Chap3 Sequence

第3章 序列

Department of Computer Science and Technology
Department of University Basic Computer Teaching
Nanjing University

3.1 序列

序列

- aStr = 'Hello, World!'
- aList = [2, 3, 5, 7, 11]
- aTuple = ('Sunday', 'happy')
- x = range(10)
- pList = [('AXP', 'American Express Company', '78.51'),
  ('BA', 'The Boeing Company', '184.76'),
  ('CAT', 'Caterpillar Inc.', '96.39'),
  ('CSCO', 'Cisco Systems, Inc.', '33.71'),
  ('CVX', 'Chevron Corporation', '106.09')]

序列是一种最基本最重要的数据结构

序列类型

序列类型是一种容器
通过索引访问成员

01 字符串

单引号、双引号、三引号内的都是字符串，不可变类型

02 列表

强大的类型，用方括号 [] 界别，可变类型

03 元组

与列表相似，用小括号 () 界别，不可变类型

04 range对象

用range()函数生成一个不可变的数字序列，不可变类型

字符串
Strings

列表
Lists

元组
Tuples

range对象
range objects

3.1.1 索引

序列的索引

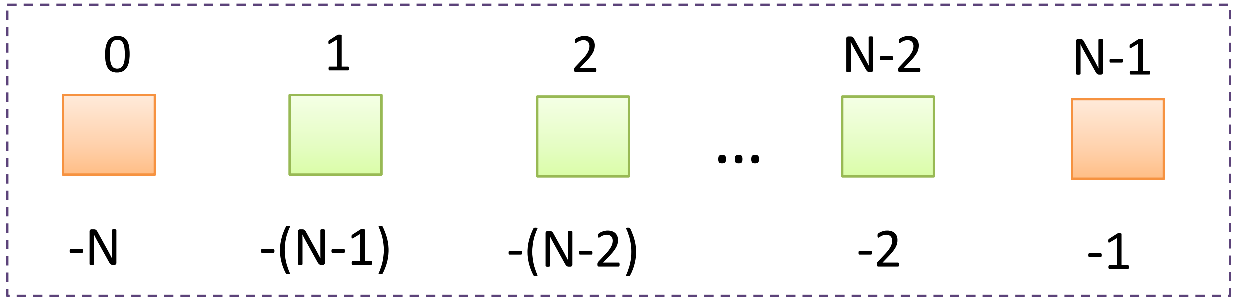
- 序列类型对象一般有多个成员组成，每个成员通常称为元素，每个元素都可以通过**索引（index）**进行访问，索引用方括号“[]”表示。如：

```
sequence[index]
```

序列的索引

week	0	1	2	3	4	5	6
	'Monday'	'Tuesday'	'Wednesday'	'Thursday'	'Friday'	'Saturday'	'Sunday'
	-7	-6	-5	-4	-3	-2	-1

序列



访问模式

- 元素从0开始通过下标偏移量访问
- 一次可访问一个或多个元素

索引的使用

Source

```
>>> aList = ['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.', 'Sun.']
>>> aList[1]
'Tues.'
>>> aList[-1]
'Sun.'
>>> aStr = 'apple'
>>> aStr[1]
'p'
```

序列相关操作

标准类型运算

值比较
对象身份比较
布尔运算

通用序列类型操作

切片
重复
连接
判断成员

序列类型函数

序列类型转换内建函数
序列类型其他内建函数

3.1.2 标准类型运算

标准类型运算符

值比较

<	>
<=	>=
==	!=

对象身份比较

is
is not

布尔运算

not
and
or

值比较

```
>>> 'apple' < 'banana'
True
>>> [1,3,5]  != [2,4,6]
True
>>> aList[1] == 'Tues.'
True
>>> [1, 'Monday'] < [1,
'Tuesday']
True
```

```
>>> ['o', 'k'] < ('o', 'k')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    ['o', 'k'] < ('o', 'k')
TypeError: unorderable types: list() < tuple()
>>> [1 , [2 , 3]] < [1 , ['a' , 3]]
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    [1 , [2 , 3]] < [1 , ['a' , 3]]
TypeError: unorderable types: int() < str()
```

对象身份比较

Source

```
>>> aTuple = ('BA', 'The Boeing Company', '184.76')
>>> bTuple = aTuple
>>> bTuple is aTuple
True
>>> cTuple = ('BA', 'The Boeing Company', '184.76')
>>> aTuple is cTuple
False
>>> aTuple == cTuple
True
```

布尔（逻辑）运算

Source

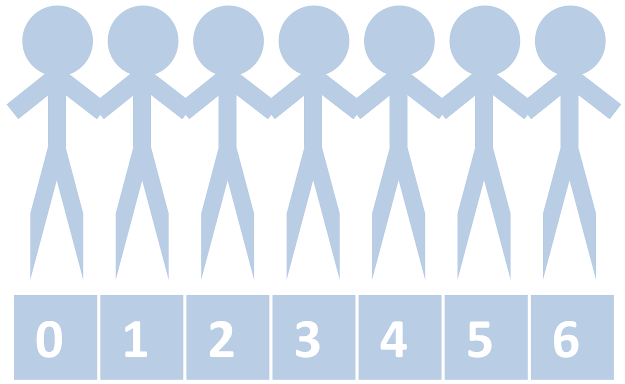
```
>>> ch = 'k'
>>> 'a' <= ch <= 'z' or 'A' <= ch <= 'Z'
True
```

3.1.3 通用序列类型操作

序列类型运算符

s[i]	s[i:j]	s[i:j:k]	s * n, n * s	s + t	x in s	x not in s

切片



索引值

```
>>> aStr = 'American Express Company'
>>> aStr[9: 16]
'Express'
```

切片操作的形式为：

sequence[startindex : endindex]

切片

Source

```
>>> aList = ['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.', 'Sun.']
>>> aList [0: 5]
['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.']
>>> aList[: 5]
['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.']
>>> aList[5: 7]
['Sat.', 'Sun.']
```

切片

Source

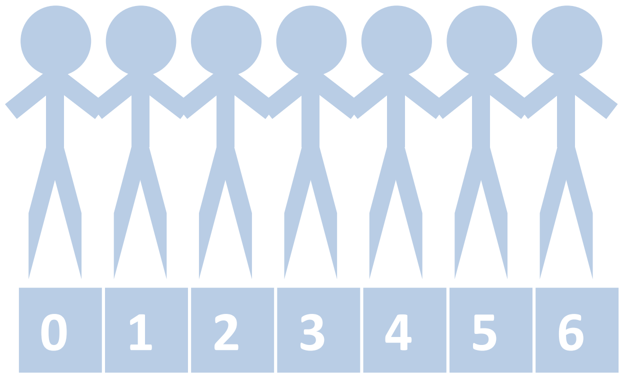
```
>>> aList[-2: -1]
['Sat.']
>>> aList[1:-1]
['Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.']
>>> aList[-2: -3]
[]
>>> aList[-2:]
['Sat.', 'Sun.']
>>> aList[:]
['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.', 'Sun.']
```

切片

切片操作的另一种格式，可以选择切片操作时的步长：

sequence[startindex : endindex : steps]

aList[0: 5] == aList[0: 5: 1]



切片

Source

```
>>> aList = ['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.', 'Sun.']
>>> aList[1: 6: 3]
['Tues.', 'Fri.']
>>> aList[::3]
['Mon.', 'Thur.', 'Sun.']
>>> aList[::-3]
['Sun.', 'Thur.', 'Mon.']
>>> aList[5: 1: -2]
['Sat.', 'Thur.']
```

切片

Source

```
>>> aStr = 'apple'
>>> aStr[0: 3]
'app'
>>> aTuple = (3, 2, 5, 1, 4, 6)
>>> aTuple[1: : 2]
(2, 1, 6)
```

切片

Source

```
>>> aList = ['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.', 'Sun.']
>>> day = aList[int(input('The day of the week(1-7): ')) - 1]
The day of the week(1-7): 5
>>> print( 'Today is ' + day + '.')
Today is Fri..
```

