列表、元组、字典、集合

In [1]:
```
#设置全部行输出
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

当我们有一百个变量要储存起来的时候,总不能弄一百个变量啊,怎么办的,用列表.

我们就把它当成一个大桶，当我们有一堆东西需要找个地方临时存放在一起，以便后续进行排序，筛选等操作，就弄一个列表，先放进去。

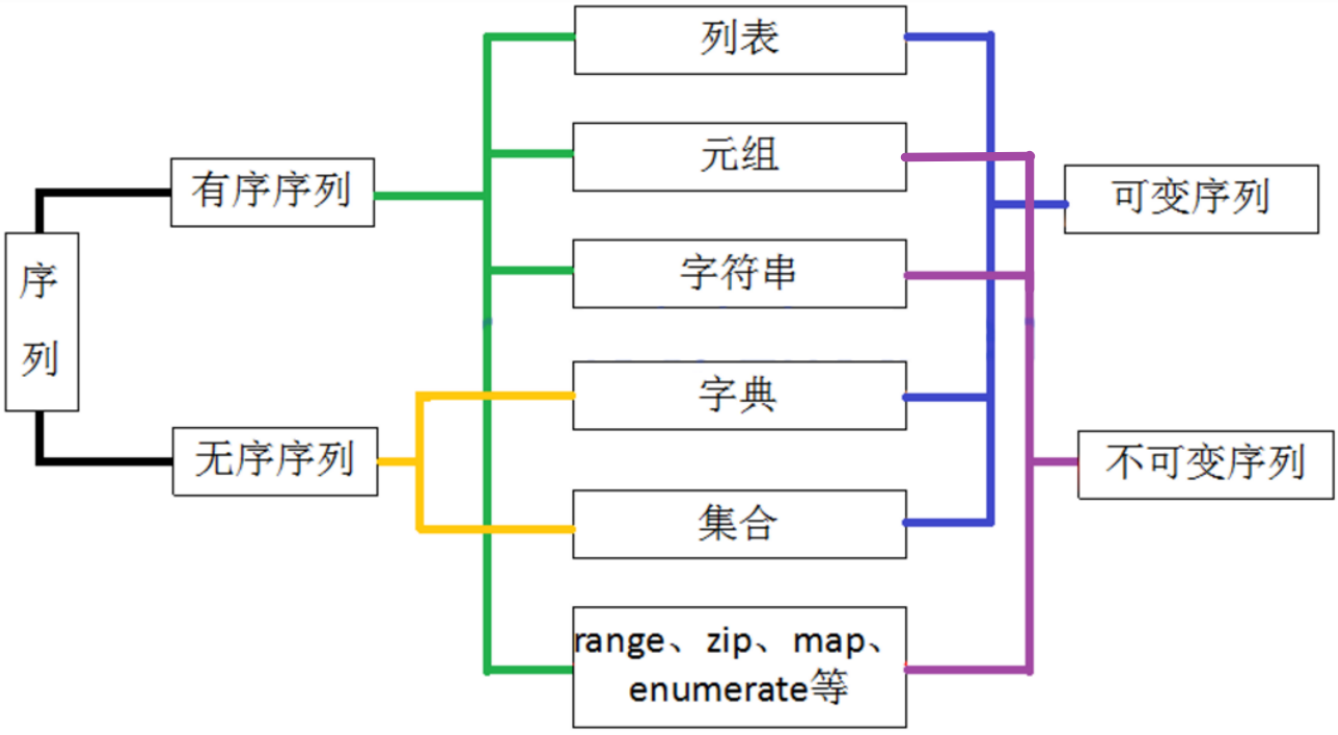
1 列表

- 列表是Python内置可变序列之一，是包含若干元素的有序连续内存空间。
- 列表元素放在一对中括号中，每个元素用逗号隔开，每个元素类型可以不同，没有长度限制。
- 当列表元素增加或删除时，列表对象自动进行扩展或收缩内存，保证元素之间没有缝隙（自动内存管理）。

In [10]:
```
list01 = [1, 2,[55,66],'on','go',True,False,None]
list01
```

1.1 列表的属性

- 列表和字符串一样，都属于序列。
- 序列的索引、切片、相加、乘法、判断元素存在否、计算长度、最大小值，这些操作列表都支持。



1.2 列表索引与切片

- 列表和字符串一样,是一种可迭代对象.
- 因此列表可以和字符串一样进行索引和切片.语法方面是一样的.
- 语法:
  - **列表[起始位置:终止位置:步长]**
  - 包含起始位置,不包含终止位置

In [2]:
```
list_11 = [1,2,[33,55],4]
```

In [21]:
```
list_11[0]

list_11[1]

list_11[2:4]
```

Out[21]: 1

Out[21]: 2

Out[21]: [[33, 55], 4]

1.3 练习1

有多少种方法截取 ["a",1,True,[33,5],6,8] 中的[6,8]?

In [3]:
```

```

Out[3]: [6, 8]

Out[3]: [6, 8]

Out[3]: [6, 8]

Out[3]: [6, 8]

思考：怎么以切片的形式来取出33？

In [4]:

```
```

Out[4]: 33

1.4 列表元素修改

通过元素的索引位置来修改元素

In [11]:

```
list_11 = [1,2,[33,55],4]
list_11

list_11[1]=11
list_11
```

Out[11]: [1, 2, [33, 55], 4]

Out[11]: [1, 11, [33, 55], 4]

思考：如何将列表list_11中55修改成88？

In [5]:

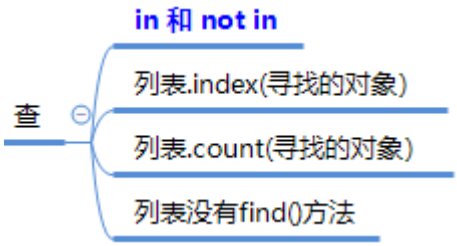
```
list_11 = [1,2,[33,55],4]
list_11[2][1]=88
list_11
```

Out[5]: [1, 2, [33, 88], 4]

1.5 列表常用方法

1.5.1 列表元素检索

- in 和 not in
- 列表.index(寻找的对象，开始索引，结束索引)
- 列表.count(寻找的对象)



（1）判断元素存在否

- in 和 not in

In [43]:
```
a=[1,'a', [2, 5]]
1 in a
```

Out[43]: False

思考：2是否在列表a中？

In [44]:
```
a=[1,'a', [2, 5]]
2 in a
```

Out[44]: False

In [45]:
```
a=[1,'a', [2, 5]]
[2, 5] in a
```

Out[45]: True

（2）元素首次出现的索引

- **列表.index(寻找的对象，开始索引，结束索引)**

  获得某个元素首次出现的索引

In [8]:
```
a=['我','爱','北','京','天','安','门','天','安','门',]

a.index("门")
```

Out[8]: 6

In [10]:
```
a.index("门",1,10)
```

Out[10]: 6

注意：列表没有方法find()

（3）元素出现次数

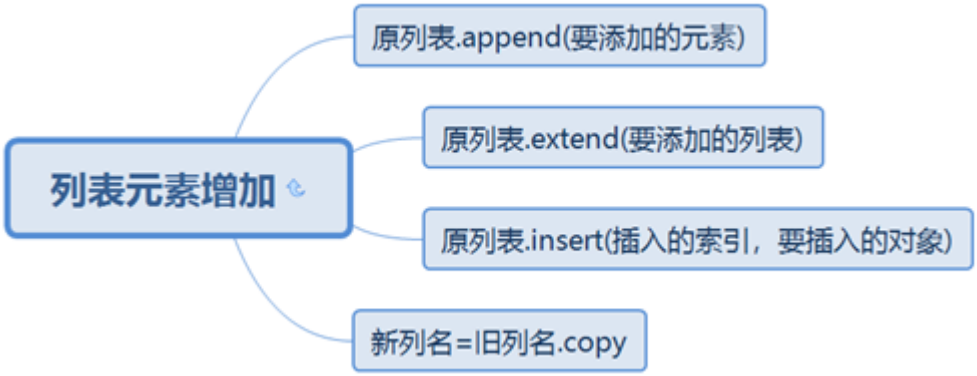
- **列表.count(寻找的对象)**

  获得某个元素元素出现次数

In [2]:
```
a=['我','爱','北','京','天','安','门','天','安','门',]

a.count("门")
```

Out[2]: 2

1.5.2 列表元素添加



（1）append

语法：

原列表.append(要添加的元素)

In [23]:
```
L = ['Superman','Hulk','Spiderman']

L.append('Leifengxia')    #只能添加一个元素

L
```

Out[23]: ['Superman', 'Hulk', 'Spiderman', 'Leifengxia']

In [8]:
```
a = [1,2]

b = [3,4]

a.append(b)

a
```

Out[8]: [1, 2, [3, 4]]

（2）extend

语法：

原列表.extend(要添加的列表)

In [24]:
```
c=[1]

c.extend([88,99])

c
```

Out[24]: [1, 88, 99]

（3）insert

语法：

insert(插入的索引，要插入的对象)

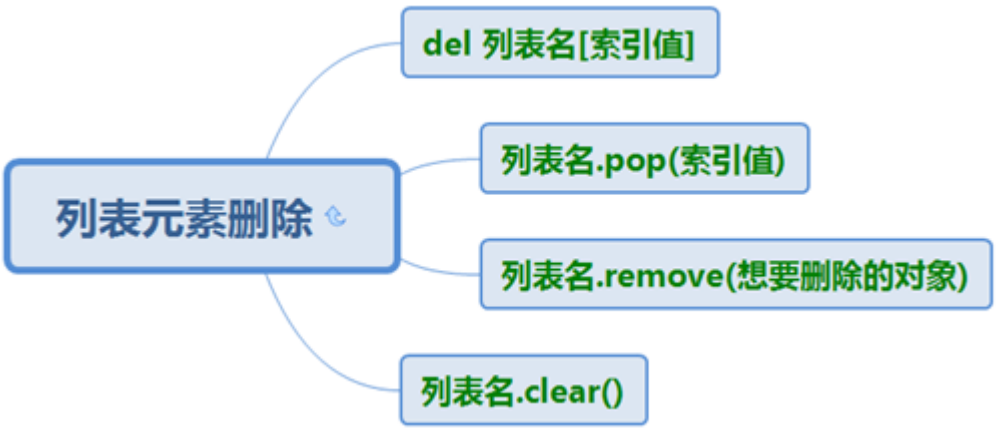
In [6]:
```
a = [0, 1, 2]

a.insert(0, "哈")

a
```

Out[6]: ['哈', 0, 1, 2]

思考：空列表可不可以插入？

In [11]:
```
a=[]
a.insert(300,"哈哈")
```

1.5.3 列表元素删除



（1）del 列表名[索引值]

根据下标进行删除

In [42]:
```
a=["a","b","c","d","e"]

del a[2]

a
```

Out[42]: ['a', 'b', 'd', 'e']

In [3]:
```
a=["a","b","c","d","e"]
del a[1:3]
a
```

Out[3]: ['a', 'd', 'e']

In [36]:
```
del a
```

In [37]:
```
a
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-37-3f786850e387> in <module>()
----> 1 a

NameError: name 'a' is not defined
```

（2）列表名.pop(索引值)

返回的是你弹出的那个索引值的元素。

In [5]:
```
a = [11,22,33,43]
b = a.pop(1)
a
b
```

Out[5]: 22

a.pop()则会弹出最后一个值

In [47]:
```
a = [11,22,33,43]
a
a.pop()
a
```

Out[47]: [11, 22, 33, 43]

Out[47]: 43

Out[47]: [11, 22, 33]

（3）列表名.remove(元素名)

用于移除列表中某个值的**第一个**匹配项

In [12]:
```
list_15=["a","b","c","d","e","c"]

list_15.remove("c")

list_15
```

Out[12]: ['a', 'b', 'd', 'e', 'c']

In [65]:
```
list15=["a","b","c","d","e","c"]

list15.clear()

list15
```

Out[65]: []

（4）列表名.clear()

清空列表所有元素。

In [71]:
```
a=["a","b","c","d","e"]

a.clear()

a
```

Out[71]: []

1.5.4 练习2

（1）aa=[1, 'a', [2, 5], 88, 99, '倒数第二', '倒数第一'] 判断字符串"a"是否在列表aa当中

In [13]:
```

```

Out[13]: True

（2）删除列表aa中索引值为3的元素

In [14]:
```

```

Out[14]: 88

Out[14]: [1, 'a', [2, 5], 99, '倒数第二', '倒数第一']

（3）删除列表aa的元素[2,5]中的5

In [15]:
```

```

Out[15]: [1, 'a', [2], 88, 99, '倒数第二', '倒数第一']

1.5.5 列表元素排序

（1）降序和升序排列

语法：列表.sort(reverse=True/False)

- 此方法会将列表按特定顺序重新排列
- 括号内的参数可以不写，默认升序排列
- reverse的英文意思是翻转，如果填写了reverse=True就意味着列表会降序排列

In [64]:
```
a = [1, 4, 2, 3]
a.sort()
a
```

Out[64]: [1, 2, 3, 4]

In [68]:
```
a = [1, 4, 2, 3]
a.sort(reverse=False)
a
```

Out[68]: [1, 2, 3, 4]

In [69]:
```
a.sort(reverse=True)
a
```

Out[69]: [4, 3, 2, 1]

（2）单纯的逆置

语法： 列表.reverser()将list逆置

In [67]:
```
a = [1, 4, 2, 3]
a.reverse()
a
```

Out[67]: [3, 2, 4, 1]

1.5.6 列表的复制

语法：

新列名=旧列名.copy

In [6]:
```
a=[1,"a",[2,3]]

b= a.copy()

b
```

Out[6]: [1, 'a', [2, 3]]

In [7]:
```
a[2][1]=33
a
b
```

Out[7]: [1, 'a', [2, 33]]

Out[7]: [1, 'a', [2, 33]]

In [9]:
```
import copy
a=[1,"a",[2,3]]
b = copy.deepcopy(a)     #深复制

a[2][0]=22

a
b #由于是深复制，所以c没有跟随a变化
```

Out[9]: [1, 'a', [22, 3]]

Out[9]: [1, 'a', [2, 3]]

1.6 列表常用的操作符

（1）比较操作符

- 列表间做比较 默认是从第一个元素开始比较，一旦有一个元素大了，则这个列表比另一个列表大
- 如果比较到两个数据类型不一致时，程序会报错

In [203]:
```
list1 = [1, 2, 3]

list2 = [1, 2, 2]

list1 > list2
```

Out[203]: True

In [205]:
```
list1 = [1, 2, 3]
list3 = ['00', 2, 3]
list1 > list3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-205-f7d87dbfa9b8> in <module>()
      1 list1 = [1, 2, 3]
      2 list3 = ['00', 2, 3]
----> 3 list1 > list3

TypeError: '>' not supported between instances of 'int' and 'str'
```

（2） ‘+’ 连接操作符

- 列表的 ‘+’ 操作符也是连接操作符，它允许把多个列表对象合并起来，
- 其实相当于extend方法，但是extend是在原来的列表基础上扩展，‘+’操作符是生成一个新的列表

In [212]:
```
list1 = [1, 2, 3]

list2 = [4, 5, 6]

list1 + list2
```

Out[212]:
```
[1, 2, 3, 4, 5, 6]
```

（3）’ *’ 重复操作符

In [216]:
```
list1 = [666, '666', 'fff']

list1 * 3
```

Out[216]:
```
[666, '666', 'fff', 666, '666', 'fff', 666, '666', 'fff']
```

1.7 列表其他统计BIF

我们之前介绍了序列的相关BIF，对于列表同样也是适用的。

- len()： 返回序列包含的元素的个数
- max()： 返回序列中最大元素
- min()： 返回序列中最小元素
- str()： 将序列转为字符串
- sum()： 计算元素和
- zip()： 将多个列表对应位置的元素组合为元祖，函数返回可迭代的zip对象
- enumerat()：枚举列表元素，返回枚举对象，其中每个元素为包含下标和值的元组

In [24]:
```
a=[1,2,3,4,7]

len(a)

max(a)

min(a)

sum(a)
```

Out[24]:
```
5
```

Out[24]:
```
7
```

Out[24]:
```
1
```

Out[24]:
```
17
```

In [31]:
```
a=["超人","闪电侠","绿箭侠"]

b=["哨兵","快银","鹰眼"]

c=list(zip(a,b))

for i in c:
    print(i)
```

```
('超人', '哨兵')
('闪电侠', '快银')
('绿箭侠', '鹰眼')
```

In [8]:
```
a=["超人","闪电侠","绿箭侠"]

c=enumerate(a)
# d=list(enumerate(a,start=1))

for i  in c:
    print(i)
```

```
(0, '超人')
(1, '闪电侠')
(2, '绿箭侠')
```

1.8 列表推导式

(学完条件判断语句再回头复习这一节)

如果我们想要生成一个由1 - 10的数字组成的列表,都有什么办法呢?

In [42]:
```
a = [1,2,3,4,5,6,7,8,9,10]
a
```

Out[42]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

range(start, stop[, step])

- start: 计数从 start 开始。默认是从 0 开始。例如range（5）等价于range（0， 5）；
- end: 计数到 end 结束，但不包括 end。例如：range（0， 5） 是[0, 1, 2, 3, 4]没有5
- step：步长，默认为1。例如：range（0， 5） 等价于 range(0, 5, 1)

Python3.x 中 range() 函数返回的结果是一个整数序列的对象，而不是列表。

In [4]:
```
range(10)
```

Out[4]:
```
range(0, 10)
```

不是列表，但是可以利用 list 函数返回列表，即：

In [9]:
```
list(range(10))
```

Out[9]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [47]:
```
list(range(1,11,2))
```

Out[47]: [1, 3, 5, 7, 9]

如果不用list(),还可以用迭代来在[ ]内实现

In [23]:
```
[i for i in range(10)]
```

Out[23]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

（1）生成指定范围数值列表

语法：［元素运算式 for i in 遍历对象］

In [6]:
```
[i+1 for i in range(10)]
```

Out[6]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

上面的range(10)可以换成字符串、列表、元组等。

In [ ]:
```

```

（2）生成选择符合条件的列表

语法：[元素运算式 for i in 可遍历对象 if 条件判断语句]

In [13]:
```
[i for i in range(10,20) if i%2==0]
```

Out[13]: [10, 12, 14, 16, 18]

1.9 练习3

(1)
生成一个列表，列表元素从1取到10，步长为1，然后所有元素的数值都乘以2

In [20]:
```

```

Out[20]: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

(2)

将列表[2,6,61,64,77,611,6461,75]中的元素转化为字符串格式

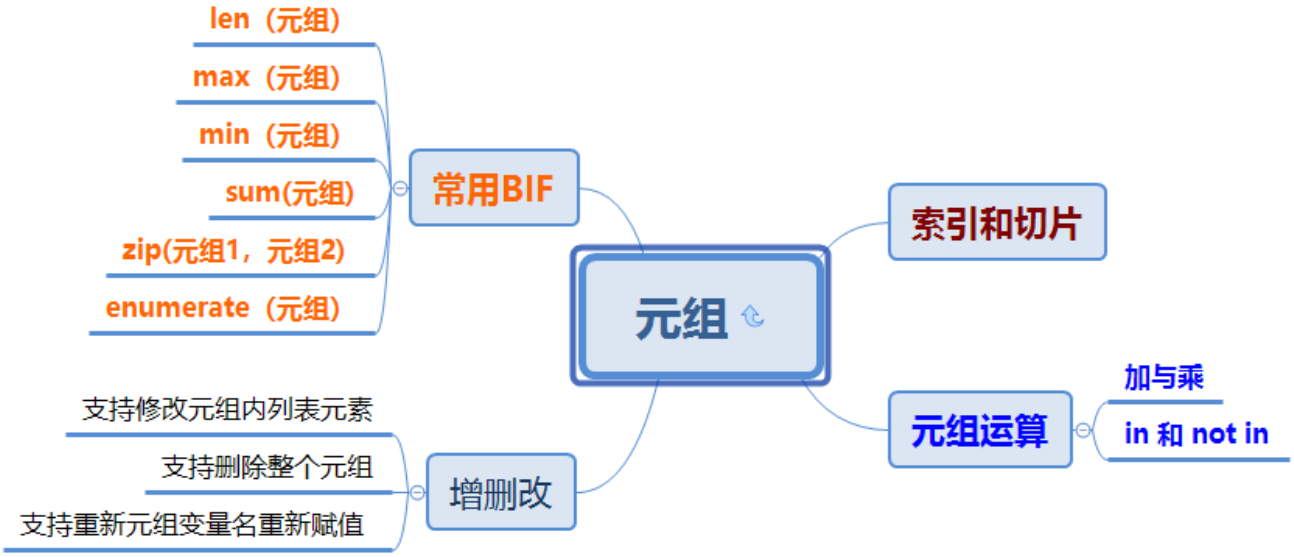
In [18]:
```
```

Out[18]: ['2', '6', '61', '64', '77', '611', '6461', '75']

2 元组(Tuple)

元组是一种序列，但是元组的元素不能更改。 元组的元素可以是任何类型的数据

- 元组的适用场景：
  - 元组比列表操作速度要快，适合遍历。
  - 如果数据不需要被修改，要“保护起来”，那么可以适用元组。
  - 由于不可变属性，在很多方法和操作中只能用元组作为结构的一部分，比如字典中的key。



2.1 元组的属性

2.1.1 创建元组

In [63]:
```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup1

tup3 = "a", "b", "c", "d"
tup3
```

Out[63]: ('physics', 'chemistry', 1997, 2000)

Out[63]: ('a', 'b', 'c', 'd')

2.1.2 删除元组

元组中的元素值是不允许删除的，但我们可以使用del语句来删除整个元组:

In [67]:
```
tup = ('physics', 'chemistry', 1997, 2000)
tup
```

Out[67]:
```
('physics', 'chemistry', 1997, 2000)
```

In [69]:
```
del tup
```

In [70]:
```
tup
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-70-95b80b2375ef> in <module>()
----> 1 tup

NameError: name 'tup' is not defined
```

2.1.3 修改元组（不能）

元组的元素不能修改的

In [193]:
```
a=2

aTuple=('et', 77, a,[1,2],(88,99))

aTuple[0] = 1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-193-828989468b39> in <module>()
      1 a=2
      2 aTuple=('et', 77, a,[1,2],(88,99))
----> 3 aTuple[0] = 1

TypeError: 'tuple' object does not support item assignment
```

元组的元素不能修改的,但是元组可以重新赋值

In [197]:
```
aTuple=('et', 77, 2,[1,2],(88,99))

aTuple = aTuple[:3]

aTuple
```

Out[197]:
```
('et', 77, 2)
```

2.2 元组的陷阱

陷阱一：单元素元组的创建

In [198]:
```
tp = (1)
tp
```

Out[198]: 1

如果想要定义一个元素的元组

In [2]:
```
tp = (1,)
tp
```

Out[2]: (1,)

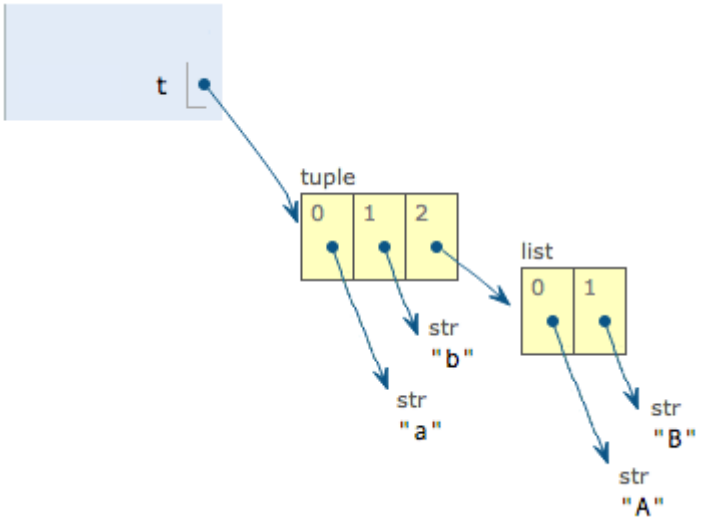
陷阱二：元组中的列表其实可变

In [200]:
```
tp = ('a', 'b', ['A', 'B'])

tp
```

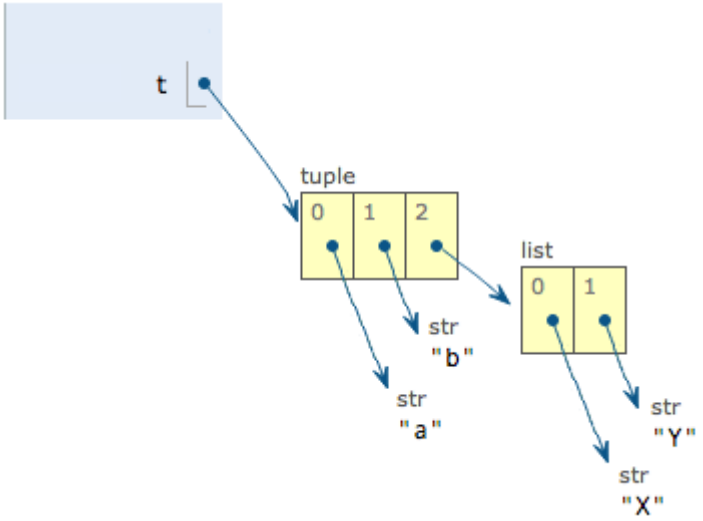
Out[200]: ('a', 'b', ['A', 'B'])

In [201]:
```
tp[2][0] = 'X'

tp[2][1] = 'Y'

tp
```

Out[201]: ('a', 'b', ['X', 'Y'])



当我们把list的元素'A'和'B'修改为'X'和'Y'后，tuple变为：



- 表面上看，元组的元素确实变了，但其实变的不是元组的元素，而是列表的元素。
- 元组一开始指向的列表并没有改成别的列表，所以，元组所谓的“不变”是说，元组的每个元素，指向永远不变。即指向'a'，就不能改成指向'b'，指向一个列表，就不能改成指向其他对象
- 但指向的这个列表本身是可变的

2.3 元组常用方法

（1）元组的索引

```
In [7]: a=(1,2,"哈",[55,88])

        a[3][1]

        a[-2]

        a[1:4]
```

```
Out[7]: 88

Out[7]: '哈'

