# 函数

一个规范的函数表达

```
def 函数名(param1,param2……):
    """
    函数功能的描述信息
    :param1：描述
    :param2：描述
    :return：返回值
    """
    code 1
    code 2
    code 3
    ...

    return 返回值
```

## 返回值

return 函数运行到返回值的时候会自动结束

没有return的时候可以返回None

return 可以有多个：只有一个的时候是对应的类型

return 如果返回值是多个 返回值是多个的时候返回的是元组

# 参数

> 形参：定义函数的时候用来表达任何值的参数

> 实参：使用函数的时候输入的真正的参数

**位置形参**

```
def func(x, y):
    print(x)
    print(y)
```

位置实参

```
func(1,2)
```

## 默认形参

在定义的时候就用关键词赋值就可以不用再调用的时候再赋值

## 关键词实参

key = value

> 1. 可以混用位置实参和关键字实参，但是位置实参必须在关键字实参的左边。
> 2. 可以混用位置实参和关键字实参，但不能对一个形参重复赋值。

```
func(x, y=2)
func(y=2, x)  # SyntaxError: positional argument follows keyword argument
func(x, x=1)  # NameError: name 'x' is not defined
```

## 可变长形参 *

```
def sun_self(*args)#args是约定俗成的
```

*号会接受所有参数，并且包装成为一个元组赋值给args

## 可变长实参

```
def func(x,y,z,*args)

print(x,y,z,*args)

func(1,*(2,4),3,5)#打散实参

return 得到 1,2,4,(3,5)
```

## 强制位置参数

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)
```

a,b 必须使用位置参数，而不能使用关键字参数

e,f 必须使用关键字参数，而不能使用位置参数

## 匿名函数

lambda arg : expression

匿名函数不会用全局变量

<lambda>() missing 1 required positional argument: 'x’

```
function_4 = lambda x : x * 2
print (function_4(2))
```

需要x参数

## 闭包函数

> 在一个外函数中定义了一个内函数，内函数里运用了外函数的临时变量，并且外函数的返回值是内函数的引用。这样就构成了一个闭包。

“闭包”的最大的作用——保存局部信息不被销毁，不会污染全局变量

- 闭包在定义函数时，比普通函数要复杂一些；
- 在调用时闭包要方便些，不需要重复传入变量a, b的值，简化代码的调用方式；
- 普通函数的局部变量在调用函数之后会被回收。

## 装饰器

装饰器让你在一个函数的前后去执行代码。

```
def decoration(func):#装饰器就是把一个函数包裹起来,这个是包装，而不是包的函数,装饰器实际上是一个闭包
    def decorator ():
        print('1111Now we start to decorate your function')
        func()
        print('33333Now,we have finished decorating your function')
    return decorator#返回内部函数

def function_need_decorate():
    print('22222I am the function that need to be decorated')

testing = decoration(function_need_decorate)
testing()#这是一种表达方式

@decoration#这是语法糖的表达方式
def function_need_decorate():
    print('22222I am the function that need to be decorated')
function_need_decorate()
```