## Python数据模型

### 1.1 一摞Python风格的纸牌

实现了\_\_getitem\_\_方法，对象就变成可迭代的了。

### 1.2 如何使用特殊方法

#### 1.2.1 模拟数值类型

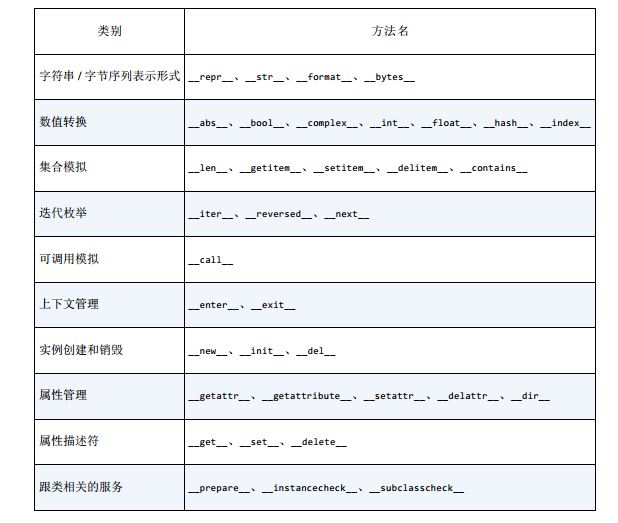
#### 1.2.2 字符串表示

#### 1.2.3 算数运算符

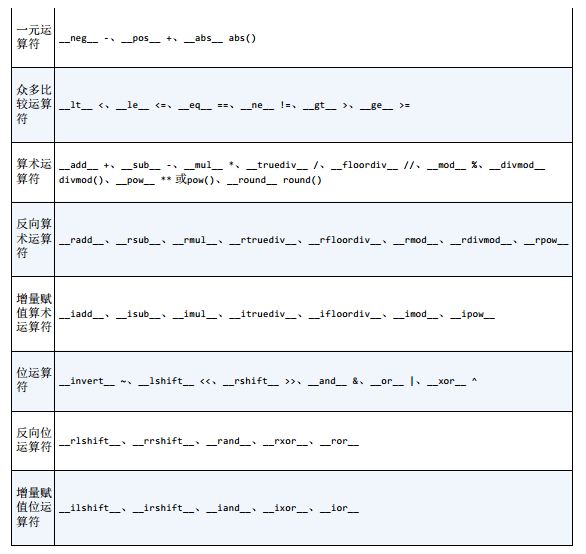
中缀运算符的基本原则就是不改变操作对象，而是产生一个新的值。

#### 1.2.4 自定义的布尔值

### 1.3 特殊方法一览



捕获



### 1.4 为什么len不是普通方法

### 1.5 本章小结

## 序列构成的数组

### 2.1 内置序列类型概览

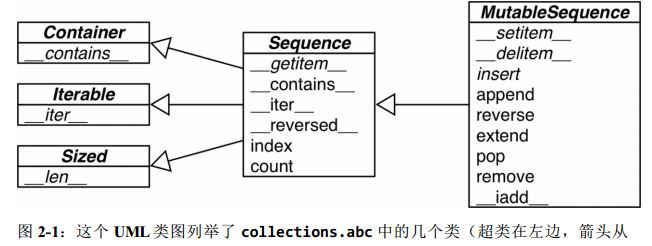
容器序列：list、tuple、collections.deque这些序列能存放不同类型的数据。

扁平序列：str、bytes、bytearray、memoryview和array.array，这类序列只能容纳一种类型。

容器类型存放的是它们所包含的任意类型的对象的引用，而扁平类型里存放的是值而不是引用。

可变序列：list、bytearray、array.array、collections.deque和memoryview。

不可变序列：tuple、str和bytes



### 2.2 列表推导和生成器表达式

#### 2.2.1 列表推到和可读性

>>> x = 'd'

>>> s = [x for x in 'abc']

>>> x

'c'

>>>

python2.x中列表解析可能会影响上下文的同名变量。

#### 2.2.2 列表推导同filter和map比较

#### 2.2.3 笛卡尔积

#### 2.2.4 生成器表达式

>>> c = tuple(i for i in 'abc')

>>> c

('a', 'b', 'c')

### 2.3 元组不仅仅是不可变的列表

#### 2.3.1 元组和记录

元组其实是对数据的记录：元组中每个元素都存放了记录中一个字段的数据，外加这个字段的位置。

#### 2.3.2 元组拆包

#### 2.3.3 嵌套元组拆包

#### 2.3.4 具名元组

collections.namedtuple是一个工厂函数，它可以用来构建一个带字段名的元组和一个有名字的类。

from collections import namedtuple

City = namedtuple(*'City'*, *'name country population coordinates'*)

tokyo = City(*'Tokyo'*, *'JP'*, 36.933, (35.69722, 139.691667))

print tokyo

* 创建一个具名元组需要两个参数，一个是类名，一个是类的各字段的名字。后者可以是由数个字符串组成的可迭代对象，或者是由空格分割开的字段名组成的字符串。
* \_fields属性是一个包含这个类所有字段名称的元组
* 用\_make（）通过接受一个可迭代对象来生成这个类的一个实例
* \_asdict把具名元组以collections.OrderedDict的形式返回。

from collections import namedtuple

City = namedtuple(*'City'*, *'name country population coordinates'*)

tokyo = City(*'Tokyo'*, *'JP'*, 36.933, (35.69722, 139.691667))

delhi\_data = (*'Tokyo'*, *'JP'*, 36.933, (35.69722, 139.691667))

del\_hi = City.\_make(delhi\_data)

print tokyo

print tokyo.\_fields

print City.\_fields

print tokyo.\_asdict()

print del\_hi

输出：

City(name='Tokyo', country='JP', population=36.933, coordinates=(35.69722, 139.691667))

('name', 'country', 'population', 'coordinates')

('name', 'country', 'population', 'coordinates')

OrderedDict([('name', 'Tokyo'), ('country', 'JP'), ('population', 36.933), ('coordinates', (35.69722, 139.691667))])

City(name='Tokyo', country='JP', population=36.933, coordinates=(35.69722, 139.691667))

#### 2.3.5 作为不可变列表的元组







### 2.4 切片

#### 2.4.1 为什么切片和区间会忽略最后一个元素

#### 2.4.2 对对象进行切片

#### 2.4.3 多维切片和省略

#### 2.4.4 给切片赋值

如果赋值的对象是一个切片，那么赋值语句的右侧必须是一个可迭代对象。即便只是单独一个值，也要把它装换成可迭代的序列。

### 2.5 对序列使用+和\*

+和\*都遵循这个规律，不修改原有的操作对象，而是构建一个全新的序列。

my\_list = [[]] \* 3 来初始化一个由列表组成的列表，但是你得到的列表里包含

的 3 个元素其实是 3 个引用，而且这 3 个引用指向的都是同一个列表。

### 2.6 序列的增量赋值

>>> t = (1,2,[30, 40])

>>> t[2] = 10

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

t[2] = 10

TypeError: 'tuple' object does not support item assignment

>>> t[2] += [50, 60]

Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>

t[2] += [50, 60]

TypeError: 'tuple' object does not support item assignment

>>> t

(1, 2, [30, 40, 50, 60])

>>> t[2].extend([70])

>>> t

(1, 2, [30, 40, 50, 60, 70])

>>>

* 不要把可变对象放在元组里面
* 增量赋值不是一个原子操作。

### 2.7 list.sort方法和内置函数sorted

list.sort会就地排序，函数返回值为None。

sorted会新建一个列表作为返回值。

两个可选参数：

**reverse**

　　如果被设定为 True，被排序的序列里的元素会以降序输出（也就是说把最大值当作

最小值来排序）。这个参数的默认值是 False。

**key**

　　一个只有一个参数的函数，这个函数会被用在序列里的每一个元素上，所产生的结果

将是排序算法依赖的对比关键字。比如说，在对一些字符串排序时，可以用

key=str.lower 来实现忽略大小写的排序，或者是用 key=len 进行基于字符串长度的排

序。这个参数的默认值是恒等函数（identity function），也就是默认用元素自己的值来排

序。

### 2.8 用bisect来管理已排序的序列

#### 2.8.1 用bisect来搜索

bisect(haystack, needle) 在 haystack（干草垛）里搜索 needle（针）的位置，该位置满足的条件是，把 needle 插入这个位置之后， haystack 还能保持升序。也就是在说这个函数返回的位置前面的值，都小于或等于 needle 的值。其中 haystack 必须是一个有序的序列。你可以先用 bisect(haystack, needle) 查找位置 index，再用haystack.insert(index, needle) 来插入新值。但你也可用 insort 来一步到位，并且后者的速度更快一些。

其次， bisect 函数其实是 bisect\_right 函数的别名，后者还有个姊妹函数叫bisect\_left。它们的区别在于， bisect\_left 返回的插入位置是原序列中跟被插入元素相等的元素的位置，也就是新元素会被放置于它相等的元素的前面，而 bisect\_right返回的则是跟它相等的元素之后的位置。

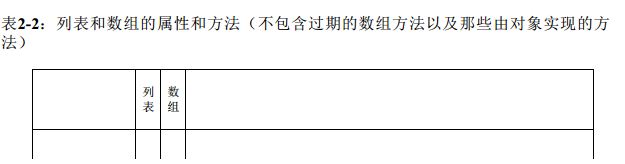
#### 2.8.2 用bisect.insort插入新元素

insort(seq, item) 把变量 item 插入到序列 seq 中，并能保持 seq 的升序顺序。

insort 跟 bisect 一样，有 lo 和 hi 两个可选参数用来控制查找的范围。它也有个变体叫 insort\_left，这个变体在背后用的是 bisect\_left。

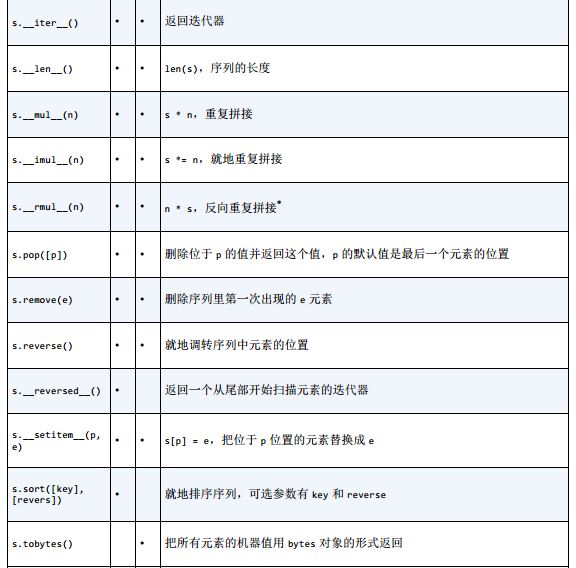
### 2.9 当列表不是首选时

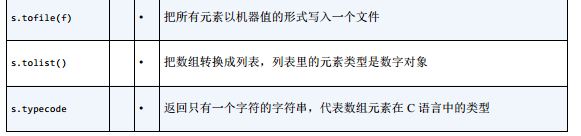
#### 2.9.1 数组







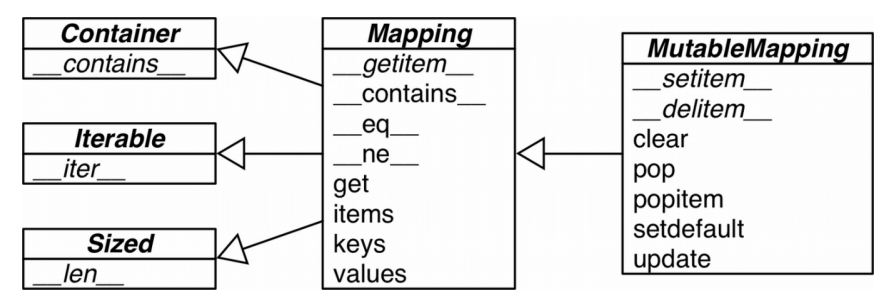




#### 2.9.2 内存视图

## 字典和集合

### 3.1 泛映射类型



什么是可散列数据类型？

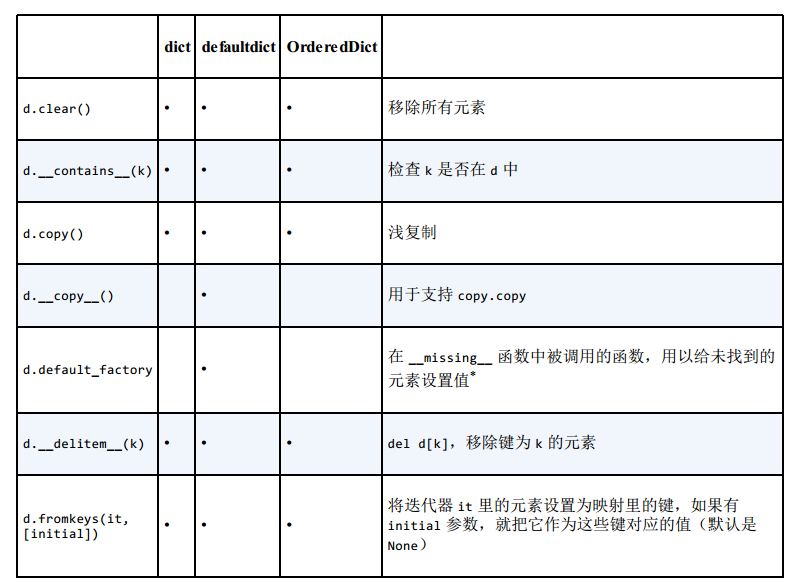
如果一个对象是可散列的，那么这个对象的生命周期中，它的散列值是不变的，而且这个对象需要实现\_\_hash\_\_（）方法。另外可散列对象还要有\_\_qe\_\_（）方法，这样才能跟其他键作比较。

### 3.2 字典推导

>>> {i:j for i in a for j in b}

{'a': 2, 'b': 2}

### 3.3 常见的映射方法









### 3.4 映射的弹性检查

### 3.5 字典的变种