Out[7]: (2, '哈', [55, 88])
```

（2）元组的相加

```
In [9]: (1,2)+(2,1)
```

```
Out[9]: (1, 2, 2, 1)
```

（3）元组的乘法

In [11]:
```
(2,'哈')*2
```

Out[11]:
```
(2, '哈', 2, '哈')
```

（4）判断元素存在否

In [13]:
```
2 in (2,'哈')
```

Out[13]:
```
True
```

我们之前介绍了序列的相关BIF，对于元组同样也是适用的。

- len()： 返回序列包含的元素的个数
- max()： 返回序列中最大元素
- min()： 返回序列中最小元素
- str()： 将序列转为字符串
- sum()： 计算元素和
- zip()： 将多个列表对应位置的元素组合为元祖，函数返回可迭代的zip对象
- enumerat()：枚举列表元素，返回枚举对象，其中每个元素为包含下标和值的元组

思考：元组内的元素可以修改吗？可以删除吗？

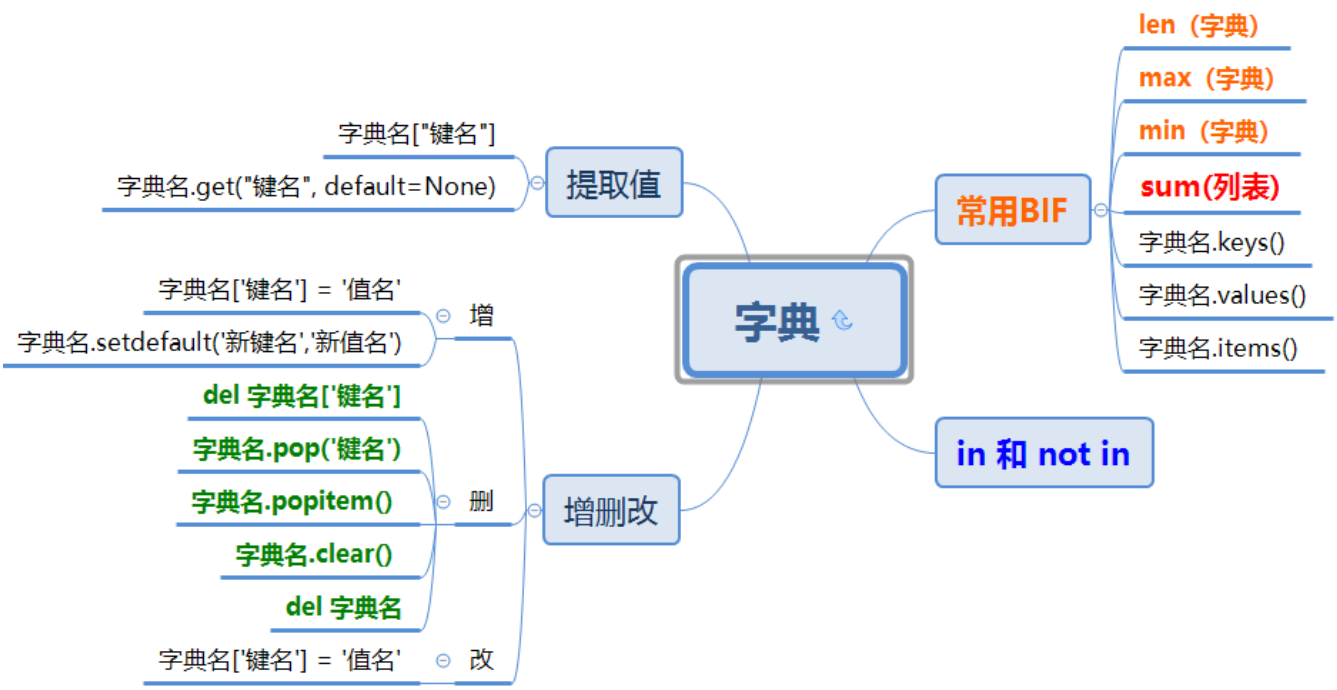
思考：元组内的元素如果是列表，列表里面的元素可以修改吗？

3 字典dict

- Python内置了字典：dict的支持，dict全称dictionary，在其他语言中也称为map，使用键-值（key-value）存储，具有极快的查找速度。
- 举个例子，我们要根据学生的学号查找对应的成绩，如果用list实现，需要两个list：

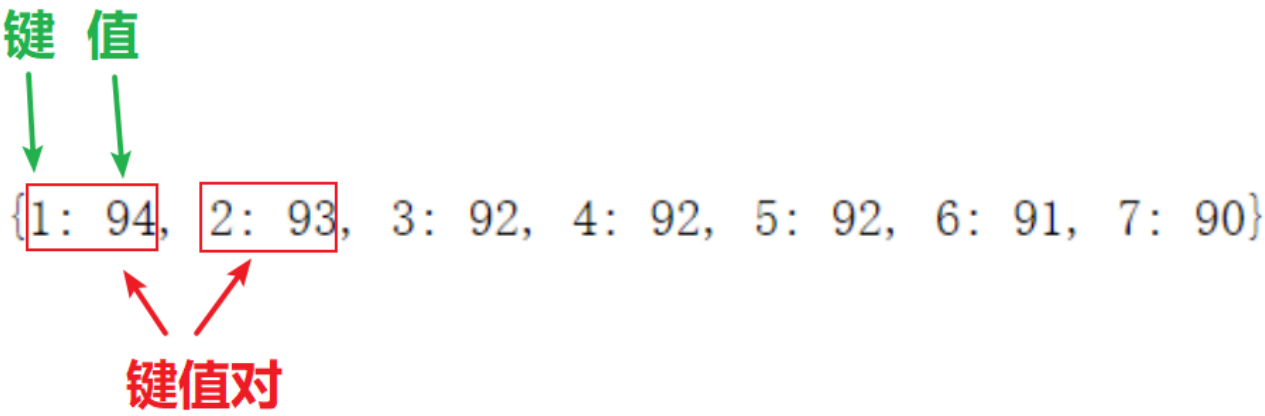
```
ids = [1,2,3,4,5,6,7]
scores = [94, 93, 92, 92, 92, 91, 90]
```

给你一个学生学号，要查找对应的成绩，就先要在id中找到对应的位置，再从scores取出对应的评分，list越长，耗时越长。



3.1 字典的创建

- 字典由键（key）和对应值（value）成对组成。基本语法如下：



- 每个键与值用冒号隔开（:），每对用逗号，每对用逗号分割，整体放在花括号中（{}）。
- 键必须独一无二，但值则不必。
- 如果用dict满足上面的查找需求，只需要一个“名字”-“评分”的对照表，直接根据名字查找评分，无论这个表有多大，查找速度都不会变慢。

3.1.1 直接创建

如果用dict实现，只需要一个“名字”-“评分”的对照表，直接根据名字查找评分，无论这个表有多大，查找速度都不会变慢。用Python写一个dict如下

In [5]:
```
{1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90}
```

Out[5]:
```
{1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90}
```

3.1.2 通过映射函数创建

通过zip()函数将多个**列表或元组**对应位置的元素组，合并为元组。

In [6]:
```
ids = [1,2,3,4,5,6,7]
scores = [94, 93, 92, 92, 92, 91, 90]

dict(zip(ids,scores))
```

Out[6]:
```
{1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90}
```

In [27]:
```
names=("Mike","Tom","Mary","Jack")

