Python函数的参数

一.形参与实参数

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:

1) 位置参数: 在默认情况下在定义函数时函数的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.

2) 默认参数: 为没有传入值的参数定义参数的值, 在定义函数的时候使用形参使用name=value的形式.

3) 关键字参数: 在调用函数时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.

4) 可变参数: 收集任意多基于位置或关键字的参数

1. 收集所有的位置形参参数组成一个元组 使用*args的形式来收集(其中args是参数名, 程序员可自己随便定义)
2. 收集所有的关键字形参参数组成一个字典 如 **kwargs(其中kwargs是参数名, 程序员可自己随便定义)

5) 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

二.匹配语法

语法 位置 解释

func(value) 调用 常规参数, 通过位置进行匹配

func(name=value) 调用 关键字参数, 通过变量名匹配

func(*sequence) 调用 解包位置参数

func(**dict) 调用 解包关键字参数

def func(name) 定义 常规参数 通过位置或者变量名匹配

def func(name=value) 定义 定义默认参数

def func(*name) 定义 捕获不固定数量位置参数

def func(**name) 定义 捕获不固定数量关键字参数

def func(*args, *kargs) 定义 同时捕获不固定数量位置参数与不固定数量关键字参数关键字参数

In [2]:
```
#示例代码1. 解包不固定数量位置参数

def f1(a, b, c):
    print(a, b, c)

f1(*(1, 2, 3))
```

```
1 2 3
```

In [3]:
```
#示例代码2. 解包不固定数量关键字参数

def f2(a, b, c):
    print(a, b, c)
f2(**{'a': 1, 'b': 2, 'c': 5})
```

```
1 2 5
```

In [4]:
```
#示例代码3. 捕获多个位置参数成为元组

def f3(*args):
    print(args)
f3(1, 2, 3, 4)
```

```
(1, 2, 3, 4)
```

In [6]:
```
#示例代码4.捕获多个关键字参数成为字典
def f4(**kargs):
    print(kargs)
f4(a=1, b = 2)
```

```
{'a': 1, 'b': 2}
```

In [7]:
```
#示例代码5.同时捕获位置参数与关键字参数
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
```

```
(1, 2, 3, 4)
{'a': 1, 'b': 2, 'c': 4}
```

In [8]:
```
#示例代码6.
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)
```

```
liushuo
1
{'score': 100, 'tall': 177}
(2, 3, 4)
```

*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

In [10]:
```
#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)
print('#' * 10)

def func(a, *args):
    print(a, args)
func(1, 2, 3)
print('#' * 10)

def func(a, **args):
    print(a, args)
func(a = 1, b = 2, c = 3)
print('#' * 10)

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

```
1 2 5
##########
1 (2, 3)
##########
1 {'b': 2, 'c': 3}
##########
1 5 6 4
```

In [ ]: