# Python学习笔记-函数

## 介绍

当Python运行到def语句时，它将会生成一个新的函数对象并将其赋值给这个函数名，函数名变成了某一个函数的引用，def语句一般格式：

def <name>(arg1, arg2,…argN):

<statement>

def语句是实时执行的，一个def可以出现在任一语句可以出现地方

## 多态

一个操作的意义取决于被操作对象的类型，只要对象支持所预期的接口，函数就能处理。如果限制操作对象的类型，反而不具备动态语言的灵活性，因此我们在Python中为对象编写接口，而不是数据类型

## 作用域

Python创建、改变或查找变量名都是在所谓的命名空间中进行的。在代码中变量名被赋值的位置决定了这个变量名能被访问到的范围。在默认情况下，一个函数的所有变量名都是与函数的命名空间相关联的，这意味着

* 一个在def内定义的变量名能够被def内的代码使用，不能在函数的外部引用这样的变量名
* def之中的变量名与def之外的变量名并不冲突，即使是使用在别处的相同的变量名，一个在def之外被赋值的变量X与这个def之中的赋值的变量X是完全不同的变量

## 传参

参数是通过赋值来进行传递的，Python中所有对象都是通过引用来访问的，所以参数的传递实际上是引用的拷贝，并非实际对象的拷贝。对于不可变对象，比如数字、字符串，不可变对象的参数传递也是通过引用拷贝来传递的，只因不可变对象的不可变性，使得实际效果很像创建了一个拷贝。

## 参数匹配

位置：从左至右进行匹配

关键字参数：通过参数名进行匹配，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，使用name=value语法

可变参数：收集任意多基于位置或关键字的参数，以字符\*开头，收集任意多的额外参数

## 任意参数匹配

在函数的定义中，\*和\*\*是让函数接受任意数目的参数的

* \*args：在元组中收集不匹配的位置参数

示例：

>>>def f(\*args): print(args)

>>>f()

()

>>>f(1)

(1,)

>>>f(1, 2, 3, 4)

(1, 2, 3, 4)

* \*\*args: 将关键字参数转换为一个新的字典

示例：

>>>def f(\*\*args): print(args)

>>>f()

{}

>>>f(a=1, b=2)

{‘a’:1, ‘b’:2}

参数匹配的混搭：

>>>def f(a, \*pargs, \*\*kargs): print(a, pargs, kargs)

>>>f(1, 2, 3, x=1, y=2)

1 (2, 3) {‘x’:1, ‘y’:2}

## 解包参数

在函数的调用时，\*和\*\*表示解包参数中的集合

\*表示在函数调用时将元组按照内部排列顺序拆解成多个参数传递给函数

>>>def func(a, b, c, d): print(a, b, c, d)

>>>args = (1, 2)

>>>args += (3, 4)

>>>func(\*args)

1 2 3 4

\*\*表示在函数调用时将字典拆解成键/值对的形式，再根据关键字参数传递给函数

>>>def func(a, b, c, d): print(a, b, c, d)

>>>args = {‘a’:1, ‘b’:2, ‘c’:3, ‘d’:4}

>>>func(\*\*args)

1 2 3 4

解包参数混搭

>>>func(\*(1, 2), \*\*{‘d’:4, ‘c’:3})

1 2 3 4

>>>func(1,\*(2, 3), \*\*{‘d’:4})

1 2 3 4

>>>func(1, c=3, \*(2,), \*\*{‘d’:4})

1 2 3 4

>>>func(1,\*(2, 3), d=4)

1 2 3 4

>>>func(1, \*(2,), c=3, \*\*{‘d’:4})

1 2 3 4

## Keyword-Only参数

Keyword-only参数编码为命名的参数，出现在参数列表中的\*args之后，所有这些参数都必须在调用中使用关键字语法来传递。

>>>def kwonly(a, \*b, c): print(a, b, c)

>>>kwonly(1, 2, c=3)

1 (2,) 3

>>>kwonly(a=1, c=3)

1 () 3

>>>kwonly(1, 2, 3)

TypeError:…..

>>>def kwonly(a, \*, b, c): print(a, b, c)

>>>kwonly(1, c=3, b=2)

1 2 3

>>>kwonly(1, 2, 3)

TypeError:……

>>>kwonly(1)

TypeError:……

>>>def kwonly(a, \*, b=’spam’, c=’ham’): print(a, b, c)

>>>kwonly(1)

a spam ham

>>>kwonly(1, c=3)

1 spam 3

>>>kwonly(a=1)

1 spam ham

>>>kwonly(c=3, b=2, a=1)

1 2 3

>>>kwonly(1, 2)

TypeError:……

## 注解

从语法上讲，函数注解编写在def头部行，就像与参数和返回值相关的任意表达式一样，对于参数，它们出现在紧随参数名之后的冒号之后；对于返回值，它们编写于紧跟在参数列表之后的一个->之后，注意：注解只在def语句中有效，在lambda表达式中无效。

示例

>>>def func(a: ‘spam’ = 5,b : (1, 10) = 2, c : float = 6) -> int:

return a + b + c

>>>func.\_\_annotations\_\_

{‘a’ : ‘spam’, ‘c’ : <class float> , ‘b’ : (1, 10) }