scores=(88,89,95,95)

dic=dict(zip(names,scores))

dic
```

Out[27]:
```
{'Mike': 88, 'Tom': 89, 'Mary': 95, 'Jack': 95}
```

3.2 字典赋值或更改

（1）方法一：字典名['键名'] = '值名'

- 使用这种方法，如果键名存在，就会修改这个键值名
- 键名不存在的话，就会当作新增加一对键值到字典里面

In [8]:
```
dic={1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90}

dic[1]=100

dic
```

Out[8]:
```
{1: 100, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90}
```

In [9]:
```
dic={1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90}

dic[0]=100

dic
```

Out[9]:
```
{1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90, 0: 100}
```

（2）方法二：字典名.setdefault('新键名','新值名')

- 这种方法只能新增一对键值

In [11]:
```
dic={1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90}

dic.setdefault(8,100)

dic
```

Out[11]: 100

Out[11]: {1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90, 8: 100}

如果插入的键是存在的，并不能改变已存在的键值对：

In [12]:
```
dic={1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90}

dic.setdefault(1,100)

dic
```

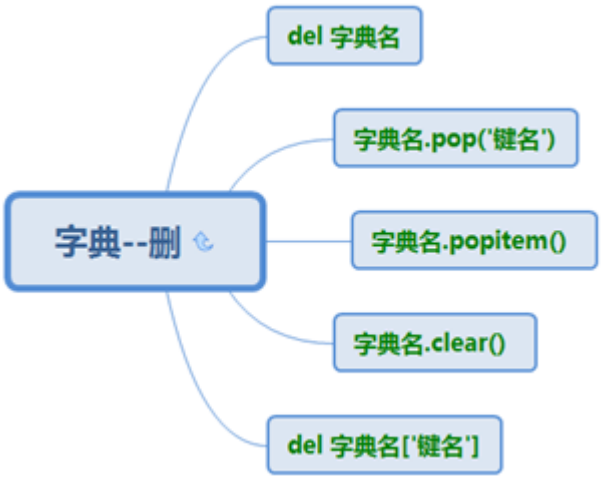
Out[12]: 94

Out[12]: {1: 94, 2: 93, 3: 92, 4: 92, 5: 92, 6: 91, 7: 90}

3.3 删除字典元素

我们就要想一个问题，字典没有索引，那怎么删除字典中的某对键值?

- del 字典名['键名'] ===> 删除指定键值
- 字典名.pop('键名') ===> 删除键值+弹出值
- 字典名.popitem() ===> 随机删除一对(一般最后一对)
- 字典名.clear() ===> 删除词典内所有元素(和列表对应操作相同)
- del 字典名 ===> 删除整个词典(和列表对应操作相同)



（1）del 字典名['键名'] 删除指定键值

In [4]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

del scores['Mike']

scores
```

Out[4]: {'Tom': 89, 'Mary': 95, 'Jack': 95}

（2）字典名.pop('键名') 删除键值+弹出值

pop()内必须要传值，因为字典是无序的

In [7]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

scores.pop('Mike')

scores
```

Out[7]: 88

Out[7]: {'Tom': 89, 'Mary': 95, 'Jack': 95}

pop()内必须要传值，因为字典是无序的，即不能通过索引取键值，也不能通过索引删除键值

（3）字典名.popitem() 随机删除键值、且弹出值

In [8]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

scores.popitem()

scores
```

Out[8]: ('Jack', 95)

Out[8]: {'Mike': 88, 'Tom': 89, 'Mary': 95}

3.4 根据键提取值

（1）方法一：字典名["键名"]

In [13]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

scores["LILI"]

scores
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-13-9b047063b187> in <module>()
      1 scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}
      2
----> 3 scores["LILI"]
      4 scores

KeyError: 'LILI'
```

（2）方法二：字典名.get("键名", default=None)

Python 字典 get() 函数返回指定键的值，如果值不在字典中，则返回默认值。

In [16]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

scores.get("LILI","没有这个人")
```

Out[16]: '没有这个人'

如果指定键的值不存在时，返回该默认值

In [81]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

scores.get("John","查不到人的时候会出现这句话")
```

Out[81]: '查不到人的时候会出现这句话'

3.5 判断键存在否

可以通过 in判断key是否存在

In [21]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

"Mike" in scores
```

Out[21]: True

3.6 练习4

In [34]:
```
d={'键1': '值1',
 '键2': [{'键2.1': ["值2.1"]}],
 '键3': '值3',
 '键4': '值4'}
d
```

Out[34]: {'键1': '值1', '键2': [{'键2.1': ['值2.1']}], '键3': '值3', '键4': '值4'}

怎么取出["值2.1"]?

In [23]:
```

```

Out[23]: ['值2.1']

3.7 字典常用方法

（1）len(字典)

计算字典元素个数

In [12]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

len(scores)
```

Out[12]: 4

（2）字典名.keys()

获取字典key

In [24]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

scores.keys()

[i for i in scores.keys()]
```

Out[24]: dict_keys(['Mike', 'Tom', 'Mary', 'Jack'])

Out[24]: ['Mike', 'Tom', 'Mary', 'Jack']

（3）字典名.values()

获取字典values

In [25]:
```
scores={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

scores.values()

[i for i in scores.values()]
```

Out[25]: dict_values([88, 89, 95, 95])

Out[25]: [88, 89, 95, 95]

（4）字典名.items()

输出一个list格式，非真正意义上的list

In [35]:
```
scores={1:88,2:89,3:95,4:95}

scores.items()

list(scores.items())  # 把字典的key 和 value 转成一个多维list

tuple(scores.items())

[i+j for i,j in scores.items()]
```

Out[35]: dict_items([(1, 88), (2, 89), (3, 95), (4, 95)])

Out[35]: [(1, 88), (2, 89), (3, 95), (4, 95)]

Out[35]: ((1, 88), (2, 89), (3, 95), (4, 95))

Out[35]: [89, 91, 98, 99]

3.8 字典推导式

使用字典推导式可以快速生成一个字典，它的表现形式和列表推导式类似。

In [22]:
```
[i for i in range(10) if i%2==0]
```

Out[22]: [0, 2, 4, 6, 8]

In [40]:
```
{i:i*2 for i in range(10)}
```

Out[40]: {0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}

3.9 练习5

生成一个字典，键从1到10，对应的值从90到99。
提示:

- 用列表生成式+zip()+dict()。
- 用字典推导式

In [37]:

Out[37]: {1: 90, 2: 91, 3: 92, 4: 93, 5: 94, 6: 95, 7: 96, 8: 97, 9: 98, 10: 99}

In [44]:

Out[44]: {1: 90, 2: 91, 3: 92, 4: 93, 5: 94, 6: 95, 7: 96, 8: 97, 9: 98, 10: 99}

4 集合 Set

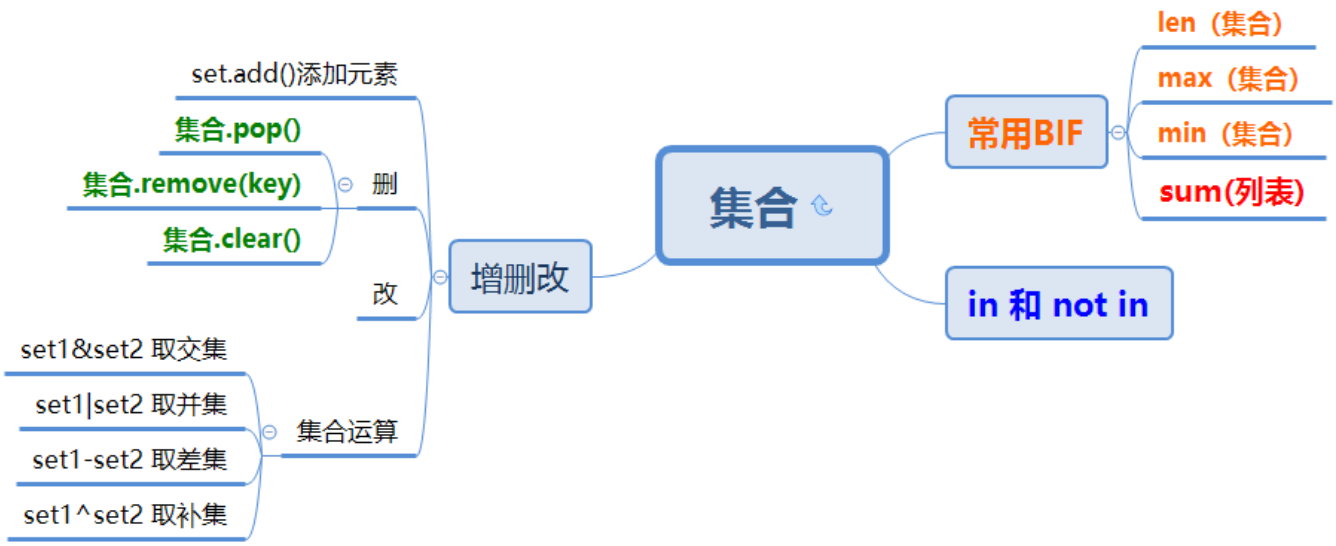
set和dict类似，也是一组key的集合，但不存储value。由于key不能重复，所以，在set中，没有重复的key。

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）



4.1 集合的创建

（1）直接使用“{}”

In [23]:
```
s={1,2,3,4,5,5,5}

s
```

Out[23]: {1, 2, 3, 4, 5}

（2）使用set()

set()函数将列表、元组等其他可迭代对象转换为集合。

In [30]:
```
L=[1,2,3,4]

s=set(L)

s
```

Out[30]: {1, 2, 3, 4}

In [31]:
```
T=(1,2,3,4,"我是元组")

s=set(T)

s
```

Out[31]: {1, 2, 3, 4, '我是元组'}

In [62]:
```
d={"键1":"值1","键2":"值2","键3":"值3"}

s=set(d)

s
```

Out[62]: {'键1', '键2', '键3'}

set 一个常规应用时过滤重复值

In [9]:
```
s = set([1, 1, 2, 2, 3, 3])
s
```

Out[9]: {1, 2, 3}

4.2 集合元素增删

（1）set.add()添加元素

可以通过add(key)方法添加元素到set中，但对于已经存在的值不会有效果。

In [32]:
```
s={1, 2, 3}

s.add(4)

s
```

Out[32]: {1, 2, 3, 4}

（2）从集合删除元素

- 集合.pop()　　删除第一个元素（不能指定弹射某个元素）
- 集合.remove(key)　　删除指定元素
- 集合.clear()　　清空集合

集合.pop()
删除第一个元素

In [24]:
```
s={1, 2, 3, 4}

s.pop()

s
```

Out[24]: 1

Out[24]: {2, 3, 4}

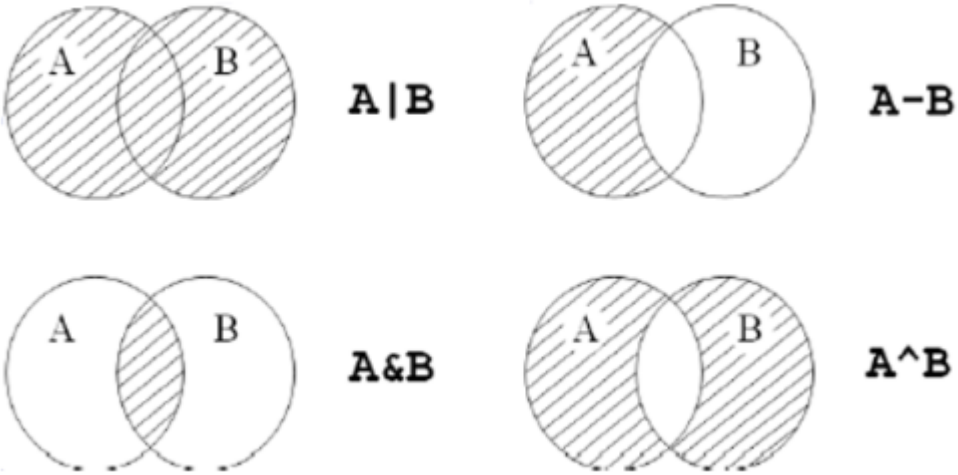
集合.remove(键值)
删除指定元素

In [33]:
```
s={1, 2, 3, 4}

s.remove(3)

s
```

Out[33]: {1, 2, 4}

4.3 集合运算

集合类型的4种基本操作：交集（&）、并集（|）、差集（-）、补集（^），操作逻辑与数学定义相同



（1）set_01&set_02 取交集

两个set可以做数学意义上的交集、并集等操作

In [73]:
```
s_01 = set([1, 2, 3])
s_02 = set([2, 3, 4])

s_01 & s_02
```

Out[73]: {2, 3}

（2）set_01|set_02 取并集

In [75]:
```
s_01 = set([1, 2, 3])

s_02 = set([2, 3, 4])

s_01 | s_02
```

Out[75]: {1, 2, 3, 4}

（3）set_01-set_02 取差集

也就是取set_01中有，set_02中没有的元素

In [76]:
```
s_01 = set([1, 2, 3])

s_02 = set([2, 3, 4])

s_01 - s_02
```

Out[76]: {1}

（4）set_01^set2_0 取补集

也就是取set_01、set_02中交集以外的元素。

In [74]:
```
s_01 = set([1, 2, 3])
s_02 = set([2, 3, 4])

s_01 ^ s_02
```

Out[74]: {1, 4}

4.4 集合包含关系测试

方法	含义
set_01.issubset(set_02)	测试set_01是否为set_02的子集
set_01.isdisjoint(set_02)	测试set_01和set_02是否有交集，有就返回False

