第二章 列表和元组

2.1 序列概述

2.2 序列的主要操作

2.3 列表

2.4 元组

2.5 集合

序列概述

- Python中共有三种基本的序列类型（注意：序列不是Python内部类型的名字）：列表（list）、元组（tuple）和range
- 序列中的每个元素被分配一个序列号，即元素的位置，也称为**索引**，第一个索引是0，第二个是1，以此类推
- 有的序列是可变类型，其对象在建立后可以通过某些操作改变其内部状态；Python中列表是可变序列类型，元组和range都是不可变序列类型

序列概述

- 当需要操作一组数值的时候，可以考虑使用序列。例如，可以使用序列表示数据库中某个人的信息

```
>>> Edward = ['Edward Gumby', 42, '6577923O']
```

- 序列中可以嵌套其他序列，例如还可以这样构建人员信息的列表

```
>>> Edward = ['Edward Gumby', 42, '6577923O']
>>> John = ['John Smith', 39, '65783298']
>>> database = [Edward, John]
>>> database
[['Edward Gumby', 42, '6577923O'], ['John Smith', 39, '65783298']]
```

第二章 列表和元组

2.1 序列概述

2.2 序列的主要操作

2.3 列表

2.4 元组

2.5 集合

索引

- 序列中所有元素都是有编号的，从0开始递增，因此可以通过编号来访问这些元素，例如

```
>>> greeting = 'Good Morning!'
>>> greeting[5]
'M'
```

- 使用负数进行索引时，Python会从序列右侧第1个元素开始计数，该元素的位置编号是-1，例如

```
>>> greeting[-3]
'n'
```

索引

- 序列字面量可以直接使用索引，并不一定需要一个变量引用它们，二者做法的效果相同

```
>>> greeting = 'Good Morning!'
>>> greeting[5]
'M'
>>> 'Good Morning!'[5]
'M'
>>> numbers = [1, 2, 3, 4, 5, 6]
>>> numbers[3]
4
>>> [1, 2, 3, 4, 5][3]
4
>>>
```

```
#根据给定的年月日以数字形式打印出日期
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

#以1-31的数字形式作为结尾的列表
endings = ['st', 'nd', 'rd']  + 17 * ['th']  \
        + ['st', 'nd', 'rd']  + 7 * ['th']  \
        + ['st']

year = input('Year = ')
month = input('Month (1-12) = ')
day = input('Day (1-31) = ')

month_number = int(month)
day_number = int(day)

#将月份和天数减1, 从而获得正确的索引
month_name = months[month_number - 1]
day_name = day + endings[day_number - 1]

print(month_name + ' ' + day_name + ' ' + year)
```

分片

- 使用索引可以访问序列中的某个元素，而使用分片则可以访问序列中一定范围内的元素

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7 , 8]
>>> numbers[3:5]
[4, 5]
```

- 分片操作的实现需要提供两个索引作为边界，第1个索引的元素是包含在分片内的，而第2个则不包含在分片内

如果我们想获得从第 4 个至序列结束的所有元素呢?

分片

```
>>> numbers[3:8]
[4, 5, 6, 7, 8]
>>> numbers[3:16]
[4, 5, 6, 7, 8]
>>> numbers[3:]
[4, 5, 6, 7, 8]
>>> numbers[:4]
[1, 2, 3, 4]
```

分片

- 假设需要访问一个序列中最后若干个元素，可以采用如下方法

```
>>> numbers = [3, 4, 12, 34, 5, 18, 22, 54, 29, 76, 13, 41, 88]
>>> numbers[9:13]
[76, 13, 41, 88]
>>> numbers[-4:]
[76, 13, 41, 88]
```

- 如果需要复制整个序列，则可以使用

```
>>> numbers = [3, 4, 12, 34, 5, 18, 22, 54, 29, 76, 13, 41, 88]
>>> numbers[:]
[3, 4, 12, 34, 5, 18, 22, 54, 29, 76, 13, 41, 88]
```

分片中的步长

- 在普通的分片中，步长隐式设置为1；如果步长设置为大于1的数，则进行分片时就会跳过某些元素

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> numbers[0:6:1]
[1, 2, 3, 4, 5, 6]
>>> numbers[0:6:3]
[1, 4]
>>> numbers[::3]
[1, 4, 7]
>>> numbers[::0]
Traceback (most recent call last):
  File "<pyshell#109>", line 1, in <module>
    numbers[::0]
ValueError: slice step cannot be zero
```

分片中的步长

- 需要从右至左提取序列中的元素时，可以使用负数作为步长

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> numbers[5:2:-2]
[6, 4]
>>> numbers[-4:2:-2]
[6, 4]
>>> numbers[-1::-2]
[9, 7, 5, 3, 1]
>>> numbers[::-2]
[9, 7, 5, 3, 1]
>>> numbers[0::-2]
[1]
```

序列相加

- 通过使用加运算符可以进行序列的连接操作

```
>>> [1, 2, 3] + [5, 6, 7]
[1, 2, 3, 5, 6, 7]
>>> 'Good' + ' Morning!'
'Good Morning!'
>>> [1, 2, 3] + 'Good!'
Traceback (most recent call last):
  File "<pyshell#120>", line 1, in <module>
    [1, 2, 3] + 'Good!'
TypeError: can only concatenate list (not "str") to list
```

- 因此复制整个序列还可以使用如下方法

```
>>> numbers = [3, 4, 12, 34, 5, 18, 22, 54, 29, 76, 13, 41, 88]
>>> numbers[:4] + numbers[4:]
[3, 4, 12, 34, 5, 18, 22, 54, 29, 76, 13, 41, 88]
```

序列重复

- 使用某个数字n乘以一个序列会生成新的序列，在该序列中原来的序列被重复了n遍

```
>>> 'Hooray!  ' * 3
'Hooray!  Hooray!  Hooray!  '
>>> [1, 3, 5] * 4
[1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5]
```

- 如果需要初始化一个长度为10的列表，可以按照以下方法

```
>>> sequence = [] * 10
>>> print(sequence)
[]
>>> sequence = [None] * 10 #None是Python的内建值,表示什么也没有
>>> print(sequence)
[None, None, None, None, None, None, None, None, None, None]
```

```
#以正确的宽度在居中的"盒子"中打印一个句子用户输入的句子，例如
#          +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
#          |                                                               |
#          |                         Good Morning!                         |
#          |                                                               |
#          +- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +

# 当你的Python IDLE设置为不同字体时，显示结果会不同，此处IDLE中字体设置为Times New Roman TUR

sentence = input('Sentence = ')

screen_width = 80
text_width = len(sentence)
box_width = text_width + 8
left_margin = (screen_width - box_width) //2

print(' ' * left_margin + '+' + '-' * box_width + '+') # ' '中为一个空格
print(' ' * (left_margin + 3) + '|' + ' ' * (text_width) + ' |')  # ' |'中有一个空格
print(' ' * (left_margin + 3)  + '|' + sentence  + ' |')
print(' ' * (left_margin + 3)  + '|' + ' ' * (text_width) + ' |')
print(' ' * left_margin + '+' + '-'  * box_width + '+')
```

成员资格判断

- 为了检查一个值是否在序列中，可以使用in运算符，该运算符检查某个条件是否为真，然后返回相应的值

```
>>> permissions = 'rwx'
>>> 'w' in permissions
True
>>> 'x' not in permissions
False
>>> users = ['Alan', 'Barry', 'Catherine']
>>> input('Enter your user name:')
Enter your user name:Tony
'Tony'
```

```
>>> database = [['albert', '1234'], ['dilbert', '4323'], ['smith', '7528'], ['jones', '9802']]
>>> username = input('Enter your user name :')
Enter your user name :albert
>>> pin = input('Enter your PIN code :')
Enter your PIN code :1234
>>> if [username, pin] in database: print('Access granted')
```

长度、最大值、最小值和求和

- 内建函数len、min、max和sum都非常有用，其中len函数返回序列中所包含元素的个数，min函数和max函数分别返回序列中最大和最小的元素，sum函数返回序列中所有元素的总和（如果可能）

