，应该按照如下代码操作。

```
def auth(*args, **kwargs):
    """
    使用方式auth(name="Albert",pwd="123")
    :param args:
    :param kwargs:
    :return:
    """
    if len(args) != 0:
        print('必须用关键字的形式传参')
        return
    if 'name' not in kwargs:
        print('必须用指定的key名name')
        return
    if 'pwd' not in kwargs:
        print('必须用指定的key名pwd')
        return

    name = kwargs['name']
    pwd = kwargs['pwd']
    print(name, pwd)


print(help(auth))  # 打印文档注释

auth(x='Albert', y='123')
auth('Albert', '123')
auth('Albert', pwd='123')
auth(name='Albert', pwd='123')  # 约束函数的调用者必须用key=value的形式传值
```

<2> 命名关键字参数使用

```
# 使用命名关键字参数
def foo(x, y, *, z):
    print(x, y, z)


# foo(1,2)  # 报错
# foo(1,2,3)  # 报错
# foo(1,2,a=3)  # 报错
foo(1, 2, z=3)
```

```python
# 其实命名关键字参数的核心是 *，args只是一个变量，有或者没有并不影响
def auth(*args, name, pwd):
    print(name, pwd)


auth(pwd='123', name='Albert')


# 命名关键字参数是硬性限制，但Python的语法风格是约定俗成，不做限制
def register(name, age):
    """
    我们不会在这里添加对name和age的要求限制
    :param name:
    :param age:
    :return:
    """
    print(type(name), type(age))


register(123, [1, 2, 3])


# 使用命名关键字参数之后，可以接收参数的最复杂的情况
def foo(x, y=1, *args, z, m=2, **kwargs):  # m=2是关键字参数的默认值
    pass


# 一般情况下，foo1和foo2这两种就够用了
def foo1(x, y=1):
    pass


def foo2(x, *args, **kwargs):
    pass
```