## 1、装饰器

装饰器（简洁说：封装一个函数，用这样或者那样的方式来修改他的行为。在不改变原函数的内部逻辑和函数调用代码的情况下进行功能扩展。由decorator和嵌套的wrapper组成一个共生体，这个共生体我们叫闭包，闭包满足三个条件：1、函数嵌套函数2、函数返回函数3、函数之间有引用）

通过decorator来传入实际要运行的函数，通过wrapper函数来传入函数参数并调用函数，同时在wrapper函数实现需要的逻辑。

@ a\_new\_decorator

#== a\_function\_requiring\_decoration = a\_new\_decorator(a\_function\_requiring\_decoration)

def a\_function\_requiring\_decoration():

print(a\_function\_requiring\_decoration.\_\_name\_\_) #\_\_name\_\_表示当前模块执行过程的名称

被装饰后的函数其实是另外一个函数了，\_\_name\_\_(变成wrapper)、\_\_doc\_\_(变成decorator的注释)的属性发生变化。python的functools包提供了叫wraps的decorator能保留原函数的名称和函数属性，通过复制操作。

python 返回函数名可以被传递，赋值给别的变量。加括号则会执行

二维数组 [::-1] 返回的是子数组内部顺序不变，子数组在原始数组的顺序逆序

数组+=单层元组

数组会把元组的元素一一加入

l = []

dc = (1,2)

l+=dc

Zip用法(一一对应组成元组)

a = ["Peter", "Ben", "Alice", "Jim"]

b = ["Apple", "Banana", "Pear", "Orange"]

pack = list(zip(a, b))

print(pack)



print(list(zip(\*pack)))

#[('Peter', 'Ben', 'Alice', 'Jim'), ('Apple', 'Banana', 'Pear', 'Orange')]

matrix = [[1,2,3],[4,5,6],[7,8,9]]

print(zip(\*matrix))

[(1, 4, 7), (2, 5, 8), (3, 6, 9)]

！！！改进

可用map

list(map(list,zip(\*matrix))) >>>[[1, 4, 7], [2, 5, 8], [3, 6, 9]]

python虽然不能利用多线程实现多核任务，但可以通过多进程实现多核任务。多个Python进程有各自独立的GIL锁，互不影响。

python线程池

在os启动时创建大量空闲线程，函数执行结束后，线程不死亡，回到线程池变成空闲状态

python

nums = [1,2,3]

sorted(nums) #sorted(iterable,key,reverse=False) False升序True降序 key，进行比较的元素。多维时，默认以第一个数排序，若要以第二个为key，则用key=lambda student:student[2]。不改变传入序列，返回一个新的序列。

通过定义\_\_lt\_\_，直接sorted排序自定义对象。（定义了小于操作符，<的行为。

）

**如果逆序不能前面加个 '-' 来解决 ， 可以用下面的办法 例如： 排序的是个字符串**

**class Reversinator(object):**

**def \_\_init\_\_(self, obj):**

**self.obj = obj**

**def \_\_lt\_\_(self, other):**

**return other.obj < self.obj**

其实也可以操作能操作的元素类型

a = [(2,'3'),(4,'1'),(2,'8'),(2,'1'),(3,'4')]

print sorted(a, key=lambda x: (x[0], Reversinator(x[1])))

res = sorted(res,key=lambda x:(-x[0],x[1])) #第一个递减，第二个递增

按照元组的第一个从小到大排序

2. 如果第一个相同 则按照元组第2个从大到小 排序

a = [[2,3],[4,1],(2,8),(2,1),(3,4)]

b = sorted(a,key=lambda x: (x[0], -x[1]))

print b

list.sort(func=None,key=None,reverse=False)直接在原列表上做排序

dic.items() 将KV转换为元组，并放在list内返回

str不可直接修改只能转化成list修改

map(function,iterable,…)根据提供函数对序列做映射。py3返回迭代器，需要list()

def square(x) : # 计算平方数

... return x \*\* 2

map(square, [1,2,3,4,5]) >>> [1,4,9,16,25]

map(lambda x,y: x+y, [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]) # 提供了两个列表，对相同位置的列表数据进行相加

python filter(function,iterable) #用于过滤序列，返回符合条件的元素组成的新列表

def is\_odd(n):

return n % 2 == 1

newlist = filter(is\_odd, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) >>>[1, 3, 5, 7, 9]

list = [1]

list.extend(iterable) #list.extend([1,2]) >>>[1,1,2]

string是不可变类型，string.replace不改变原字符，只是返回一个新的。

创建二维数组：

[ [0 for I in range(3)] for j in range(3)]

python自定义函数内修改矩阵值是有效的，因为传入的是引用变量

a=b地址相同，拷贝需要a=b[:]

python 从各位将某个数分开的方法 nums = [int(x) for x in str(num)]

dict.get(key,default=None)

“”.join(str)

lambda

g = lambda x:x+1

等价于

def g(x):

return x+1

应用

def sq(x):

return x\*\*2

map(sq,[y for y in range(10)])

等价于 map(lambda x:x\*\*2,[y for y in range(10)])

set

a = set()

a.add()

a.update({a,b})可以添加任何结构 (“string”)将字符串拆分成单个字符，({“string”})将字符串添加到集合

a.remove(x) #不存在会报错

a.discard(x)#不存在不会报错

a|b 并集（int|int 按位或） a^b异或 a&b交集

import functools

@functools.lru\_cache(None)

def (**dfs**)

允许我们将一个函数的返回值快速地缓存或取消缓存。

该装饰器用于缓存函数的调用结果，对于需要多次调用的函数，而且每次调用参数都相同，则可以用该装饰器缓存调用结果，从而加快程序运行。

该装饰器会将不同的调用结果缓存在内存中，因此需要注意内存占用问题。

如果局部要对全局变量修改

count = 0

def global\_test():

global count

count += 1

nonlocal申明的变量不是局部变量，也不是全局变量，而是外部函数嵌套的变量

def nonlocal\_test():

count = 0

def test2():

nonlocal count

count += 1

return count

return test2

54,78 53,77 52,78 51,77 50,78 49,78 48,79 47,79 46,80 45,79 44,80 43,80 42,79 41,80 40,79 39,78 38,79 37,79 36,80 35,81 34,81 33,80 32,79 32,78 31,77 31,76 30,75 31,74 32,73 31,72 30,71 29,70 28,70

54,78 53,77 52,78 51,79 50,80 50,81 49,82 48,81 47,81 46,80 45,79 44,80 43,80 42,79 41,80 40,81 39,81 38,80 37,79 36,80 35,81 34,81 33,80 32,79 32,78 31,77 31,76 30,75 31,74 30,73 29,73 28,72 28,71 28,70

12028

4,4 5,3 6,3