```
>>> numbers = [100, 24, 87]
>>> len(numbers)
3
>>> max(numbers)
100
>>> min(numbers)
24
>>> sum(numbers)
211
>>> sum([1, 2, 3, 'One', 'Two', 'Three'])
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    sum([1, 2, 3, 'One', 'Two', 'Three'])
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

第二章 列表和元组

2.1 序列概述

2.2 序列的主要操作

2.3 列表

2.4 元组

2.5 集合

创建和删除列表

- 使用list函数可以创建列表

```
>>> list('Hello')
['H', 'e', 'l', 'l', 'o']
```

- 删除列表则需要使用del语句

```
>>> print(a)
['H', 'e', 'l', 'l', 'o']
>>> a
['H', 'e', 'l', 'l', 'o']
>>> del a
>>> print(a)
Traceback (most recent call last):
  File "<pyshell#180>", line 1, in <module>
    print(a)
NameError: name 'a' is not defined
```

列表的基本操作：append

- **方法**是一个与某些对象有紧密联系的函数,这些对象可能是列表、数字,也可能是字符串或者其他类型的对象，方法通常这样调用

  对象.方法（参数）

- append方法：在列表末尾追加新的对象，它**不是简单地返回一个修改过的新列表**，而是直接修改原来的列表

```
>>> lst = ['Apple', 'Banana', 'Citrus']
>>> lst.append('dragonfruit')
>>> lst
['Apple', 'Banana', 'Citrus', 'dragonfruit']
>>> lst.append(['fig', 'grape'])
>>> lst
['Apple', 'Banana', 'Citrus', 'dragonfruit', ['fig', 'grape']]
```

列表的基本操作：count

- count方法可以统计某个元素在列表中出现的次数

```
>>> x = [[1, 2], 1, 1, [2, 1, [1, 2]]]
>>> x.count(1)      # 只需统计元素1出现的次数，如果1作为某列表成员，则不计
2
>>> x.count([1, 2])
1
>>>
>>> x = [[1, 2], 1, 1, [2, 1, [1, 2]]]
>>> x.count(1)      # 只需统计元素1出现的次数，如果1作为某列表成员，则不计
2
>>> x.count([1, 2]) # 只需统计[1,2]出现的次数，如果[1,2]作为某列表成员，则不计
1
>>> ['to', 'be', 'or', 'not', 'to', 'be'].count('be')
2
```

列表的基本操作：extend

- extend方法可以一次性在列表的末尾追加另外一个序列中的多个值，也就是说可以扩展原有的列表

```
>>> a = [1, 3, 5]
>>> b = [7, 9, 11, 13]
>>> a.extend(b)
>>> a
[1, 3, 5, 7, 9, 11, 13]
>>> # 通过extend方法，原列表a得到了扩展
>>> a = [1, 3, 5]
>>> b = [7, 9, 11, 13]
>>> a.append(b)  # 此处将列表b作为一个元素加入到列表a中
>>> a
[1, 3, 5, [7, 9, 11, 13]]
```

列表的基本操作：extend

- extend方法看起来很像连接操作，但是二者之间的主要区别在于：**extend方法修改了被扩展的原始序列，而连接操作会返回一个全新的列表**

```
>>> a = [1, 3, 5]
>>> b = [7, 9, 11, 13]
>>> a + b
[1, 3, 5, 7, 9, 11, 13]
>>> a
[1, 3, 5]
>>> a[len(a):] = b
>>> # 以上利用分片赋值来扩展原始列表a
>>> a
[1, 3, 5, 7, 9, 11, 13]
```

列表的基本操作：index

- index方法用于从列表中找出某个值第一个匹配项的索引位置

```
>>> knights = ['we', 'are', 'the', 'knights', 'who', 'say', 'ni']
>>> knights.index('say')
5
>>> knights.index('thy')
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    knights.index('thy')
ValueError: 'thy' is not in list
```

列表的基本操作：insert

- insert方法用于将对象插入到列表中的指定位置

```
>>> numbers = [1, 3, 11, 13, 15]
>>> numbers.insert(2, 5)
>>> numbers
[1, 3, 5, 11, 13, 15]
>>> numbers[3:3] = [7, 9]  # 使用分片赋值的方法来达到与insert方法相同的效果
>>> numbers
[1, 3, 5, 7, 9, 11, 13, 15]
```

列表的基本操作：pop

- pop方法会移除列表中的一个元素（默认是最后一个），并且返回该元素的值
- **pop是唯一一个既能修改列表又返回元素值（除None以外）的列表方法**

```
>>> lst = [1, 3, 5, 7]
>>> lst.pop()
7
>>> lst.pop(2)  # 也可以指明待删除元素的位置
5
>>> lst
[1, 3]
```

列表的基本操作：pop

- pop方法可以用来实现一种常见的数据结构—栈，即后进先出的线性表
- 栈的两种基本操作入栈和出栈在Python中可以实现，尽管没有直接的入栈方法，但可以使用append方法来替代，出栈则直接使用pop方法

```
>>> x = [1, 3, 5]
>>> x.append(7)
>>> x
[1, 3, 5, 7]
>>> x.pop()
7
>>> x
[1, 3, 5]
```

列表的基本操作：remove

- remove方法用于移除列表中的**某个元素的第一个匹配项**
- remove方法修改了列表但是没有返回值，这一点和pop方法相反

```
>>> x = ['to', 'be', 'or', 'not', 'to', 'be']
>>> x.remove('be')
>>> x
['to', 'or', 'not', 'to', 'be']
>>> x.pop(4)
'be'
>>> x
['to', 'or', 'not', 'to']
```

列表的基本操作：reverse

- reverse方法将列表中的元素反向存放，该方法也改变了列表但是不返回值（与remove一样）

```
>>> x = [1, 3, 5, 7]
>>> x.reverse()
>>> x
[7, 5, 3, 1]
>>> list(reversed(x)) #使用reversed()对序列进行反向迭代
[1, 3, 5, 7]
>>> x
[7, 5, 3, 1]
```

列表的基本操作：sort

- sort方法用于在原位置对列表进行排序，“原位置排序”意味着改变原来的列表。从而让其中的元素按照一定的顺序排列

```
>>> x = [5, 7, 3, 1, 9]
>>> x.sort()
>>> x
[1, 3, 5, 7, 9]
```

列表的基本操作：sort

- 如果在排序时希望保留原有列表不变，采用以下方法是不行的

```
>>> x = [5, 7, 3, 1, 9]
>>> y = x.sort()
>>> y
>>> # sort()方法返回的是空值
```

- 应该先将待排序列表的副本先保留

```
>>> x = [5, 7, 2, 1, 9]
>>> y = x[:]
>>> y.sort()
>>> print(y)
[1, 2, 5, 7, 9]
>>> print(x)
[5, 7, 2, 1, 9]
```

遍历列表中的元素

- 列表中的元素是可迭代的（iterable），Python支持使用for循环来顺序遍历列表中的所有元素

```
>>> lst = [1, 3, 5, 7, 9]
>>> for i in lst:  # 对于列表lst中的每个元素
        print(i)   # 打印该元素


1
3
5
7
9
```

随机排序

- 可以使用random模块中的shuffle函数将列表中的所有元素进行随机排序

```
>>> from random import shuffle
>>> numbers = [100, 23, 12, 67, 39]
>>> shuffle(numbers)
>>> print(numbers)
[23, 39, 67, 12, 100]
```

第二章 列表和元组

2.1 序列概述

2.2 序列的主要操作

2.3 列表

2.4 元组

2.5 集合

创建和删除元组

- 元组与列表一样,也是一种序列,不同的是**元组不能修改!**
  当不允许对你的应用中某个列表进行修改时,可以考虑使用元组

- 创建元组的语法很简单:使用逗号分隔一些值,就可以自动创建了元组

```
>>> 1, 3, 5
(1, 3, 5)
>>> ()  # 空的元组
()
>>> (1, 3, 5)
(1, 3, 5)
>>> tuple([1, 3, 5])  #从列表中创建元组
(1, 3, 5)
>>> tuple('Hello')  #从字符串中创建元组
('H', 'e', 'l', 'l', 'o')
```

创建和删除元组

- **逗号对于元组的表示非常重要**,仅靠添加圆括号是不够的,例如(24)和24是完全一样的,但是添加了逗号就能彻底改变表达式的值

```
>>> 3 * (20 + 4) # 此处圆括号中的表达式可以求值
72
>>> 3 * (20 + 4,)# 此处圆括号中的是只有一个值的元组
(24, 24, 24)
```

- 删除元组的操作和删除列表一样,使用del语句

```
>>> x = (1, 3, 5)
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#107>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

元组的基本操作

- 元组其实并不复杂,除了创建和访问元组以外,没有太多其他操作;其中访问元组可以使用分片操作

```
>>> x = 1, 3, 5
>>> x[2] = 4
Traceback (most recent call last):
  File "<pyshell#137>", line 1, in <module>
    x[2] = 4
TypeError: 'tuple' object does not support item assignment
>>> x[1:]
(3, 5)
>>> x.index(3)
1
>>> 3 in x
True
>>> x.count(2)
0
>>> len(x)
3
>>> max(x)
5
>>>
```

第二章 列表和元组

2.1 序列概述

2.2 序列的主要操作

2.3 列表

2.4 元组

2.5 集合

集合的创建

- 与列表一样,集合可以用来存放大量的元素;与列表不同的是,集合中不允许有重复的元素,且元素之间没有特别的存放次序
- 创建集合需要将所有元素放在一对大括号({})中,然后用逗号将它们隔开

```
>>> set()  # 创建空集
set()
>>> {1, 3, 5}
{1, 3, 5}
>>> set([1, 3, 5])  # 从列表中创建集合
{1, 3, 5}
>>> set((1, 3, 5))  # 从元组中创建集合
{1, 3, 5}
>>> set('Hello')   # 从字符创中创建集合
{'o', 'l', 'H', 'e'}
>>> # 上面这个例子可以看出集合中的元组是没有特定的存放次序的
>>>
>>> set([1, 2, 3, 'One', 'Two', 'Three'])
{1, 2, 3, 'Two', 'Three', 'One'}
>>> # 一个集合中的元素可以不是相同的数据类型
```

存取访问集合

- 可以通过add()和remove()方法向一个集合中增加新元素和删除已有元素；还可以使用len()、max()、min()和sum()函数来求取一个集合的长度、所有元素的最大值和最小值、以及集合中所有元素的总和（如果可能）

```
>>> s1 = {1, 3, 5}
>>> s1.add(2)
>>> s1
{1, 2, 3, 5}
>>> s1.remove(3)
>>> s1
{1, 2, 5}
>>> len(s1)
3
>>> max(s1)
5
>>> min(s1)
1
>>> sum(s1)
8
>>> sum({1, 2, 3, 'One', 'Two', 'Three'})
Traceback (most recent call last):
  File "<pyshell#170>", line 1, in <module>
    sum({1, 2, 3, 'One', 'Two', 'Three'})
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

子集和超集

- 如果一个集合S1中每个元素同时也是另外一个集合S2中的元素，那么S1是S2的子集（subset），S2是S1的超集（superset）

```
>>> s1 = {1, 3, 5}
>>> s2 = {1, 2, 3, 4, 5, 6}
>>> s1.issubset(s2)   # 判断s1是否是s2的子集
True
>>> s2.issuperset(s1)   # 判断s2是否是s1的超集
True
>>>
```

集合的相等性测试

- 可以使用==和!=操作符来判断两个集合是否包含相同的元素，此时元素在各个集合中的存放次序是无需考虑的

```
>>> s1 = {1, 2, 3, 4, 5, 6}
>>> s2 = {1, 3, 5, 2, 4, 6}
>>> s1 == s2
True
>>> s1 != s2
False
```

- 当用于集合时，普通的比较操作符有特殊的含义
  - 如果s1是s2的真子集，s1 < s2 返回 True
  - 如果s1是s2的子集，s1 <= s2返回True
  - 如果s1是s2的真超集，s1> s2返回True
  - 如果s1是s2的超集，s1 >= s2返回True

集合的基本操作

- Python提供了用来进行集合的并（union）、交（intersection）、差（difference）和对称差（symmetric difference：数学上，两个集合的对称差是只属于其中一个集合，而不属于另一个集合的元素组成的集合）等操作

```
>>> s1 = {1, 2, 3, 5}
>>> s2 = {1, 2, 4, 6}
>>> s1.union(s2)     # 求s1和s2的并集
{1, 2, 3, 4, 5, 6}
>>> s1
{1, 2, 3, 5}
>>> s1 | s2          # 求s1和s2的并集
{1, 2, 3, 4, 5, 6}
>>> s1.intersection(s2)    # 求 s1和s2的交集
{1, 2}
>>> s1 & s2          # 求s1和s2的交集
{1, 2}
```

集合的基本操作

```
>>> s1 = {1, 2, 3, 5}
>>> s2 = {1, 2, 4, 6}
>>> s1.difference(s2)  # 求s1和s2的差集
{3, 5}
>>> s1 - s2       # 求s1和s2的差集
{3, 5}
>>> s1.symmetric_difference(s2)  # s1和s2的对称差
{3, 4, 5, 6}
>>> s1 ^ s2     # 求s1和s2的对称差
{3, 4, 5, 6}
```

集合与列表的性能比较

- 就in和not in操作符、以及remove方法而言，集合的效率比列表的效率更高

- 访问列表中的元素可以使用索引操作符，但是如果要访问集合中的所有元素，则需要使用循环语句，例如for循环，因为集合中的元素是无序的

```
import time

NUMBER_OF_ELEMENTS = 10000

s1 = set(list(range(NUMBER_OF_ELEMENTS))) # 创建一个包含1000个元素的集合
s2 = list(range(NUMBER_OF_ELEMENTS))      # 创建一个包含1000个元素的列表

# 测试某个元素是否在集合s1中
startTime = time.time()   # 获取测试开始时间
for i in range(NUMBER_OF_ELEMENTS):
    i in s1
endTime = time.time()   # 获取测试结束时间
runningTime = int((endTime - startTime) * 1000)  # 得到运行时间
print('To test if', NUMBER_OF_ELEMENTS, 'elements are in the set, the\
     running time is', runningTime, 'milliseconds')


# 测试某个元素是否在列表s2中
startTime = time.time()   # 获取测试开始时间
for i in range(NUMBER_OF_ELEMENTS):
    i in s2
endTime = time.time()   # 获取测试结束时间
runningTime = int((endTime - startTime) * 1000)  # 得到运行时间
print('To test if', NUMBER_OF_ELEMENTS, 'elements are in the list, the\
    running time is', runningTime, 'milliseconds')
```

小结

- 可以使用len、max、min和sum函数来获取列表的长度、以及所有元素的最大值、最小值和总和
- random模块中的shuffle函数可以将列表中的元素进行随机排序
- 元组是固定的列表，不允许在元组中增加、删除和修改元素
- 由于元组是序列的一种，因此所有用于序列的操作也可以用于元组
- 集合可以像列表一样用于存放大量的元素，与列表不同的是，集合中不允许有重复元素，且所有元素的位置是无序的

小结

- 使用add方法可以向集合中增加元素，而remove方法则可以从集合中删除指定元素
- len、max、min和sum函数均可以用于集合
- 需要遍历列表或集合中的所有元素时均可以考虑使用for循环
- 需要判断子集和超集的关系时，可以使用issubset和issuperset方法，此外还可以使用|、&、-和^来求解两个集合的并、交、差和对称差
- 涉及到测试集合或列表中是否包含特定元素，或者从集合或列表中移出指定元素时，集合的效率比列表的效率更高

本章节涉及到的函数

len(seq)	返回序列长度
list(seq)	从序列中构建列表
max(args)	返回序列或参数集中的最大值
min(args)	返回序列或参数集中的最小值
reversed(seq)	对序列进行反向迭代
sorted(seq)	返回已经排序的包含seq所有元素的列表
tuple(seq)	从序列中构建元组
set(seq)	从序列中构建集合