07. 万恶之源-set集合,深浅拷贝以及部分知识点补充

本节主要内容:

1. 基础数据类型补充
2. set集合
3. 深浅拷贝

主要内容:
一. 基础数据类型补充
首先关于int和str在之前的学习中已经讲了80%以上了. 所以剩下的自己看一看就可以了. 我们补充给一个字符串基本操作

```
li = ["李嘉诚", "麻花藤", "黄海峰", "刘嘉玲"]
s = "_".join(li)
print(s)

li = "黄花大闺女"
s = "_".join(li)
print(s)
```

列表:
循环删除列表中的每一个元素

```
li = [11, 22, 33, 44]
for e in li:
    li.remove(e)

print(li)

结果:
[22, 44]
```

分析原因:
for的运行过程. 会有一个指针来记录当前循环的元素是哪一个, 一开始这个指针指向第0个. 然后获取到第0个元素. 紧接着删除第0个. 这个时候. 原来是第一个的元素会自动的变成第0个. 然后指针向后移动一次, 指向1元素. 这时原来的1已经变成了0, 也就不会被删除了.

用pop删除试试看:

```
li = [11, 22, 33, 44]
for i in range(0, len(li)):
    del li[i]

print(li)
```

```
结果: 报错
# i= 0, 1, 2  删除的时候li[0] 被删除之后. 后面一个就变成了第0个.
# 以此类推. 当i = 2的时候. list中只有一个元素. 但是这个时候删除的是第2个 肯定报错啊
```

经过分析发现. 循环删除都不行. 不论是用del还是用remove. 都不能实现. 那么pop呢?

```
for el in li:
    li.pop()    # pop也不行

print(li)
结果:
[11, 22]
```

只有这样才是可以的:

```
for i in range(0, len(li)):    # 循环len(li)次, 然后从后往前删除
    li.pop()

print(li)
```

或者. 用另一个列表来记录你要删除的内容. 然后循环删除

```
li = [11, 22, 33, 44]
del_li = []
for e in li:
    del_li.append(e)

for e in del_li:
    li.remove(e)

print(li)
```

注意: 由于删除元素会导致元素的索引改变, 所以容易出现问题. 尽量不要再循环中直接去删除元素. 可以把要删除的元素添加到另一个集合中然后再批量删除.

dict中的fromkey(),可以帮我们通过list来创建一个dict

```
dic = dict.fromkeys(["jay", "JJ"], ["周杰伦", "麻花藤"])
print(dic)

结果:
{'jay': ['周杰伦', '麻花藤'], 'JJ': ['周杰伦', '麻花藤']}
```

前面列表中的每一项都会作为key, 后面列表中的内容作为value. 生成dict
好了. 注意:

```
dic = dict.fromkeys(["jay", "JJ"], ["周杰伦", "麻花藤"])
print(dic)

dic.get("jay").append("胡大")
print(dic)

结果:
{'jay': ['周杰伦', '麻花藤', '胡大'], 'JJ': ['周杰伦', '麻花藤', '胡大']}
```

代码中只是更改了jay那个列表. 但是由于jay和JJ用的是同一个列表. 所以. 前面那个改了. 后面那个也会跟着改

dict中的元素在迭代过程中是不允许进行删除的

```
dic = {'k1': 'alex', 'k2': 'wusir', 's1': '金老板'}
# 删除key中带有'k'的元素
for k in dic:
    if 'k' in k:
        del dic[k]      # dictionary changed size during iteration, 在循环迭代的时候不允许进行删除操作

print(dic)
```

那怎么办呢? 把要删除的元素暂时先保存在一个list中, 然后循环list, 再删除

```
dic = {'k1': 'alex', 'k2': 'wusir', 's1': '金老板'}
dic_del_list = []
# 删除key中带有'k'的元素
for k in dic:
    if 'k' in k:
        dic_del_list.append(k)

for el in dic_del_list:
    del dic[el]

print(dic)
```

类型转换:
元组 => 列表　　list(tuple)
列表 => 元组　　tuple(list)

list=>str　str.join(list)
str=>list　str.split()

转换成False的数据:
0,'',None,[],(),{},set() ==> False

二. set集合

set集合是python的一个基本数据类型. 一般不是很常用. set中的元素是不重复的.无序的.里面的元素必须是可hash的(int, str, tuple,bool), 我们可以这样来记. set就是dict类型的数据但是不保存value, 只保存key. set也用{}表示

注意: set集合中的元素必须是可hash的, 但是set本身是不可hash得. set是可变的.

```
set1 = {'1','alex',2,True,[1,2,3]} # 报错
set2 = {'1','alex',2,True,{1:2}}  # 报错
set3 = {'1','alex',2,True,(1,2,[2,3,4])}  # 报错
```

set中的元素是不重复的, 且无序的.

```
s = {"周杰伦", "周杰伦", "周星星"}
print(s)

结果:
{'周星星', '周杰伦'}
```

使用这个特性.我们可以使用set来去掉重复

```
# 给list去重复
lst = [45, 5, "哈哈", 45, '哈哈', 50]
lst = list(set(lst))    # 把list转换成set, 然后再转换回list
print(lst)
```

set集合增删改查

1. 增加

```
s = {"刘嘉玲", '关之琳', "王祖贤"}
s.add("郑裕玲")
print(s)
s.add("郑裕玲")    # 重复的内容不会被添加到set集合中
print(s)

s = {"刘嘉玲", '关之琳', "王祖贤"}
s.update("麻花藤")    # 迭代更新
print(s)

s.update(["张曼玉", "李若彤","李若彤"])
print(s)
```

2. 删除

```
s = {"刘嘉玲", '关之琳', "王祖贤","张曼玉", "李若彤"}
item = s.pop()  # 随机弹出一个.
print(s)
print(item)

s.remove("关之琳") # 直接删除元素
# s.remove("马虎疼") # 不存在这个元素. 删除会报错
print(s)

s.clear()    # 清空set集合.需要注意的是set集合如果是空的. 打印出来是set() 因为要和dict区分的.
print(s)    # set()
```

3. 修改

```
# set集合中的数据没有索引. 也没有办法去定位一个元素. 所以没有办法进行直接修改.
# 我们可以采用先删除后添加的方式来完成修改操作
s = {"刘嘉玲", '关之琳', "王祖贤","张曼玉", "李若彤"}
# 把刘嘉玲改成赵本山
s.remove("刘嘉玲")
s.add("赵本山")
print(s)
```

4. 查询

```
# set是一个可迭代对象. 所以可以进行for循环
for el in s:
    print(el)
```

5. 常用操作

```
s1 = {"刘能", "赵四", "皮长山"}
s2 = {"刘科长", "冯乡长", "皮长山"}

# 交集
# 两个集合中的共有元素
print(s1 & s2)  # {'皮长山'}
print(s1.intersection(s2))  # {'皮长山'}

# 并集
```

```
print(s1 | s2)  # {'刘科长', '冯乡长', '赵四', '皮长山', '刘能'}
print(s1.union(s2))     # {'刘科长', '冯乡长', '赵四', '皮长山', '刘能'}

# 差集
print(s1 - s2)  # {'赵四', '刘能'} 得到第一个中单独存在的
print(s1.difference(s2))     # {'赵四', '刘能'}

# 反交集
print(s1 ^ s2)  # 两个集合中单独存在的数据 {'冯乡长', '刘能', '刘科长', '赵四'}
print(s1.symmetric_difference(s2)) # {'冯乡长', '刘能', '刘科长', '赵四'}


s1 = {"刘能", "赵四"}
s2 = {"刘能", "赵四", "皮长山"}

# 子集
print(s1 < s2)  # set1是set2的子集吗? True
print(s1.issubset(s2))

# 超集
print(s1 > s2) # set1是set2的超集吗? False
print(s1.issuperset(s2))
```

set集合本身是可以发生改变的. 是不可hash的. 我们可以使用frozenset来保存数据.
frozenset是不可变的. 也就是一个可哈希的数据类型

```
s = frozenset(["赵本山", "刘能", "皮长山", "长跪"])

dic = {s:'123'} # 可以正常使用了
print(dic)
```

这个不是很常用. 了解一下就可以了

三. 深浅拷贝

```
lst1 = ["金毛狮王", "紫衫龙王", "白眉鹰王", "青翼蝠王"]
lst2 = lst1
print(lst1)
print(lst2)

lst1.append("杨逍")
print(lst1)
print(lst2)

结果:
['金毛狮王', '紫衫龙王', '白眉鹰王', '青翼蝠王', '杨逍']
['金毛狮王', '紫衫龙王', '白眉鹰王', '青翼蝠王', '杨逍']
```

```
dic1 = {"id": 123, "name": "谢逊"}
dic2 = dic1
print(dic1)
print(dic2)

dic1['name'] = "范瑶"
print(dic1)
print(dic2)

结果:
{'id': 123, 'name': '谢逊'}
{'id': 123, 'name': '谢逊'}
{'id': 123, 'name': '范瑶'}
{'id': 123, 'name': '范瑶'}
```

对于list, set, dict来说, 直接赋值. 其实是把内存地址交给变量. 并不是复制一份内容. 所以. lst1的内存指向和lst2是一样的. lst1改变了, lst2也发生了改变

浅拷贝

```
lst1 = ["何炅", "杜海涛","周渝民"]
lst2 = lst1.copy()
lst1.append("李嘉诚")
print(lst1)
print(lst2)
print(id(lst1), id(lst2))

结果:
两个lst完全不一样. 内存地址和内容也不一样. 发现实现了内存的拷贝

lst1 = ["何炅", "杜海涛","周渝民", ["麻花藤", "马芸", "周笔畅"]]
lst2 = lst1.copy()
lst1[3].append("无敌是多磨寂寞")
print(lst1)
print(lst2)
print(id(lst1[3]), id(lst2[3]))

结果:
['何炅', '杜海涛', '周渝民', ['麻花藤', '马芸', '周笔畅', '无敌是多磨寂寞']]
['何炅', '杜海涛', '周渝民', ['麻花藤', '马芸', '周笔畅', '无敌是多磨寂寞']]
4417248328 4417248328
```

浅拷贝. 只会拷贝第一层. 第二层的内容不会拷贝. 所以被称为浅拷贝

深拷贝

```
import copy
```

```
lst1 = ["何炅", "杜海涛","周渝民", ["麻花藤", "马芸", "周笔畅"]]
lst2 = copy.deepcopy(lst1)
lst1[3].append("无敌是多磨寂寞")
print(lst1)
print(lst2)
print(id(lst1[3]), id(lst2[3]))

结果:
['何炅', '杜海涛', '周渝民', ['麻花藤', '马芸', '周笔畅', '无敌是多磨寂寞']]
['何炅', '杜海涛', '周渝民', ['麻花藤', '马芸', '周笔畅']]
4447221448 4447233800
```

都不一样了. 深度拷贝. 把元素内部的元素完全进行拷贝复制. 不会产生一个改变另一个跟着改变的问题

补充一个知识点:

最后我们来看一个面试题:

```
a = [1, 2]
a[1] = a
print(a[1])
```