切片

Source

```
>>> week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
>>> print(week[1], week[-2], '\n', week[1:4], '\n', week[:6], '\n', week[::-1])
Tuesday Saturday
['Tuesday', 'Wednesday', 'Thursday']
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
['Sunday', 'Saturday', 'Friday', 'Thursday', 'Wednesday', 'Tuesday', 'Monday']
```

重复

Source

```
>>> 'apple' * 3
'appleappleapple'
>>> (1, 2, 3) * 2
(1, 2, 3, 1, 2, 3)
>>> aTuple = (3, 2, 5, 1)
>>> aTuple * 3
(3, 2, 5, 1, 3, 2, 5, 1, 3, 2, 5, 1)
>>> ['P' , 'y', 't', 'h', 'o', 'n'] * 2
['P', 'y', 't', 'h', 'o', 'n', 'P', 'y', 't', 'h', 'o', 'n']
```

重复操作的形式为：

sequence * copies

连接

Source

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
>>> (1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)
>>> 'pine' + 'apple'
'pineapple'
>>> ['t', 'h', 'e'] + 'apple'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    ['t', 'h', 'e'] + 'apple'
TypeError: can only concatenate list (not "str") to list
```

连接操作的形式为：

sequence1 + sequence2

判断成员

Source

```
>>> aList = ['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.', 'Sun.']
>>> 'Mon.' in aList
True
>>> 'week' in aList
False
>>> 'week' not in aList
True
```

判断一个元素是否属于一个序列操作的形式为：

```
obj in sequence
obj not in sequence
```

判断成员

Source

```
>>> username = ['Jack', 'Tom', 'Halen', 'Rain']
>>> input("please input your name: ") in username
please input your name: Halen
True
```

3.1.4 序列类型函数

序列类型转换内建函数

list()
tuple()
str()

Source

```
>>> list('Hello, World!')
['H', 'e', 'l', 'l', 'o', ',',  ' ', 'W', 'o', 'r', 'l', 'd', '!']
>>> tuple("Hello, World!")
('H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!')
>>> list((1, 2, 3))
[1, 2, 3]
>>> tuple([1, 2, 3])
(1, 2, 3)
```

序列类型转换内建函数

list()
tuple()
str()

Source

```
>>> str(123)
'123'
>>> str(('t','h','e'))
"('t', 'h', 'e')"
```

序列类型其他常用内建函数

len()	enumerate()
sorted()	max()
sum()	min()
zip()	reversed()

Source

```
>>> aStr = 'Hello, World!'
>>> len(aStr)
13
>>> sorted(aStr)
[' ', '!', ',', 'H', 'W', 'd', 'e', 'l', 'l', 'l', 'o', 'o', 'r']
```

序列类型其他常用内建函数

len()

Source

```
>>> aStr = 'Hello, World!'
>>> len(aStr)
13
```

sorted()

Source

```
>>> nList = [3, 2, 5, 1]
>>> sorted(nList)
[1, 2, 3, 5]
>>> nList
[3, 2, 5, 1]
```

序列类型其他常用内建函数

reversed()

Source

```
>>> nList = [3, 2, 5, 1]
>>> reversed(nList)
<list_reverseiterator object at 0x0000018024361B70>
>>> list(reversed(nList))
[1, 5, 2, 3]
```

序列类型其他常用内建函数

sum()

Source

```
>>> sum(['a', 'b', 'c'])
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    sum(['a', 'b', 'c'])
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> sum([1, 2, 3.5])
6.5
```

序列类型其他常用内建函数

max() 和 min()

Source

```
>>> aList = ['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.', 'Sun.']
>>> max(aList)
'Wed.'
>>> max([1, 2.5, 3])
3
>>> max([1, 5, 3],[1, 2.5, 3])
[1, 5, 3]
>>> max([1, 5, 3, 1],[1, 9, 3])
[1, 9, 3]
```

序列类型其他常用内建函数

enumerate()

Source

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start = 1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

序列类型其他常用内建函数

zip()

Source

