**day17笔记**

1.高阶函数：

特点：函数的形参位置必须接受一个函数对象

分类学习：

**1).map(fn,lsd1,[lsd2])：**

参数一：fn --> 函数对象

参数二：lsd1 --> 序列对象(字符串、列表、range...)

功能：

将fn函数作用于lsd1中的每一个元素上,

将每次执行的结果存入到一个map对象中返回；

【注意】得到的这个map对象是一个**迭代器对象**，**属于惰性序列**的范畴

**需求：lt = ['1','2','3','4','5'] --> [1,2,3,4,5]**

*#代码实现一：使用老技术来实现*

lt = [**'1'**,**'2'**,**'3'**,**'4'**,**'5'**,**'6'**]  
lt1 = []  
**for** i **in** lt:  
 num = int(i)  
 lt1.append(num)  
*#由于列表属于非惰性序列范畴，即可以直接打印看效果*print(lt1) *# [1, 2, 3, 4, 5, 6]*

*#代码实现二：使用新技术来实现  
#思路步骤一：定义一个函数，功能：将str数据 --> int数据*

**import** collections  
lt = [**'1'**,**'2'**,**'3'**,**'4'**,**'5'**,**'6'**]  
**def** chr2Int(chr):  
 *#① return {'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,*

*'9':9}.[chr]  
 #②* **return** {**'0'**:0,**'1'**:1,**'2'**:2,**'3'**:3,**'4'**:4,**'5'**:5,**'6'**:6,**'7'**:7,**'8'**:8,

**'9'**:9}.get(chr)  
 *#③* *return int(chr) 比上面两个字典通过键匹配值简便很多，int强转*  
  
mo = map(chr2Int,lt) *#这里的chr2Int后面不能加(),也不能传参\*\*\*\*\**print(map,type(mo)) *# <class 'map'> <class 'map'>*  
print(isinstance(mo,collections.Iterator)) *# True*

*上面代码是验证以下，是迭代器对象才能用next，返回 True，所以可以*   
print(next(mo))  *# 1* **将map对象(惰性的)转换为list对象(非惰性的)**print(list(mo)) *#[2, 3, 4, 5, 6]  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*#代码实现三：终极操作*  
print(list(map(chr2Int,lt))) *#[1, 2, 3, 4, 5, 6]*  
print(list(map(int,lt)) *#[1, 2, 3, 4, 5, 6]*

map(int,lt)：执行过程如下：

1).lt --> 取出第一个元素：'1'当做实际参数传递给int函数的形参位置 --> int('1')

将转换以后的结果：1保留到map对象的第一个元素位置

2).lt --> 取出第二个元素：'2'当做实际参数传递给int函数的形参位置 --> int('2')

将转换以后的结果：2保留到map对象的第二个元素位置

以此类推...

直到map函数执行完了，整个map对象才真正成型了...

**需求1：lt = [1,2,3,4,5] --> 得到：['1','2','3','4','5']**

lt = [1,2,3,4,5]  
*#自定义函数：从int --》 str***def** int2Str(i):  
 **return** str(i)  
print(list(map(int2Str,lt))) *#['1', '2', '3', '4', '5']*  
print(list(map(str,lt))) *#['1', '2', '3', '4', '5']*  
print(list(map(**lambda** x: str(x),lt))) *#['1', '2', '3', '4', '5']*

**需求2：lt = [1,2,3,4,5] --> 得到：[1,4,9,16,25]**

lt = [1,2,3,4,5]

*#自定义函数：目标实现开方操作***def** kaifang(num):  
 **return** num \*\* 2  
  
print(list(map(kaifang,lt))) *# [1, 4, 9, 16, 25]*

*#或者map结合匿名函数使用，用的比较多*  
print(list(map(**lambda** x: x \*\* 2,lt))) *# [1, 4, 9, 16, 25]*

**2).reduce(fn,lsd)：**

参数一：fn --> 函数对象

参数二：lsd --> 序列对象

功能：

先将lsd中的第一和第二个元素传入到fn中参与运算，

运算后得到结果，再和第三个元素传入到fn中参与运算，

以此类推...

【注意】：reduce函数属于functools模块中的函数，所以需要显示的先导入functools模块再使用from functools import reduce

**tp = (1,2,3,4)  
自定义函数 --> add 作用：对列表中的元素进行求和操作 def add(x,y）  
使用reduce函数执行过程如下：  
第一次：add(1,2)  
第二次：add(add(1,2),3)  
第三次：add(add(add(1,2),3),4)  
#需求：得到元祖中元素的和值**tp = (1,2,3,4)  
*#代码实现一：递归的思想来实现***def** mySum(num):  
 **if** num == 1:  
 **return** 1  
 **return** num + mySum(num - 1)  
  
print(mySum(4))  
  
*#代码实现二：新技术(reduce)*

**from** functools **import** reduce**def** add(x,y):  
 **return** x + y  
  
res = reduce(add,tp)  
print(res,type(res)) *#10 <class 'int'>*  
print(reduce(**lambda** x,y: x + y,tp))  
print(sum(tp)) *#直接使用内置函数sum（）了*

**#需求：lt = [1,2,3,4] 得到其中元素的乘积**lt = [1,2,3,4]  
print(reduce(**lambda** x,y: x \* y,lt))

**需求：从键盘读入一个整数字符串数据，例如：'12345'  
需要将其转换为12345；注意：不能直接使用int()来实现  
思路：使用map和reduce配合来实现  
步骤一：'12345' --》拆分为散装数据：1 2 3 4 5 可以使用map来实现  
步骤二：将map对象中的数据 1 2 3 4 5 组合成为 --> 12345 可以使用reduce来实现  
from** functools **import** reducestr1 = **'12345'**

**def** chr2Int(str):  
 **return** int(str)

**def** zuhe(x,y):  
 **return** x \* 10 + y  
  
mo = map(chr2Int,str1)  
num = reduce(zuhe,mo)  
print(num,type(num)) *#12345 <class 'int'>*  
*#终极版：*print(reduce(**lambda** x,y: x \* 10 + y,map(int,str1))) *#12345*

**3).filter(fn,lsd)：**

filter函数：过滤数据的，最终返回一个**惰性序列对象**(filter对象，**迭代器对象**)

解释：filter的意思：在计算机领域中我们都称为过滤器

格式：filter(fn,lsd)：

参数和map、reduce一样理解

功能：

将lsd中的每一个元素都给到fn函数

如果fn函数的返回值为True，那么就保留这个元素到filter对象中

如果fn函数的返回值为False，那么就舍弃这个元素，不会保留到filter对象中

最终filter函数执行完毕了，返回给程序一个filter对象(迭代器对象)

**#需求：lt = [1,2,3,4,5,6,7,8] --> 得到：[2,4,6,8]**

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

*#代码实现一：老技术*lt1 = []  
**for** i **in** lt:  
 **if** i % 2 == 0:  
 lt1.append(i)  
  
print(lt1)  
  
*#代码实现二：新技术(filter)***def** func(o):  
 **if** o % 2 == 0:  
 **return True  
 return False**

fo =filter(func,lt)  
print(fo,type(fo)) *#<filter object at 0x0000000001E8F710> <class 'filter'>*  
print(list(filter(func,lt))) *#[2, 4, 6, 8]*  
print(isinstance(fo,collections.Iterator)) *#Ture*print(isinstance(fo,collections.Iterable)) *#Ture*print(next(fo)) *#2*  
print(next(fo)) *#4*  
print(list(fo))  *#[6, 8]*

*# 终极版：*print(list(filter(**lambda** x:x % 2 == 0,lt))) *#[2, 4, 6, 8]*

**需求1：  
lt = [345,0,'abcde',1.2,0,3.14,0.0,'haha','hehe',True,False,[]**

**,(),{},{1,2,3},[10,20,30],{'name':'zs','age':30}，None]  
得到如下效果：  
lt = [345,'abcde',1.2,3.14,'haha','hehe',True,{1,2,3},[10,20,30],{'**

**name':'zs','age':30}]**

lt = [345,0,**'abcde'**,1.2,0,3.14,0.0,**'haha'**,**'hehe'**,**True**,**False**,[]

,(),{},{1,2,3},[10,20,30],{**'name'**:**'zs'**,**'age'**:30},**None**]  
  
print(list(filter(**lambda** x: bool(x),lt)))  
*#或者*

print(list(filter(bool,lt)))  
 **需求2：lt1 = ['aaaaaaaa','bbbbb','cccccc','ddd']  
得到如下效果：['aaaaaaaa','cccccc']**lt1 = [**'aaaaaaaa'**,**'bbbbb'**,**'cccccc'**,**'ddd'**]  
print(list(filter(**lambda** x: len(x) > 5,lt1)))

**归纳总结：高阶函数以及匿名函数之间的配合使用：  
模板一：lambda和filter的配合使用  
#需求：lt = [1,2,3,4,5,6,7,8,9] --> 得到[3,6,9]**lt = [1,2,3,4,5,6,7,8,9]  
print(list(filter(**lambda** x: x % 3 == 0,lt)))  
  
**模板二：lambda 和map的配合使用  
#需求：演示开平方操作 --> 容器对象：range**mo = map(**lambda** x: x \*\* 2,range(5))  
print(list(mo))  
  
**模板三：在模板二的基础上进行功能扩展：range(10)  
 过滤以后保留的数据范围大小为：(5,50)之间**mo = map(**lambda** x: x \*\* 2,range(10))  
fo = filter(**lambda** x: x > 5 **and** x < 50,mo)  
print(list(fo))  
  
**模板四：lambda 和reduce配合使用  
#求和值**

**import** functoolslt = [1,2,3,4,5]  
my\_sum = functools.reduce(**lambda** x,y: x + y,lt)  
print(my\_sum)  
**模板五：求两个列表对象中元素的和，返回新列表  
lt1 = [1,2,3,4]  
lt2 = [5,6]  
结果：lt3 = [6,8,10,12]**

lt1 = [1,2,3,4]  
lt2 = [5,6]  
  
mo = map(**lambda** x,y: x + y,lt1,lt2)  
print(list(mo))  
 **模板六：求字符串中每个单词的长度  
content = "welcome to shanghai"  
结果：[7,2,8]**content = **"welcome to shanghai"***#使用切割的思想：切完之后得到一个列表对象，内部元素为["welcome","to","shanghai"]*words\_list = content.split()  
mo = map(len,words\_list)  
print(list(mo))

**补充：**

**sorted()函数的使用：**

我们将sorted和sort进行一番比较：

**相同点：**它们都是来实现排序的操作(功能层面)

**不同点：**列表中的sort函数，它执行完毕后会直接影响原本这个list的内部结构(内部的数据发生改变了)；而内置函数sorted函数，它执行完毕后不会影响原本容器中的内部结构，而会返回一个**新的列表**给程序；

回顾排序：

选择排序，冒泡排序它们的性能都很低下；意味着开发不会用，但是面试喜欢面(一般开发不用的，面试都喜欢面)

**代码演示**

lt = [42,-9,0,-11,67,92,100,-33]

lt.sort()  
print(lt) *# [-33, -11, -9, 0, 42, 67, 92, 100]*

lt = [42,-9,0,-11,67,92,100,-33]

lt1 = sorted(lt)  
print(lt1,type(lt1))  *#[-33, -11, -9, 0, 42, 67, 92, 100] <class 'list'>*print(lt)  *#[42, -9, 0, -11, 67, 92, 100, -33]，lt没有变*

tp = (42,-9,0,-11,67,92,100,-33)

lt1 = sorted(tp)  *#得到的仍然是list对象*print(lt1,type(lt1))  *#[-33, -11, -9, 0, 42, 67, 92, 100] <class 'list'>*print(lt) *#(42, -9, 0, -11, 67, 92, 100, -33)，tp没有变*

r = range(10)

lt1 = sorted(r，reverse=**True**)  *#reverse=True表示降序*  
print(lt1,type(lt1)) *#[9,8,7,6,5,4,3,2,1,0] <class 'list'>*

lt = [42,-9,0,-11,67,92,100,-33]

*#这里的思想和高阶函数思想很像，****key****=abs相当于取每个数的绝对值*lt2 = sorted(lt,key=abs,reverse=**True**)

print(lt2) *#[100, 92, 67, 42, -33, -11, -9, 0]*

lt3 = [**'aaaaa'**,**'bbb'**,**'ccccccc'**,**'d'**]  
lt4 = sorted(lt3,key=len,reverse=**True**)  
print(lt4) *#['ccccccc', 'aaaaa', 'bbb', 'd']*print(lt3) *#['aaaaa', 'bbb', 'ccccccc', 'd']*

*#回顾：*lt5 = [  
 {**'name'**:**'武大郎'**,**'age'**:41},  
 {**'name'**:**'武二郎'**,**'age'**:28},  
 {**'name'**:**'金莲'**,**'age'**:22},  
 {**'name'**:**'西门大官人'**,**'age'**:24},  
 {**'name'**:**'王婆'**,**'age'**:50}  
]  
*#年龄排序：*lt6 = sorted(lt5,reverse=**True**,key=**lambda** x:x[**'age'**])  
print(lt6)

*#得到[{'name': '王婆', 'age': 50}, {'name': '武大郎', 'age': 41}, {'name': '武二郎', 'age': 28}, {'name': '西门大官人', 'age': 24}, {'name': '金莲', 'age': 22}]*

2.单元测试：（还未回顾）

目的：测试功能的准确度

测试：对于代码的功能进行检测，并不注重语法问题的检测

分类：

1).对函数(模块中的)进行单元测试

2).对类(类中的函数)进行单元测试

相同点：

1).都需要导入unittest模块

2).都需要自定义类并且继承unittest下的TestCase类

3).都可以选择性的重写setUp()和tearDown()方法

4).都需要断言(调用self.assertEqual()函数)

5).都需要通过unittest去启动main()函数来执行

不同点

对类进行测试，需要在断言之前实例化一个当前类对象

**Py2和py3的区别：(面试会用)**

1. py2的性能比py3高
2. Py3在编码上有扩充，所以能够使用中文字符
3. Py3中取消了 <> 定义不低于的方式
4. Py3加入了with和as关键字
5. 加入了nonlocal关键字（局部变量中改变全局变量）
6. Py3舍去了file类，融合到了os模块中

......