Python 形式参数与实际参数

形式参数和实际参数

1.在定义函数时函数名后面圆括号中的变量名称叫做形式参数

2.在调用函数时 函数名后面圆括号中的变量名称叫实际参数

几种参数的类型：

位置参数：在默认情况下 参数是通过其位置进行匹配的 从左到右而且必须精确的传递和函数头部参数名一样多的参数 这是位置参数 还可以定义关键字参数， 默认参数等

关键字参数： 调用者可以定义哪一个函数接受这个值 通过关键字进行匹配 在调用过程中使用name = value这种语法

默认参数： 为没有传入值的参数定义参数值，在创建函数的时候使用name=value

可变参数： 收集任意多基于位置或关键字的参数 通常以 * 开头

1） 收集所有的位置参数组成一个元组 如 *args

2） 收集所有的关键字参数组成一个字典 如 **kwargs

可变参数解包： 传递任意多基于位置或关键字的参数，调用者能够再使用 * 语法将参数集合打散，这个与在函数头部的 * 恰好相反，在函数头部意味着收集任意多的参数，在调用者中意味着传递任意多参数

匹配语法

语法	位置	解释
func(value)	调用者	常规参数: 通过位置进行匹配
func(name=value)	调用者	关键字参数 通过变量名匹配
func(*sequence)	调用者	解包位置参数
func(**dict)	调用者	解包关键字参数

在函数的调用过程中 可以通过变量名进行位置匹配 使用name=value的形式进行关键字匹配

例：

```
def f1(a, b, c):       #函数需要3个参数
    print a, b, c
f1(*(1, 2, 3))          #传入一个具有相应多个元素的元组

def f2(a, b, c):       ##函数需要3个参数
    print a, b, c
f2(**{'a': 1, 'b': 2, 'c': 5})     ##传递有3个Key的字典
```

#使用 *args **kwargs 允许我们在一个序列或字典中相应的封装任意多位置相关或者关键字对象 并且将他们传递给函数的时候 将他们解包为分开的单个的参数

def func(name) 函数 常规参数 通过位置或者变量名匹配

def func(name=value) 函数 定义默认参数

def func(*name) 函数 捕获位置参数

def func(**name) 函数 捕获关键字参数

def func(*args, **kargs): 函数 捕获位置参数 与关键字参数

#在函数头部 通过位置或变量名进行参数匹配 使用name=value的形式定义参数的默认值

例:

```
def f3(*args):
    print args
f3(1, 2, 3, 4)  ##将多个位置参数捕获成元组

def f4(**kargs):
    print kargs
f4(a=1, b = 2)  ##将多个关键字参数捕获成字典

def f5(*args, **kargs):
```

```
    print args
    print kargs
f5(1, 2, 3, 4, a=1, b = 2, c= 4)
##位置参数被args收集　关键字参数被 kargs收集
def f6(name, age=17, *args, **kargs):
    print name
    print age
    print kargs
    print args
f6('liushuo', 1, 2, 3, 4, score=100, tall=177)
```

*args　收集了任意的额外不匹配的参数到元组中

**kwargs　收集额外的关键字参数到字典中

函数的参数匹配遵循以下模型 否则会报错

任何位置参数 任何关键字参数　*args　**kwargs

习题

以下代码输出的是什么？为什么

```
def func(a, b, c=5):
    print a, b, c
func(1, b=2)
```

```
def func(a, *args):
    print a, args
func(1, 2, 3)

def func(a, **args):
    print a, args
func(a = 1, b = 2, c = 3)

def func(a, b, c = 3, d=4):
    print a, b, c, d
func(1, *(5, 6))
```