```
>>> list(zip('hello', 'world'))
[('h', 'w'), ('e', 'o'), ('l', 'r'), ('l', 'l'), ('o', 'd')]
```

3.2 字符串

3.2.1 字符串的表示

字符串的表示形式

```
>>> aStr = 'The Boeing Company'
>>> bStr = "The Boeing Company "
>>> cStr = '''The Boeing
company'''
>>> aStr
'The Boeing Company'
>>> bStr
'The Boeing Company'
>>> cStr
'The Boeing\nCompany'
```

字符串的表示形式

Source

```
>>> dStr = "I'm a student."
>>> dStr
"I'm a student."
>>> eStr = '"No pain, No gain." is a good saying.'
>>> eStr
'"No pain, No gains." is a good saying.'
>>> "break" 'fast'    # "break""fast"或'break''fast'等形式亦可
'breakfast'
```

字符串的表示形式

Source

```
>>> cStr = '''The Boeing
company'''
>>> cStr
'The Boeing\nCompany'
>>> fStr = '''It's said that
... where there is a will, there is a way.'''
>>> fStr
"It's said that\nwhere there is a will, there is a way."
```

三引号
分行输入

字符串的表示形式

Source

```
>>> gStr = r'd:\python\n.py'
>>> gStr
'd:\\python\\n.py'
```

原始字符串
操作符

字符串的创建和访问

Source

```
>>> aStr = 'The Boeing Company'
>>> print('football')
football
```

访问方式：

切片

创建方式：

赋值

直接

输出

Source

```
>>> aStr = 'The Boeing Company'
>>> hStr = aStr[:4] + 'IBM' + aStr[-8:]
>>> hStr
'The IBM Company'
```

字符串的创建和访问——不可变

Source

```
>>> testStr = 'hello'
>>> testStr[0] = 'H'
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    testStr[0] = 'H'
TypeError: 'str' object does not support item assignment
>>> hStr
'The IBM Company'
>>> hStr = ''
>>> hStr
''
```

常用转义字符

字符	说明
\t	横向制表符
\n	换行
\r	回车
\"	双引号
\'	单引号
\\	反斜杠
\(在行尾时)	续行符
\ooo	值为八进制数ooo的字符
\xhh	值为十六进制数hh的字符

Source

```
>>> aStr = '\101\t\x41\n'
>>> bStr = '\141\t\x61\n'
>>> print(aStr, bStr)
A        A
a        a
```

字符串常用方法

capitalize()	center()	count()	encode()	endswith()	find()
format()	index()	isalnum()	isalpha()	isdigit()	islower()
isspace()	istitle()	isupper()	join()	ljust()	lower()
lstrip()	maketrans()	partition()	replace()	rfind()	rindex()
rjust()	rpartition()	rstrip()	split()	splitlines()	startswith()
strip()	swapcase()	title()	translate()	upper()	zfill()

字符串常用方法

center()

Source

```
>>> aStr = 'Python!'
>>> aStr.center(11)
'  Python!  '
```

count()

Source

```
>>> bStr = 'No pain, No gain.'
>>> bStr.count('no')
0
>>> bStr.count('No')
2
```

字符串小例子

?

给出一个字符串，不区分大小写，字符串中可能包含 ‘A’ - ‘Z’，‘a’ - ‘z’，‘ ’ (空格)等字符。输出字母a（包括大小写）出现的次数。测试数据：abc&ABC。

File

```
# Filename: char_count.py
s1 = "abc&ABC"
s = s1.lower()
n = s.count("a")
print(n)
```

字符串常用方法

find()

Source

```
>>> bStr = 'No pain, No gain. '   # 逗号后面有一个空格！
>>> bStr.find('No')
0
>>> bStr.find('no')
-1
>>> bStr.find('No', 3)
9
>>> bStr.find('No', 3, 10)
-1
>>> bStr.find('No', 3, 11)
9
```

字符串常用方法

index()

Source

```
>>> bStr = 'No pain, No gain. '   # 逗号后面有一个空格！
>>> bStr.index('no')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    bStr.index('no')
ValueError: substring not found
>>> bStr.index('No', 3, 10)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    bStr.index('No', 3, 10)
ValueError: substring not found
```

字符串常用方法

join()

Source

```
>>> ' love '.join(['I', 'Python!'])
'I love Python!'
>>> ' '.join(['Hello,', 'World'])
'Hello, World'
>>> '->'.join(('BA', 'The Boeing Company', '184.76'))
'BA->The Boeing Company->184.76‘
>>> '.'.join(('2020','1','1'))
'2020.1.1'
```

字符串常用方法

replace()

Source

```
>>> cStr = 'Hope is a good thing.'
>>> cStr.replace('Hope', 'Love')
'Love is a good thing.'
```

字符串常用方法

split()

Source

```
>>> '2020 1 1'.split()
['2020', '1', '1']
>>> dStr = 'I am a student.'
>>> dStr[:-1].split()
['I', 'am', 'a', 'student']
>>> '2020.1.1'.split('.')
['2020', '1', '1']
```

字符串常用方法

split()

Source

```
>>> nums = input('Enter the nums: ').split(',')
Enter the nums: 12,34,56
>>> nums
['12', '34', '56']
```

字符串的应用

有一些从网络上下载的类似如下形式的一些句子：
What do you think of this saying "No pain, No gain"?
对于句子中双引号中的内容，首先判断其是否满足标题格式，不管满足与否最终都将其转换为标题格式输出。

字符串的应用

File

```
# Filename: totitle.py
aStr = 'What do you think of this saying "No pain, No gain"?'
lindex = aStr.index('\"',0,len(aStr))
rindex = aStr.rindex('\"',0,len(aStr))
tempStr = aStr[lindex+1:rindex]
if tempStr.istitle():
    print('It is title format.')
else:
    print('It is not title format.')
print(tempStr.title())
```

```
tempStr= aStr.split("\"")[1]
```

字符串常用方法

strip()

Source

```
>>> "   hello\n".strip()
'hello'
```

3.3 列表

列表

B

可变的容器

A

经典的序列类型

C

包含不同类型元素

3.3.1 列表的表示

列表的表示

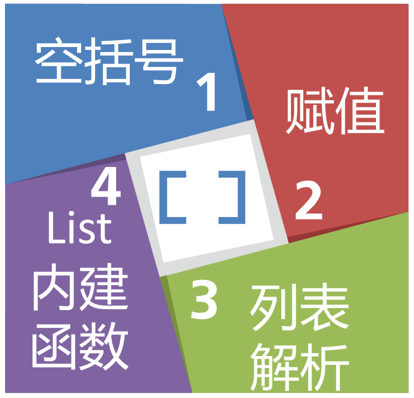
Source

```
>>> aList = ['P' , 'y', 't', 'h', 'o', 'n']
>>> pList = [1, 'BA', 'The Boeing Company', 184.76]
```

中括号

[ ]

列表的创建



Source

```
>>> aList = []
>>> pList = [1, 'BA', 'The Boeing Company', 184.76]
>>> cList = [x for x in range(1, 10, 2)]
>>> dList = list('Python')
>>> eList = [0] * 10
```

列表的创建

可扩展的容器对象

Source

```
>>> aList = list('Hello.')
>>> aList
['H', 'e', 'l', 'l', 'o', '.']
>>> aList = list('hello.')
>>> aList
['h', 'e', 'l', 'l', 'o', '.']
>>> aList[0] = 'H'
>>> aList
['H', 'e', 'l', 'l', 'o', '.']
```

包含不同类型对象

Source

```
>>> bList = [1, 2, 'a', 3.5]
```

列表的创建

- aList = [1, 2, 3, 4, 5]
- names = ['Zhao', 'Qian', 'Sun', 'Li']
- bList = [3, 2, 1, 'Action']
- pList = [('AXP', 'American Express Company', '78.51'),

```
             ('BA', 'The Boeing Company', '184.76'),
            ('CAT', 'Caterpillar Inc.', '96.39'),
            ('CSCO', 'Cisco Systems, Inc.', '33.71'),
            ('CVX', 'Chevron Corporation', '106.09')]
```

列表的操作

Source

```
>>> pList = [('AXP', 'American Express Company', '78.51'),
             ('BA', 'The Boeing Company', '184.76'),
             ('CAT', 'Caterpillar Inc.', '96.39'),
             ('CSCO', 'Cisco Systems, Inc.', '33.71'),
             ('CVX', 'Chevron Corporation', '106.09')]
>>> pList[1][1]
'The Boeing Company'
>>> pList[:2]
[('AXP', 'American Express Company', '78.51'), ('BA', 'The Boeing
Company', '184.76')]
```

列表的操作

Source

```
>>> fList = list('hello')
['h', 'e', 'l', 'l', 'o']
>>> fList[0] = 'H'
>>> fList
['H', 'e', 'l', 'l', 'o']
```

可变的列表可以修改元素值

[ ]

列表的方法

append()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()

参数的作用： list.sort(key=None, reverse=False)

Source

```
>>> numList = [3, 11, 5, 8, 16, 1]
>>> fruitList = ['apple', 'banana', 'pear', 'lemon', 'avocado']
>>> numList.sort(reverse = True)
>>> numList
[16, 11, 8, 5, 3, 1]
>>> fruitList.sort(key = len)
>>> fruitList
['pear', 'apple', 'lemon', 'banana', 'avocado']
```

列表的方法

append()

Source

```
>>> aList = [1, 2, 3]
>>> aList.append(4)
>>> aList
[1, 2, 3, 4]
>>> aList.append([5, 6])
>>> aList
[1, 2, 3, 4, [5, 6]]
>>> aList.append('Python!')
>>> aList
[1, 2, 3, 4, [5, 6], 'Python!']
```

列表的方法

extend()

Source

```
>>> bList = [1, 2, 3]
>>> bList.extend([4])
>>> bList
[1, 2, 3, 4]
>>> bList.extend([5, 6])
>>> bList
[1, 2, 3, 4, 5, 6]
>>> bList.extend('Python!')
>>> bList
[1, 2, 3, 4, 5, 6, 'P', 'y', 't', 'h', 'o', 'n', '!']
```

列表的方法

extend()

Source

```
>>> bList = [1, 2, 3]
>>> bList.extend(4)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    bList.extend(4)
TypeError: 'int' object is not iterable
```

列表的方法

Source

copy()

```
>>> a = [1, 2, [3, 4]]
>>> b = a.copy()     # b = a[:] 也是浅拷贝
>>> b
[1, 2, [3, 4]]
>>> b[0], b[2][0] = 5, 5
>>> b
[5, 2, [5, 4]]
>>> a
[1, 2, [5, 4]]
```

浅拷贝

列表的方法

copy()

深拷贝

Source

```
>>> import copy
>>> a = [1, 2, [5, 4]]
>>> c = copy.deepcopy(a)
>>> c
[1, 2, [5, 4]]
>>> c[0], c[2][0] = 8, 8
>>> c
[8, 2, [8, 4]]
>>> a
[1, 2, [5, 4]]
```

列表的方法

pop()

Source

```
>>> scores = [7, 8, 8, 8, 8.5, 9, 9, 9, 10, 10]
>>> scores.pop()
10
>>> scores
[7, 8, 8, 8, 8.5, 9, 9, 9, 10]
>>> scores.pop(4)
8.5
>>> scores
[7, 8, 8, 8, 9, 9, 9, 10]
```

列表的方法

remove()

Source

```
>>> jScores = [7, 8, 8, 8, 9, 9, 9, 10]
>>> jScores.remove(9)
>>> jScores
[7, 8, 8, 8, 9, 9, 10]
```

列表的方法

reverse()

Source

```
>>> week = ['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.', 'