**day07笔记**

day07：

1.list中常用的函数：

lt = [**'路飞'**,**'索罗'**,**'山治'**,**'娜美'**,**'乌索普'**,**'乔巴'**,**'布鲁克'**]  
**#append(obj)：将obj存入到列表的最后一个位置**lt.append(**'香克斯'**)  
print(lt)   
print(len(lt))  
*# lt.append(['卡卡西','鸣人','佐助','樱'])  
# print(lt)  
# print(len(lt))***#extend(iterable):将iterable对象中的每个元素分别取出存入到列表的最后位置**lt.extend([**'卡卡西'**,**'鸣人'**,**'佐助'**,**'樱'**])  
print(lt)  
print(len(lt))  
*#得到['路飞', '索罗', '山治', '娜美', '乌索普', '乔巴', '布鲁克', '香克斯', '卡卡西', '鸣人', '佐助', '樱']*

*12*

lt.extend(**'大蛇丸'**)  
print(lt)  
print(len(lt))  
*#得到['路飞', '索罗', '山治', '娜美', '乌索普', '乔巴', '布鲁克', '香克斯', '卡卡西', '鸣人', '佐助', '樱', '大', '蛇', '丸']*

*15* **以下代码会报错：  
由于200是int类型的数据，不属于iterable，所以不能被extend()函数所使用***# lt.extend(200)  
# print(lt)  
# print(len(lt))*

**# index(obj)：返回obj在列表中出现的索引位置；  
#如果存在多个相同的obj元素，返回的是第一次出现的那个元素的索引，  
#如果一个有没有，直接报错了;**

lt = [**'路飞'**,**'索罗'**,**'山治'**,**'娜美'**,**'乌索普'**,**'乔巴'**,**'布鲁克'**,**'娜美'**]print(lt.index(**'娜美'**))  
*# print(lt.index('娜美1')) #报错***#count(obj)：返回obj在列表中出现的次数；如果一次都没有出现，返回0**print(lt.count(**'路飞'**)) *#1*  
print(lt.count(**'娜美'**)) *#2*  
print(lt.count(**'娜美1'**)) *#0*

**#pop([index])：如果是空参数的pop()，将列表的最后一个元素弹出(作为返回值给到程序)；  
#如果有参数，必须是索引值，将具体索引位置上的元素弹出；**lt = [**'路飞'**,**'索罗'**,**'山治'**,**'娜美'**,**'乌索普'**,**'乔巴'**,**'布鲁克'**,**'娜美'**]  
print(lt.pop()) *#娜美*  
print(lt) *#['路飞','索罗','山治','娜美','乌索普','乔巴','布鲁克']*  
print(lt.pop(1)) *#索罗*  
print(lt) *#['路飞', '山治', '娜美', '乌索普', '乔巴', '布鲁克']*  
*#lt.pop('乔巴') #报错 非法操作*

lt = [**'路飞'**,**'索罗'**,**'山治'**,**'娜美'**,**'乌索普'**,**'乔巴'**,**'布鲁克'**,**'娜美'**]  
**#remove(element)：删除和element匹配的某个元素，没有返回值；如果有多个匹配的，只删除第一个**print(lt.remove(**'山治'**)) *# None*  
print(lt) *#['路飞','索罗','娜美','乌索普','乔巴','布鲁克','娜美']*lt.remove(**'娜美'**)  
print(lt)  *#['路飞','索罗','乌索普','乔巴','布鲁克','娜美']***#clear()：清空列表，保留容器**print(lt.clear()) *#None*print(lt) *#[]*

lt = [**'路飞'**,**'索罗'**,**'山治'**,**'娜美'**,**'乌索普'**,**'乔巴'**,**'布鲁克'**,**'娜美'**]  
**#reverse()：反转列表元素**lt.reverse()  
print(lt)

*#得到['娜美','布鲁克','乔巴','乌索普','娜美','山治','索罗','路飞']*  
**#sort([reverse=True],[key=...])：默认以升序排列(reverse=False)，如果想要降序排列，显示定义reverse=True**lt1 = [53,23,-17,9,-21,0,79,88,-30]  
lt1.sort()  
print(lt1) *#[-30, -21, -17, 0, 9, 23, 53, 79, 88]***#降序排列**lt1.sort(reverse=**True**)  
print(lt1) *#[88, 79, 53, 23, 9, 0, -17, -21, -30]*

**以下内容很重要，好好听...  
讲解：论浅拷贝&深拷贝  
涉及的函数：  
list中的copy()、copy模块中的copy()和deepcopy()  
  
浅拷贝：1).引用地址传递 代码：lt1 = lt  
 2).list中的copy()、copy模块中的copy()函数都是  
  
深拷贝：copy模块中的deepcopy()函数（完全脱离）  
import** copy  
lt = [11,22,33,44,55,[1,2,3]]  
lt1 = lt *#引用地址传递*lt2 = lt.copy()  
lt3 = copy.copy(lt)  
**lt4 = copy.deepcopy(lt)**print(lt,id(lt[-1])) *#[11, 22, 33, 44, 55, [1, 2, 3]] 42560072*print(lt1,id(lt1[-1])) *#[11, 22, 33, 44, 55, [1, 2, 3]] 42560072*print(lt2,id(lt2[-1])) *#[11, 22, 33, 44, 55, [1, 2, 3]] 42560072*print(lt3,id(lt3[-1])) *#[11, 22, 33, 44, 55, [1, 2, 3]] 42560072***print(lt4,id(lt4[-1])) *#[11, 22, 33, 44, 55, [1, 2, 3]] 42561352***  
print(lt,id(lt)) *#[11, 22, 33, 44, 55, [1, 2, 3]] 32467720*print(lt1,id(lt1)) *#[11, 22, 33, 44, 55, [1, 2, 3]] 32467720*print(lt2,id(lt2)) *#[11, 22, 33, 44, 55, [1, 2, 3]] 32489544*print(lt3,id(lt3)) *#[11, 22, 33, 44, 55, [1, 2, 3]] 32468936***print(lt4,id(lt4)) *#[11, 22, 33, 44, 55, [1, 2, 3]] 32468872***lt.pop()  
print(lt,id(lt)) *#[11, 22, 33, 44, 55] 37116680*print(lt1,id(lt1)) *#[11, 22, 33, 44, 55] 37116680*print(lt2,id(lt2)) *#[11, 22, 33, 44, 55, [1, 2, 3]] 37142600*print(lt3,id(lt3)) *#[11, 22, 33, 44, 55, [1, 2, 3]] 37117896***print(lt4,id(lt4)) *#[11, 22, 33, 44, 55, [1, 2, 3]] 37117832*'''  
lt = [11,22,33,44,55,[1,2,3]]  
'''**lt[-1].append(4)  
print(lt) *#[11, 22, 33, 44, 55, [1, 2, 3, 4]]*print(lt1) *#[11, 22, 33, 44, 55, [1, 2, 3, 4]]*print(lt2) *#[11, 22, 33, 44, 55, [1, 2, 3, 4]]*print(lt3) *#[11, 22, 33, 44, 55, [1, 2, 3, 4]]***print(lt4) *#[11, 22, 33, 44, 55, [1, 2, 3]]***

1. dict中常用的函数：

**update(dict)：  
例如：dic1.update(dic2)  
将dic2中每一个键值对和dic1中的进行比较，  
如果dic1中经过存在相同的 键，那么就保证键不变，值被覆盖；  
如果dic1中没有相应的键值对，那么就将其加入进dic1中**

dic1 = {**"name"**:**"张三丰"**,**"age"**:120,**"height"**:168.0}  
dic2 = {**"name"**:**"金毛狮"**,**"weight"**:100,**"faceValue"**:**False**,**"girlFriend"**:**"灭绝师太"**}  
dic1.update(dic2)  
print(dic1)  *#得到{'name': '金毛狮王', 'age': 120, 'height': 168.0, 'weight': 100, 'faceValue': False, 'girlFriend': '灭绝师太'}*print(dic2) *#得到{'name': '金毛狮王', 'weight': 100, 'faceValue': False, 'girlFriend': '灭绝师太'}*

**items()：返回一个dict\_items类型的对象，内部封装了一个一个元祖充当其元素，元祖中包含键和值两块内容  
keys()：返回一个dict\_keys类型的对象，内部封装了键集  
values()：返回一个dict\_values类型的对象，内部封装了值集**dic1 = {**"name"**:**"张三丰"**,**"age"**:120,**"height"**:168.0}  
print(dic1.items(),type(dic1.items()))

*#得到dict\_items([('name', '张三丰'), ('age', 120), ('height', 168.0)]) <class 'dict\_items'>*print(dic1.keys(),type(dic1.keys()))

*#得到dict\_keys(['name', 'age', 'height']) <class 'dict\_keys'>*print(dic1.values(),type(dic1.values()))  
*#得到dict\_values(['张三丰', 120, 168.0]) <class 'dict\_values'>***for** k,v **in** dic1.items():  
 print(k + **'==>'** + str(v))

*#得到 name==>张三丰*

*age==>120*

*height==>168.0*  
**for** k **in** dic1.keys():  
 print(k)  *# name*

*age*

*height*

**for** k **in** dic1:  
 print(k)

**for** v **in** dic1.values():  
 print(v) *#张三丰*

*120*

*168.0*

*100*

**popitem()：弹出最后一个键值对，以元祖的形式返回  
pop(key)：传入key时，弹出整个键值对，最终返回给程序value**dic2 = {**"name"**:**"金毛狮"**,**"weight"**:100,**"faceValue"**:**False**,**"girlFriend"**:**"灭绝师太"**}  
print(dic2.popitem()) *#('girlFriend', '灭绝师太')*print(dic2) *#{'name': '金毛狮王', 'weight': 100, 'faceValue': False}*print(dic2.pop(**'weight'**)) *#100*print(dic2) *#{'name': '金毛狮王', 'faceValue': False}***#copy()：**dic3 = dic2.copy()  
dic4 = dic2  
print(dic2,id(dic2)) *#得到{'name': '金毛狮王', 'weight': 100, 'faceValue': False, 'girlFriend': '灭绝师太'} 5999280*print(dic3,id(dic3)) *#得到{'name': '金毛狮王', 'weight': 100, 'faceValue': False, 'girlFriend': '灭绝师太'} 5999352*print(dic4,id(dic4))  *#得到{'name': '金毛狮王', 'weight': 100, 'faceValue': False, 'girlFriend': '灭绝师太'} 5999280*

3.set中常用的函数：

**add(e)：添加一个元素e(不可变的数据类型)到集合中  
pop()： 只能是空参的(集合没有索引的概念)，随机从集合中弹出一个元素  
remove(e)：从集合中删除和e匹配的元素  
clear()：清空集合，保留容器  
copy()： 拷贝**s1 = {10,100,3.14,**"abcd"**}s1.add(**'hello'**)  
print(s1)  
print(s1.pop())  
print(s1)  
s1.remove(100)  
s1.clear()  
print(s1) *#set()*

**issubset(s)：判断一个集合是另一个集合的子集吗？  
举例：s1.issubset(s2) s1是s2的子集吗？ 结果为：False  
issuperset(s)：判断一个集合是另一个集合的父集吗？  
举例：s1.issubset(s2) s1是s2的父集吗？ 结果为：True  
isdisjoint(s)：判断两个集合之间是否存在交集，如果存在，返回False；反之，返回True**

s1 = {10,100,3.14,**"abcd"**}s2 = {3.141,100}  
print(s1.issubset(s2)) *#False*  
print(s1.issuperset(s2)) *#True*  
print(s1.isdisjoint(s2)) *#False*  
print(s2.isdisjoint(s1)) *#False*

4.闭包：(思想)

满足以下三个条件：

1).有外部函数和内部函数这样的结构

2).外部函数中定义的变量需要被内部函数所使用

3).内部函数的对象作为返回值被外部函数返回

*演示闭包的定义和使用：***def** outer():  
 a = 10  
 **def** inner():  
 print(a + 20)  
 **return** inner  
f=outer() *#f() #f()相当于outer()()*f() *#*得到结果为30