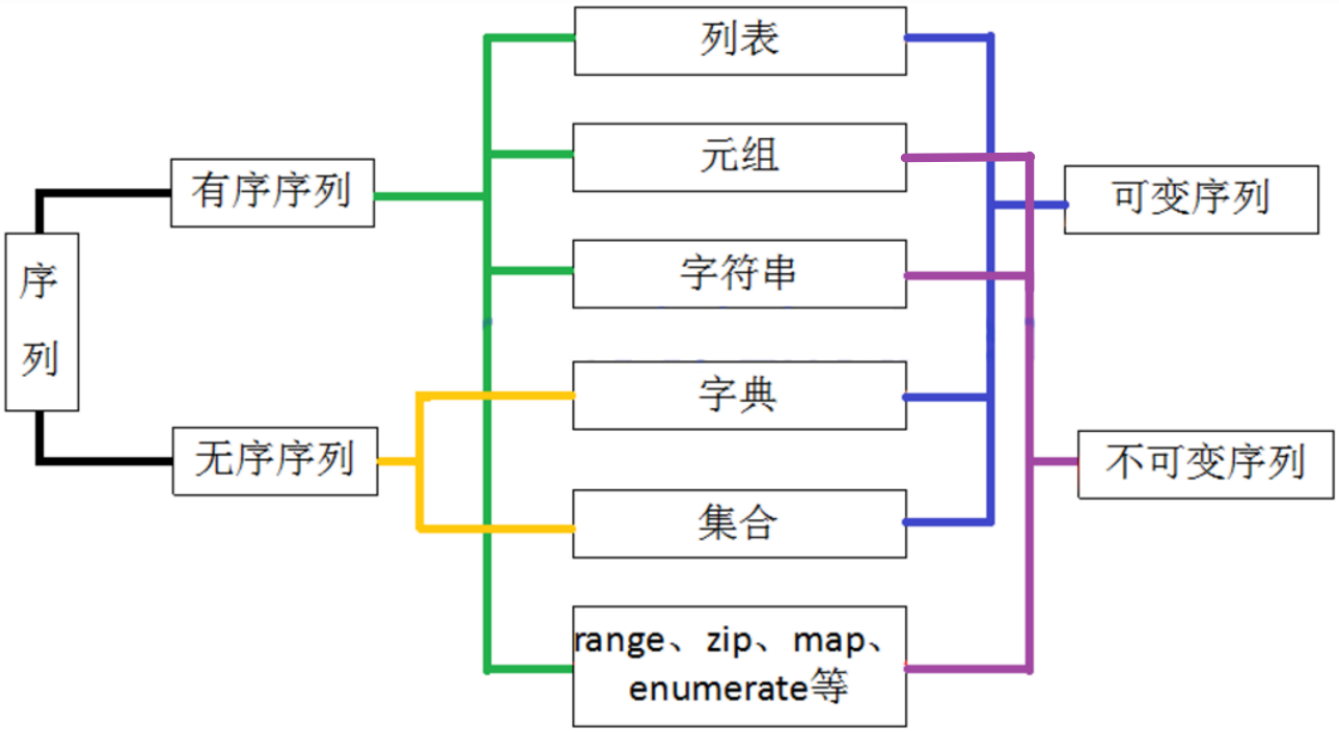
In [77]:
```
{2,3}.issubset({1,2,3,4,5})
```

Out[77]: True

In [78]:
```
{2,3}.isdisjoint({1,2,3,4,5})
```

Out[78]: False

5 序列知识总结



5.1 可变与不可变对象

（1）关于可变对象

首先，我们捋一下可变和不可变指什么。

- 可变对象：
  - 该对象所指向的内存中的值可以被改变。
  - 通俗点说就是原地改变。

列表、字典、集合这三类可变数据类型。

In [5]:
```
a=[1,2,3]

id(a)

a.append(5)

a

id(a)
```

Out[5]: 1799211634312

Out[5]: [1, 2, 3, 5]

Out[5]: 1799211634312

我们再看看字典

In [6]:
```
b={"Mike":88,"Tom":89,"Mary":95,"Jack":95}

id(b)

b["Lily"]=99

b

id(b)
```

Out[6]: 1799221325040

Out[6]: {'Mike': 88, 'Tom': 89, 'Mary': 95, 'Jack': 95, 'Lily': 99}

Out[6]: 1799221325040

（2）关于不可变对象

- 不可变对象：
  - 该对象所指向的内存中的值不能被改变。
  - 当改变某个变量时候，由于其所指的值不能被改变。
  - 相当于把原来的值复制一份后再改变。
  - 这会开辟一个新的地址，变量再指向这个新的地址。

我们举个例子，我们说字符串是不可变对象：

In [12]:
```
a = 'abc'
id(a)

b = a.replace('a','A')

id(b)
```

Out[12]: 1799153055200

Out[12]: 1799221978312

5.2 遍历

In [25]:
```
for i in "abc":
    print(i)
```

```
a
b
c
```

In [27]:
```
for i in ["a","b","c"]:
    print(i)
```

```
a
b
c
```

In [28]:
```
for i in ("a","b","c"):
    print(i)
```

```
a
b
c
```

In [35]:
```
for i in {"a","b","c"}:
    print(i)
```

```
c
a
b
```

In [45]:
```
for i,j in {"a":1,"b":1,"c":1}.items():
    print(i,j)
```

```
a 1
b 1
c 1
```

5.3 zip()函数

In [36]:
```
str_01="abc"
str_02="ABC"

aa=zip(str_01,str_02)

for i in aa:
    print(i)
```

```
('a', 'A')
('b', 'B')
('c', 'C')
```

In [39]:
```
list_01=["a","b","c"]   #这里换成元组也一样
list_02=["A","B","C"]   #这里换成元组也一样

bb=zip(list_01,list_02)

for i in bb:
    print(i)
```

```
('a', 'A')
('b', 'B')
('c', 'C')
```

In [40]:
```
list_01=["a","b","c"]   #这里换成元组也一样
list_02="ABC"  #这里换成元组也一样

bb=zip(list_01,list_02)

for i in bb:
    print(i)
```

```
('a', 'A')
('b', 'B')
('c', 'C')
```

In [22]:
```
list_01={"a":1,"b":2,"c":3}
list_02={"A":1,"B":2,"C":3}

bb=zip(list_01,list_02)

for i in bb:
    print(i)
```

```
('a', 'A')
('b', 'B')
('c', 'C')
```

In [38]:
```
list_01={"a":1,"b":2,"c":3}
list_02={"A":1,"B":2,"C":3}

bb=zip(list_01,list_02)

list(bb)

for i in bb:
    print(i)
```

Out[38]:
```
[('a', 'A'), ('b', 'B'), ('c', 'C')]
```