**函数，闭包，装饰器**

**函数**

**位置参数和关键字参数**

位置参数：与形式参数的数量和位置对应的实际参数

关键字参数：通过形参的名字来传入的实际参数

**/和\*作为参数**

/后的参数可以为位置参数，也可以为关键字参数

\*后的参数必须为关键字参数，例如：

def func(para1,para2,/,para3,\*,para4,para5)

para1和para2必须为位置参数  
para3可以为位置参数，也可以为关键字参数

para4和para5必须为关键字参数

**星号用于打包：\*args和\*\*kwargs**

对函数参数进行打包指的是将传递给函数的元素打包成元祖或字典

\*args和\*\*kwargs主要用于在函数定义过程中接受可变数量的参数\*args表示位置参数，以元组形式接受非键值对的可变数量参数；\*\*kwarg表示关键字参数，以字典形式接受可变数量的键值对

注意：\*args必须在\*\*kwargs之前

def test1(a,\*args,\*\*kwargs):

*#通过\*区分，名称不固定，如\*args也可以是\*a,\*b等*

print(a)*#a=1*

print(args)

print(kwargs)

test1(1,3,5,7,c='2',d=4)

>>>

1

(3, 5, 7)

{'c': '2', 'd': 4}

test(None,None,c='2',d=4)

>>>

None

(None,)

{'c': '2', 'd': 4}

**星号\*用于拆包**

拆包是将传递给函数的列表，元祖或字典拆解为多个独立的元素

def func(a,\*lis):

print(a)

print(lis)

list\_1 = [1, 2, 3]

func(\*list\_1)

>>>

1

(2, 3)

**函数作为另一个函数的参数**

将fun1作为参数传入fun2后，即可在fun2中使用fun1()，这使得fun2可以接受接受多种不同的fun1，增加了代码的适应性

def add\_params(a,b):

return a+b

def mult\_params(func,a,b,c):

return func(a,b)\*c

mult = mult\_params(add\_params,1,2,3)

print(mult)

>>>

9

**闭包**

闭包可定义为：如果在一个内部函数里，对在外部作用域(但不是在全局作用域)的变量进行引用，那么内部函数就称为闭包(closure)

其他定义1：闭包是函数，一个可以访问别的函数作用域中的变量的函数

其他定义2：闭包=环境+控制，环境是访问到的外部函数作用域中的变量，控制是内部函数

**闭包的作用**

①保存外部函数作用域内变量的安全，使之只能通过内部函数访问

def f():

a = 5

def g():*#g()是闭包*

print(a)

return g

G = f()

>>>

5

②使得外部函数运行并返回闭包或闭包函数运行后，计算机不会回收外部函数作用域中的变量所占用的资源，保持当前的环境

origin = [0, 0]*#坐标系统原点*

def create(pos=origin):

def player(direction,step):

new\_x = pos[0] + direction[0]\*step

new\_y = pos[1] + direction[1]\*step

pos[0] = new\_x

pos[1] = new\_y

return pos

return player

Li = create()*#返回闭包，创建玩家Li，起点为原点*

print(Li([1,0],10))*#向x轴正方向移动10步*

print(Li([0,1],20))*#向y轴正方向移动20步*

print(Li([-1,0],10))*#向x轴负方向移动10步*

>>>

[10, 0]

[10, 20]

[0, 20]

**装饰器**

装饰器的作用是对原有函数进行功能扩展，而且还不需要修改原有函数的内容，也不需要修改原有函数的调用

*"""手写一个简单装饰器"""*

import time

def baiyu():

print("我是攻城狮白玉")

time.sleep(2)

def count\_time(func):*#wrapper是一个装饰器，其作用是将传入的函数func包裹起来*

def wrapper():*#wrapper函数是要实现的装饰器内容*

t1 = time.time()

func()

print("执行时间为：", time.time() - t1)

return wrapper

baiyu = count\_time(baiyu)

baiyu()

>>>

我是攻城狮白玉

执行时间为：2.0050249099731445

**装饰器的语法糖**

语法糖指计算机语言中添加的某种语法，这种语法对语言的功能并没有影响，但是能够增加程序的可读性，从而减少程序代码出错的机会，更方便使用。装饰器的语法糖是@，简单装饰器可以通过语法糖来实现

import time

def count\_time(func):

def wrapper():

t1 = time.time()

func()

print("执行时间为：", time.time() - t1)

return wrapper

@count\_time*#@fun使得下面的函数g实际上成为f(g())*

def baiyu():

print("我是攻城狮白玉")

time.sleep(2)

baiyu()*#用语法糖之后，可省去baiyu = count\_time(baiyu)*

>>>

我是攻城狮白玉

执行时间为： 2.015547037124634