python的函数

基本定义

```
def fun_name():
    # 函数体
```

对于python的函数，最基础的就是要注意**缩进格式**；缩进格式真的很重要，**要么用四个空格，要么用tab，千万不要混用！！！**

除此之外python的函数定义和其他的编程语言十分相似，不过python的函数仍有一些特殊的特性

注意事项

1. 函数要先定义再使用

在调用一个函数之前，我们必须保证这个函数在上文中被定义过；如果调用一个没有被定义过的函数，解释器会因为找不到这个函数而报错；

```
def my_func():
    print("this is a function")

my_func() # 合法调用
```

```
my_func() # 非法调用，my_func()尚未定义

def my_func():
    print("this is a function")
```

2.避免空定义函数

不同于C语言，python不允许空定义，例如：

```
def my_func():  # 这样是非法的
```

如果处于某些原因，真的需要写一个函数又不实现，可以用pass：

```
def my_func():
    pass    # 这样是合法的
```

3.python的函数可以返回多个返回值

不同于c语言一个函数最多只能返回一个值，python可以返回多个不同的返回值：

```
def my_func():
    return 1,2,3

x,y,z = my_func()
# x=1, y=2, z=3
```

4. 使用kwargs传递参数

```
# python支持使用键值对传递参数，python文档将这种方式称为kwargs（即key word args）
def my_function(name, age):
    print("my name is %s",name) # 这里用了格式控制
    print("i am %d years old",age) # 同上

my_function(name = "lifugui", age = 26) # 使用键值对传递的样式传递参数
```

5.传递任意参数

```
# 可变参数使用*声明
def my_func(*args)
    print(args[1])

my_func("hi","hello", 100)
# 这个函数将会输出"hello"
```

需要说明的是：

1. 使用*表明函数的参数是任意参数时，输入的参数会以一个元组（tuple）的形式传递给函数，所以在函数内部我们无法修改参数；

```
>>> def func(*args):
...     print(args[1])
...     args[1]="haha"
...
>>> func("a","b")
b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in func
TypeError: 'tuple' object does not support item assignment
# 可见无法修改
```

2. 使用args[n]的方式可以调用我们想要调用的参数，但是在使用之前最好先确认以下，传递给函数的参数是否拥有args[n]这项；

```
>>> def func(*args):
...     print(args[1])
...     args[1]="haha"
>>> func("a")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in func
IndexError: tuple index out of range # 很明显，我们调用了一个不存在的元素，所以报错
```

6. 全局变量和局部变量

定义在函数中的是局部变量，定义在函数之外的变量是全局变量

```
a = 1 #这是一个全局变量

def my_func():
    print("a in my_func = %d", a) # 我们在函数中仍能访问这个变量

my_func()
```

```
def my_func():
    a = 1   #局部变量
    print(a)

my_func()#这个函数可以输出a的值
print(a)#在函数外调用函数内定义的局部变量会引起错误
```

python不允许局部变量和全局变量同名，如果先定义了全局变量，而后又在函数中定义了同名的局部变量；那么局部变量将会把全局变量隐藏掉，例：

```
a = 1 # 全局变量
print(a) # 可以调用

def my_func():
    a = 2   # a变成了一个局部变量

print(a) # 报错！ 因为此时a不是一个全局变量了，解释器找不到a
```

我们有时候会遇到需要在函数内对全局变量重新赋值的情况，如果直接重新赋值就会将全局变量覆盖，此时我们可以使用global来声明我们正在操作一个全局变量，例：

```
a = 1
print(a) # 输出1

def my_func():
    global a = 2

my_func()
print(a) # 可以调用a，a仍旧是一个全局变量，输出2
```

7. 匿名函数

```
# 格式
# lambda [list]:表达式
# 例:
add = lambda x,y: x+y
print(add(1,2)) # 输出结果为3
```

匿名函数可以简洁的封装一些逻辑，有点像C语言中的内联函数，（其实匿名函数的应用很少...遇到的时候能看懂即可