形式参数（formal parameters） 实参(actual parameters)

🡪

Python函数定义形式为：

def <function\_name>([<parameters>]):

<statement(s)>

|  |  |
| --- | --- |
| def | def为Python定义函数关键字 |
| <function\_name> | 定义的函数名 |
| <parameters> | 由,分开的形式参数申明  特殊的：  参数\*表示在\*以后的参数在函数被调用时，必须以命名参数形式进行传参。  参数/表示在/之前的参数在函数被调用时，必须以定位参数形式进行传参。 |
| : | 表明函数头部申明结束 |
| <statement(s)> | 函数体，含有一条条Python语句 |

🡪

以下总结Python函数传递参数的5种形式

方式1：通过位置参数(positional parameters)，传递参数

例如：

def f(qty, item, price):

print("{0} {1} cost ${2}".format(qty, item, str(price)))

f(6, 'bananas', 1.74)

f(6, 'bananas') # 非法，少于规定位置参数数量  
f(6, 'bananas', 1.74, 'kumquats') # 非法，多于规定位置参数数量

方式2：通过命名参数(keyword parameters)，传递参数

def f(qty, item, price):

print("{0} {1} cost ${2}".format(qty, item, str(price)))

f(qty=6, item='bananas', price=1.74)# 通过命名参数传递参数  
f(qty=6, item='bananas', cost=1.74)# 非法，传递cost未指定的命名参数  
f(item='bananas', price=1.74, qty=6)# 合法，通过命名参数传递，顺序可以于定义不同  
f(qty=6, item='bananas') # 非法，传递的参数少于要求的参数  
f(6, price=1.74, item='bananas') # 合法，可以位置参数和命名参数共同传递  
f(6, 'bananas', price=1.74) # 合法，可位置参数和命名参数共同传递  
f(6, item='bananas', 1.74) # 非法， 函数调用中，一旦使用命名参数以后，后面的参数传递不能使用位置参数形式

方式三：默认参数(default/optional parameters)

def f(qty=6, item='bananas', price=1.74):

print("{0} {1} cost ${2}".format(qty, item, str(price)))

Note:规定默认参数必须在最后申明

Note:任何未传递的默认参数都将使用默认值。

🡪

Python中传入参数A调用函数时，如果传入的是A不可变类型变量，例如int,str,tuple,frozenset。则在函数中无法改变A在主函数中的值，在函数中改变值只会改变在A拷贝上。如果传入的是可变类型的变量A，例如list,dict,set,则在函数中如果直接整体引用改变，则A在主函数中的值不会改变。但如果直接在A上更改某个值，则A在主函数中的值会被改变。

例如：

def f(x):

x = 'foo'  
for i in (  
 40,  
 dict(foo=1, bar=2),  
 {1, 2, 3}, # 申明集合

(1, 2, 3), # 申明元组tuple  
 'bar',  
 ['foo', 'bar', 'baz']):  
 f(i)  
 print(i)  
  
# 40， 非可变类型只会在其拷贝上改变  
# {'foo': 1, 'bar': 2} 可变类型，但x = ‘foo’是整体引用重指定，所以不改变原来的值  
# {1, 2, 3} 非可变类型只会在其拷贝上改变

# (1, 2, 3) 非可变类型只会在其拷贝上改变  
# bar 非可变类型只会在其拷贝上改变  
# ['foo', 'bar', 'baz'] 可变类型，但x = ‘foo’是整体引用重指定，所以不改变原来的值

def f(x):  
 x[0] = '---'  
my\_list = ['foo', 'bar', 'baz', 'qux']  
f(my\_list)  
print(my\_list) # 输出为['---', 'bar', 'baz', 'qux']，my\_list为可变类型list，并且不是整体引用重指定，而是通过x[0]=”..”直接在x上更改值，所以x在主函数中的值会变。

🡪

可变长位置参数\*(Argument Tuple packing/unpacking)

若在函数**定义**头的参数列表中使用\*，则为可变长参数打包(packing)成tuple，从而在函数内参数会放入tuple当中。

例如：

def f(\*args): # 在函数头函数列表申明中，\*args表示可接受变长参数并且放入tuple当中。

虽然可以任意指定参数名，但是一般都以args命名

for x in args:  
 print(x)  
  
f(1, 2, 3) # 输出1， 2， 3.

若在函数**调用时**使用\*，则为一个iterable对象解包(unpacking)，并将参数一个个传递。

例如：

t = ('foo', 'bar', 'baz')

f(\*t) # t为iterable对象tuple, 在调用函数时表示将t解包。与f('foo', 'bar', 'baz')等价

综合例子：

def f(\*args): # 在函数定义时打包，将参数打包放入args中

[print(i) for i in args]  
a = ['foo', 'bar', 'baz', 'qux']   
f(\*a) # 在调用函数时，解包。将参数逐一传入

🡪

可变长命名参数\*\*(Argument Dictionary packing/unpacking)

若在函数**定义**头的参数列表中使用\*\*，则为可变长参数打包(packing)成dictionary，从而在函数内参数会放入dictionary形式的key=value形式当中。

例如：

def f(\*\*kwargs): # 在函数头函数列表申明中，\*\*kargs表示可接受变长参数并且放入dictionary

当中。虽然可以任意指定参数名，但是一般都以kargs命名

for key, val in kwargs.items():  
 print(key, '->', val)  
  
f(foo=1, bar=2, baz=3) # 输出foo->1, bar->2, baz->3

若在在函数**调用时**使用\*\*，则为一个字典对象解包(unpacking)，并将参数以命名参数形式一个个传递。

例如：

d = {'a': 'foo', 'b': 25, 'c': 'qux'}

f(\*\*d) # 相当于f(a='foo', b=25, c='qux')

🡪

可变长位置参数\*(Argument Tuple packing/unpacking)与可变长命名参数\*\*(Argument Dictionary packing/unpacking)

可以合用

例如：

def f(a, b, \*args, \*\*kwargs):

print(F'a = {a}')  
 print(F'b = {b}')  
 print(F'args = {args}')  
 print(F'kwargs = {kwargs}')  
  
f(1, 2, 'foo', 'bar', 'baz', 'qux', x=100, y=200, z=300)

#输出为

a = 1

b = 2

args = ('foo', 'bar', 'baz', 'qux')

kwargs = {'x': 100, 'y': 200, 'z': 300}

🡪 Python3中在函数定义头中，参数\*表示在\*以后的参数在函数被调用时，必须以命名参数形式进行传参。

参数/表示在/之前的参数在函数被调用时，必须以定位参数形式进行传参。

例如：

# This is Python 3.8

def f(x, y, /, z, w, \*, a, b): # /表明形参x,y必须在函数被调用时以定位参数形式进行传参。\*表明形参a,b,必须在函数调用时，以命名参数形式进行传参。  
 print(x, y, z, w, a, b)

f(1, 2, z=3, w=4, a=5, b=6) #合法, 1, 2以定位参数形式给a,b传参。5,6以命名参数方式给a,b进行传参。  
# 1 2 3 4 5 6  
  
f(1, 2, 3, w=4, a=5, b=6) #合法, 1, 2以定位参数形式给a,b传参。5,6以命名参数方式给a,b进行传参。  
# 1 2 3 4 5 6

🡪

在调用函数为函数中的形参kargs赋值时，不可以使用数字作为键，并且键也不能加双引号。虽然一般的字典可以使用数字作为键。

例如:

def Test(a,b,\*\*kargs):

print(a,b,kargs)

Test(1,2,3=”12”) #非法，不可以使用数字作为键

Test(1,2,”p” =”12”) #非法，键也不能加双引号

Test(1,2,p=”12”) # 合法, kargs = {“p” : “12”}

🡪不可以定义函数的时候有两个\*args,\*args1或者两个\*\*kargs,\*\*kargs2的形参。

🡪如果函数申明时候，有形参没有默认值，但是该形参前面有\*args或者\*\*kargs吸收参数，调用函数时必须使用命名参数的形式进行指定，否则报错。

例如：

def Test(\*args,a,b,c=2):

print(args,a,b,c)

Test(1,2,3,4,5) #非法，1,2,3,4,5都被\*args所吸收，导致后面非缺省参数a,b没有赋值，报错

Test(1,2,a=3,c=5,5) #非法，命名实参必须在其他位置实参的后面

Test(1,2,5, a=3,b=5)#合法，首先命名参数在所有位置实参前面，非缺省参数a,b,被赋值。另外由于缺省参数c在\*args之后，所以形参c永远无法被按位置赋值，除非通过命名参数复制。

🡪

Python3总str类的format函数，查看format的内置函数定义为：

def format(self,\*args,\*\*kargs):

所以即使万能参数的函数定义。从而

“{0}+{1}+{2}”.format(1,2,3) #将为输出”1+2+3”

“{A}+{B}+{C}”.format(A=”Hello”,B=”World”,C=”!”) #将为输出”Hello World !”

“{0}+{1}+{C}”.format(”Hello”,”World”,C=”!”) #将为输出”Hello World !”