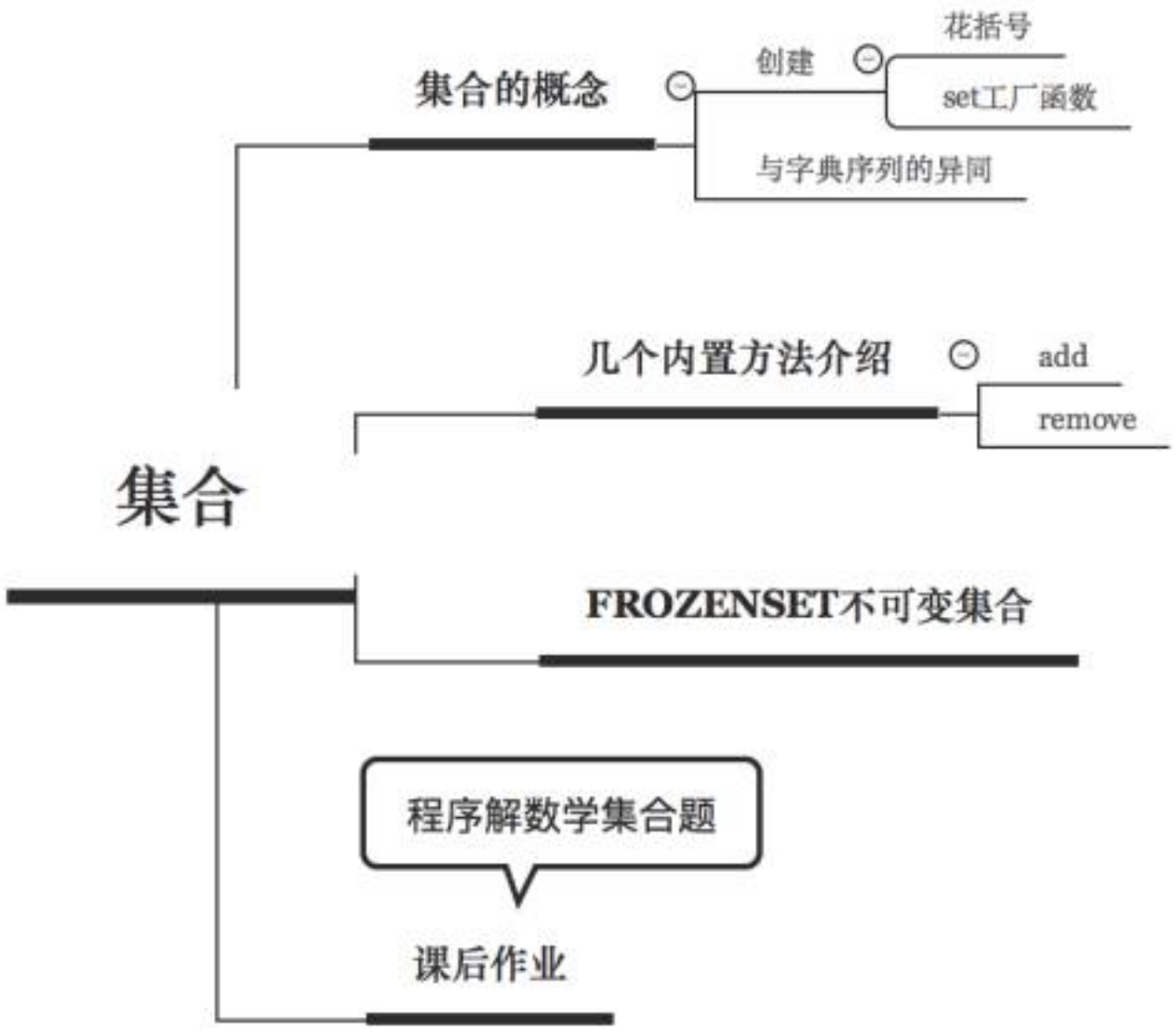
python集合－程序与数学的千丝万缕

Original 2016-11-20 哈哈小 菜猿观世界

通过本节的学习，你将了解一下内容：



当序列不好用时，我们想到了字典。除了字典，还有一位好伙计值得我们提及。他少了序列字典的多元接纳，却有着自己的沉稳谨慎，也是魅力不一般。

一，集合的概念

集合，提到这个词，满脑子最先蹦出的可能是，子集，真子集，补集，韦恩图等等概念，因为曾经大把的青春时光没少跟它们相处过。

我们从时间纵向的角度去分析，基础数学集合概念十九世纪就已诞生，而计算机科学直到上个世纪末才蓬勃

发展。所以，我们可以拍胸脯，信心十足地认为，这里将要讲到的集合与数学集合的关系必然千丝万缕。

为了更好地推进下文，看来有必要对数学集合的概念做一个简单的回顾。

由一个或多个元素所构成的叫做集合。若*x*是集合*A*的元素，则记作***x*∈*A***。集合中的元素有三个特征：

1.确定性（集合中的元素必须是确定的）
2.互异性（集合中的元素互不相同。例如：集合A={1，a}，则a不能等于1）
3.无序性（集合中的元素没有先后之分）

好样的，有此概念助力，简直要省好多事啊。

1.与序列字典的异同

相同点，元素的类型可以是多样的，各种类型都可以成为集合的元素。除此，同为python的数据结构，为python语言的简洁强大提供了强有力的保障。如果混为使用，并用到恰到好处，必定力大无穷，大放异彩。用得恰到好处，是决策能力，算法优良的体现。所以可以这么说，你掌握了数据结构与算法，你就掌握了编程。

不同点，数学集合的三大特性，也真是这里要讲到的python集合的三大特性，确定性无需多讲，互异性（元素互不相同）保证了集合中的元素都是不同的，这一点与字典稍有不同，字典只是明确了键值的不同。在无序性（元素没有先后之分）上，同字典，之所以不把它归之于序列，正是因为这一点。总之一句话，字典与集合稍有不同，但与序列却是大有不同。

通过上段文字的描述，如果你还是不甚理解。没关系，通过接下来一个又一个的例子的展示，定能让你扒开雾霾见天明的。

2.创建

第一种方式，直接用花括号，简单直接又暴力。

```
>>> w1={1,2,3,4}
```

```
>>> w1
{1, 2, 3, 4}
>>> w1={1,2,3,4,4}#眼尖的有发现吧，我有重复元素4
>>> w1#想得美，你是集合君，你就得不同
{1, 2, 3, 4}
>>>
```

第二种方式，使用工厂函数set，有范还是这个正。

```
>>> help(set)
Help on class set in module builtins:
class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
```

（未完全显示）

我们试试第二种方式，传入一个可迭代对象，new出一个集合对象。

```
>>> w2=set([1,2,3,4,4])
>>> w2
{1, 2, 3, 4}
>>>
```

这里哈哈小要抛出一个问题了，现在有一带重复元素的列表

list1=['h','a','h','a','x','i','a','o']

现在要把它变成一个不带相同元素的列表list2，该如何做呢?

（停，先别往下翻，答案在下面，如果两分钟之后还没有结果，那就请自便）

这里哈哈小提供两种参考：

way1

```
>>> list2=list(set(list1))
>>> list2
['a', 'h', 'x', 'o', 'i']
>>> list2
['a', 'h', 'x', 'o', 'i']
>>>
```

way2

```
>>> list2=[]
>>> list1=['h','a','h','a','x','i','a','o']
>>> for each in list1:
        if each not in list2:
            list2.append(each)

>>> list2
['h', 'a', 'x', 'i', 'o']
>>>
```

二，几个内置方法

哈哈小表示，学习python，最需要使用的两个函数是dir和help，有事没事来两下。

python好学的一个原因是，当在编写程序忘记想要调用方法的名称时，dir和help两个方法基本能在不通过互联网的情况下帮你解决当时之困惑（不过这玩意都是外国来的，如果还有一项好的英文阅读技能，那简直就是如虎添翼啊）

```
>>> dir(set)
['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__for-
mat__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__iand__', '__init__', '__ior__',
'__isub__', '__iter__', '__ixor__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__',
'__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__',
'__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor__', 'add', 'clear',
'copy', 'difference', 'difference_update', 'discard', 'intersection', 'intersection_up-
date', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference',
'symmetric_difference_update', 'union', 'update']
>>>
```

好像set的内置函数也不少，从函数的名字，像difference（补集），union（并集），isdisjoint（是互异），issubset（是子集）等等这些我们连同数学集合的概念，使用起来应该也不难。

在这里，哈哈小只稍提几个。

通过某种途径，哈哈小弄到了下面这样一个号码本：

```
addrbook={"188xxxx5930","188xxxx5338","188xxx4930"}
```

> 增加一位手机号码为188xxxx3570的用户

```
>>> addrbook.add('188xxx3570')
>>> addrbook
{'188xxxx5338', '188xxx3570', '188xxxx5930', '188xxx4930'}
>>>
```

> 删除手机尾号为5930的用户

```
>>> addrbook.remove('188xxxx5930')
>>> addrbook
```

{'188xxxx5338', '188xxx3570', '188xxx4930'}
>>>

> 再随机删除一位用户

```
>>> addrbook.pop()
'188xxxx5338'
>>> addrbook
{'188xxx3570', '188xxx4930'}
>>>
```

通过上面例子的描述，想必大家已经知道add，remove，pop方法的使用了。

在这里，有一个函数，哈哈下要特别照顾一下，copy函数。

想看看它是何方神圣。

```
>>> help(set.copy)
Help on method_descriptor:
copy(...)
    Return a shallow copy of a set.
>>>
```

返回一个集合的shallow copy，shallow copy这个词很新鲜啊，如果说要给它一个英文名字的话，浅复制是合适的。

什么是浅复制呢?

现在我们要复制出一份号码本给addrbook_copy，我们一般会是这样做

```
>>> addrbook={'188xxx3570', '188xxx4930'}
>>> addrbook_copy=addrbook
```

```
>>> addrbook_copy
{'188xxx3570', '188xxx4930'}
```

是没有问题的，既然这样做，是可以达到复制的目的，那现在又来个copy函数，我要它又有何用?

别着急，我们马上就能发现它们的不同，浅复制，我们不妨也先依葫芦画瓢一下。

```
>>> addrbook_shallow_copy=addrbook.copy()
>>> addrbook_shallow_copy
{'188xxx3570', '188xxx4930'}
>>>
```

结果一样，看不出任何的不同，去删除一个元素试试

```
>>> addrbook_shallow_copy.pop()
'188xxx3570'
>>> addrbook
{'188xxx4930'}#浅复制不会改变addrbook的状态
>>> addrbook_shallow_copy
{'188xxx4930'}
>>>

>>> addrbook_copy.pop()
'188xxx3570'
>>> addrbook
{'188xxx4930'}#深复制改变了addrbook的状态
>>> addrbook_copy
{'188xxx4930'}
>>>
```

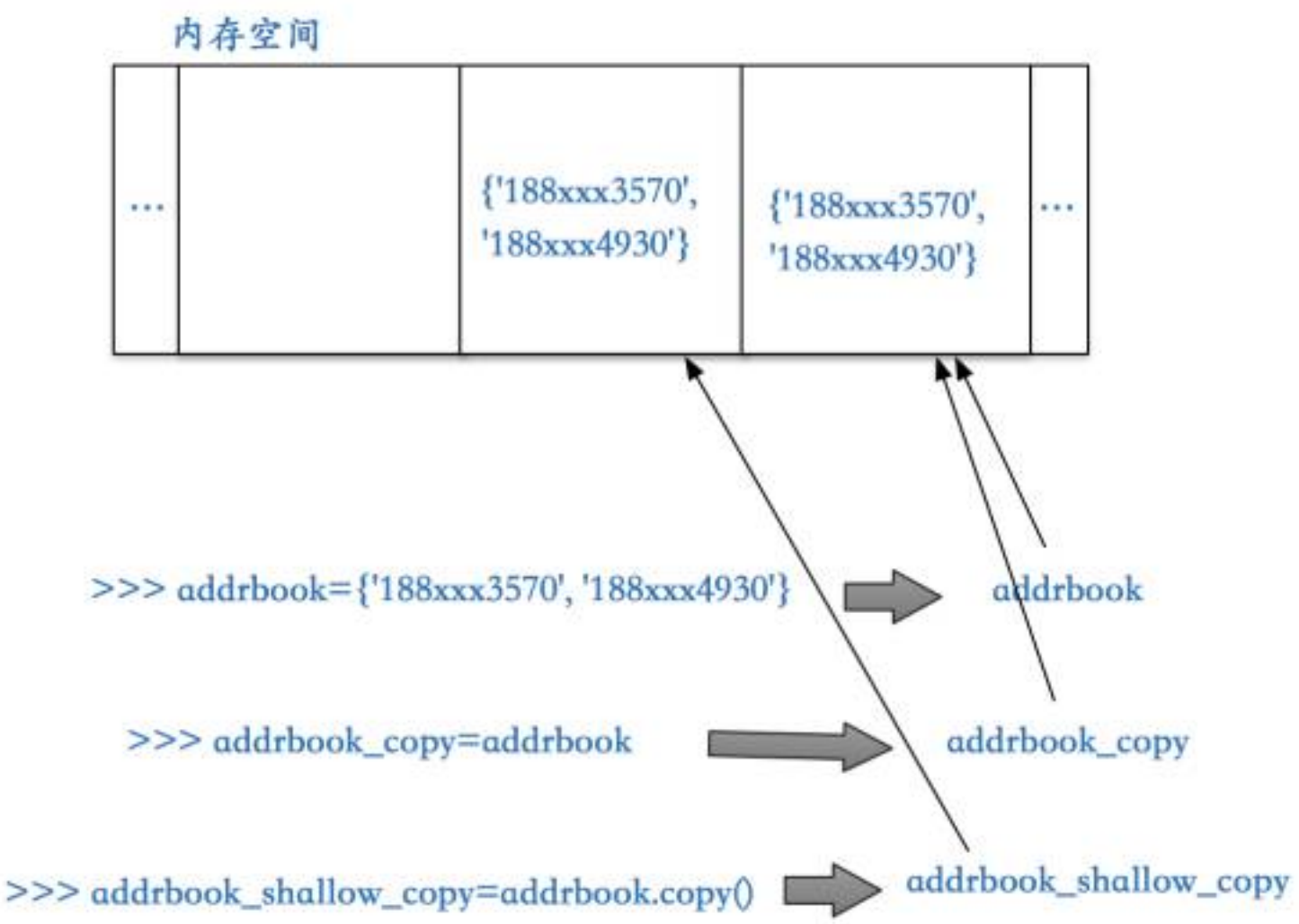
这是什么原因呢？内存分配机制使然。

其实之前讲变量贴标签的时候，哈哈小就提及过，由于这个知识点的掌握对以后的编程会起到重要作用，哈哈小有必要再提一遍。

浅复制就好像，我的内存空间的数据拷贝一份给你，然后你拿去再找块地存好，自此咱两毫不相干。虽然咱两的数据内容是一样的，但是各自管辖的土壤是平等独立，毫无约束牵绊的。所以你删你的，与我无干，我依然保持我原有的状态。

深复制，就是简单粗暴的a＝b这样的复制，这种情况不同在于，他们都指向了同一块内存空间，相互依赖，相互约束。你删除一元素，我也得跟着变。

看下图，也许哈哈小的话能更明白些：



三，frozenset不可变集合

froezenset，冰冻集合?

是的，冰冻了，就动不了了。

> 申明一个不可变集合

```
>>> frozen=frozenset([1,2,3])
```

> 我加，加，加

```
>>> frozen.add(0)
Traceback (most recent call last):
  File "<pyshell#107>", line 1, in <module>
    frozen.add(0)
AttributeError: 'frozenset' object has no attribute 'add'
```

> 我删，删，删

```
>>> frozen.pop()
Traceback (most recent call last):
  File "<pyshell#108>", line 1, in <module>
    frozen.pop()
AttributeError: 'frozenset' object has no attribute 'pop'
>>>
```

不可变集合，增删都行不通。

四，课后作业

衡南二中高一年级，某集合章节数学试卷有这么一道题：

已知集合A ＝ {0，2，4，...，20} ，B＝ {x | x＝k^2，k为1，2，3，4，5} ，求B 在A 中的补集。

用python来完成此题，哈哈小的思路是这样的：

A 集合0到20的所有偶数可以这样表示

```
set(range(0,21,2))
```

{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

B集合可以这样表示

set(map(lambda x: x*x, range(1,6)))

{16, 1, 4, 9, 25}

所以B 在A 中的补集的补集为：

```
>>> set(range(0,21,2)).difference(set(map(lambda x: x*x, range(1,6))))
{0, 2, 6, 8, 10, 12, 14, 18, 20}
>>>
```

除了这道，还有以下这些：

1、设U={0，1，2，3，4}，A ={0，1，2，3}，B={2，3，4}，则（CuA）∩（CuB）=_____。

2、若U={1，2，3，4}，M={1，2}，N={2，3}，则

C∪（M∪N）= _____。

3、若A={-2，2，3，4}, B={x｜x=t^2，t∈A}，用列举法表示B_____。

记住，你要用程序方式把这些题完成哦。