类属性：

\_\_name\_\_: 类名

\_\_doc\_\_ : 类的文档字符串

\_\_dict\_\_ : 类的属性（包含一个字典，由类的数据属性组成）

dir() : 函数的属性与方法

dir(\_\_builtins\_\_)：返回python3所以的内置函数

class People(object):

'人的类'

def \_\_init\_\_(self,name,sex,height):

self.name = name

self.sex = sex

self.height = height

def nationality(self):

print('中国')

>>> People.\_\_name\_\_

'People'

>>> People.\_\_doc\_\_

'人的类'

>>> People.\_\_dict\_\_

mappingproxy({'\_\_module\_\_': '\_\_main\_\_', '\_\_doc\_\_': '人的类', '\_\_init\_\_': <function People.\_\_init\_\_ at 0x039A70C0>, 'nationality': <function People.nationality at 0x03A081E0>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'People' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'People' objects>})

>>> dir(People)

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'nationality']

特殊方法：

\_\_init\_\_ 初始化

\_\_repr\_\_ object

\_\_str\_\_ print object (如果类里面先定义了\_\_repr\_\_的话print x时也会返回对应的值)

\_\_call\_\_ object() 使实例可被调用

class A(object):

def \_\_init\_\_(self,name):

self.name = name

def \_\_repr\_\_(self):

return('hahah')

def \_\_str\_\_(self):

return ('mmmm')

def \_\_call\_\_(self):

return('xxxx')

>>> a = A('zhong') #先实例化后面才能输出

>>> a

hahah

>>> print(a)

mmmm

>>> a()

'xxxx'

运算符魔法方法

\_\_add\_\_(self,other) x+y

\_\_sub\_\_(self,other) x-y

\_\_mul\_\_(self,other) x\*y

\_\_mod\_\_(self,other) x%y

\_\_iadd\_\_(self,other) x+=y

\_\_isub\_\_(self,other) x-=y

\_\_radd\_\_(self,other) y+x

\_\_rsub\_\_(self,other) y-x

\_\_imul\_\_(self,other) x\*=y

class B(object):

def \_\_init\_\_(self,aa,bb):

self.aa = aa

self.bb = bb

def \_\_add\_\_(self,other):

return self.aa+other.aa

b = B(2,'BB')

c = B(4,'bb')

>>> b+c #这里运算符号一定要与写类时的方法对的上，就是+对\_\_add\_\_才能运算

6 但结果是return的公式

>>> b-c

Traceback (most recent call last):

File "<pyshell#61>", line 1, in <module>

b-c

TypeError: unsupported operand type(s) for -: 'B' and 'B'

装饰器：

@property 装饰过的函数返回的不再是一个函数，而是一个property对象

装饰过后的方法不再是可调用的对象，可以看做数据属性直接访问。

把方法函数变成数据函数不使用()

@staticmethod 把没有参数的函数装饰过后变成可被实例调用的函数，

函数定义时是没有参数的,( )里不用写self

@classmethod 把装饰过的方法变成一个classmethod类对象，

既能能被类调用又能被实例调用。

注意参数是cls代表这个类本身。而是用实例的方法只能被实例调用。

class A(object):

def \_\_init\_\_(self,name):

self.name = name

@property

def show(self):

print ('xxxxxx')

@staticmethod

def fun():

print('mmmm')

@classmethod

def area(cls):

print('yyyyyyyy')

>>> a = A('name') #先实例化

>>> a.show #如果不加@property，show要加()即a.show()

xxxxxx

>>> a.fun() #写函数的时候括号里不用加self，也可直接用fun()

mmmm

>>> a.area() #被实例调用

yyyyyyyy

>>> A.area() #被类调用

yyyyyyyy

函数装饰器 (先要了解内嵌函数和闭包)

使用函数的嵌套，外层函数传进来的参数是一个函数对象，里层函数对传进来的这个函数进行加工处理后再返回处理后的函数。

方法1：

def fun(a):

return a\*a

def ff1(x):

def ff2(a):

return x(a)+1 #x(a)就说明这是一个函数

return ff2

>>> ff1(fun)(3) #ff1(fun),把fun赋值给x,在函数的内部隐藏执行了

10

>>> fun(3) #x(a)=fun(a)=9

9

--------------------------------------------------------------------------------------------------------------

方法2：

def ff1(x):

def ff2(a):

return x(a)+1

return ff2

@ff1 #fun1 = ff1(fun1)(b) 将下面的函数fun1带入到ff1里运算

def fun1(b):

return b\*\*b

>>> fun1(3)

28

案例二：

def fun1(f):

def fun2(\*arg1,\*\*arg2):

return f(\*arg1,\*\*arg2)+1 #f(\*arg1,\*\*arg2)就说明这是一个函数

return fun2

def ff(a,b):

return a\*b

>>> ff = fun1(ff)

>>> ff(1,2)

3