**案例一：****def** outer():  
 count = 0  
 **def** inner():  
 **nonlocal** count  
 count += 1  
 print(**'hello，老郭，呵呵哒，%d'** %count)  
 **return** inner  
outer()() *#得到hello，老郭，呵呵哒，1*

outer()() *#得到hello，老郭，呵呵哒，1*

k = outer() *#得到hello，老郭，呵呵哒，1*k()  *#得到hello，老郭，呵呵哒，2*

k()  *#得到hello，老郭，呵呵哒，3*

**案例二：**

**def** outer():  
 a = 23  
 b = 44  
 c = 56  
 **def** inner(num):  
 d = a + num  
 e = b + num  
 f = c + num  
 **return** d,e,f  
 **return** inner  
tp1 = outer()(100)  
tp2 = outer()(200)  
print(tp1,type(tp1)) *# (123, 144, 156) <class 'tuple'>*  
print(tp2,type(tp2)) *# (223, 244, 256) <class 'tuple'>*

5.装饰器：

装饰器是闭包的一种使用场景(体现)；

python中的装饰器(函数)技术在定义格式上需要传入一个函数对象，

在此函数执行前或者后都可以追加其它的操作；

这样做的好处是：在不改变源码(原本的业务逻辑)的同时，进行功能的扩展；

在后期的开发中一般使用在：性能测试、插入日志、事物管理、权限校验...

它好比是一个切面(可插拔的)

也就是我们之后学习过程中遇到的面向切片

满足开放封闭原则：

开放：

在不改变源代码(不破坏原本的业务逻辑)的同时扩展新的功能

封闭：

不允许随意改变源码

**装饰器的定义格式：4种**

**格式一：无参无返回值的装饰器**

**def** outer(fn):  
 **def** inner():  
 print(**'功能开始前追加日志信息...'**)  
 fn()  
 print(**'功能结束后追加日志信息...'**)  
 **return** inner  
  
@outer  
**def** test():  
 print(**'我是test函数...'**)  
*#i = outer(test)  
#i()*test()

**区别有参无返回值的**

**def** outer(fn):  
 **def** inner(num1,num2):  
 print(**'功能开始前追加日志信息...'**)  
 fn(num1,num2)  
 print(**'功能结束后追加日志信息...'**)**return** inner  
  
@outer  
**def** mySum(num1,num2):  
 print(num1 + num2)

mySum(1,2)

**#格式二：无参有返回值的装饰器****def** make\_bold(fn):  
 **def** wrapper():  
 **return "<b>"** + fn() + **"</b>"  
 return** wrapper  
  
**def** make\_italic(fn):  
 **def** wrapper():  
 **return "<i>"** + fn() + **"</i>"  
 return** wrapper  
  
@make\_italic  
@make\_bold  
**def** test():  
 **return "hello zsq"**

print(test())

**#格式三：有参有返回值的装饰器****def** zhuangshiqi(fn):  
 **def** wrapper(name,age,sex):  
 print(name,age)  
 fn(name,age,sex)  
 print(sex)  
 **return "abcdefg"  
 return** wrapper  
  
@zhuangshiqi  
**def** test(n,a,s):  
 print(**"我叫：%s，年龄为：%d，性别是：%s"** %(n,a,s))  
  
res = test(sex=**"男"**,name=**"老郭"**,age=30)  
print(res)

*#或者print(test(sex="男",name="老郭",age=30))*

**区别有参无返回值的**

**def** zhuangshiqi(fn):  
 **def** wrapper(name,age,sex):  
 print(name,age)  
 fn(name,age,sex)  
 print(sex) **return** wrapper  
  
@zhuangshiqi  
**def** test(n,a,s):  
 print(**"我叫：%s，年龄为：%d，性别是：%s"** %(n,a,s))  
  
test(sex=**"男"**,name=**"老郭"**,age=30)  
**#格式四：通用装饰器****def** zsq(fn):  
 **def** wrapper(\*args,\*\*kwargs):  
 print(args)  
 fn(\*args,\*\*kwargs)  
 print(kwargs)  
 **return "装饰器不难学"  
 return** wrapper  
  
@zsq  
**def** test(name,age,hobby):  
 print(**'我叫：%s，年龄为：%d，爱好是：%s'** %(name,age,hobby))  
print(test(name=**"杜老师"**,age=35,hobby={**'游戏'**:**'吃鸡'**,**'男性'**:**'小鲜肉'**,**'体育'**:**'举重'**}))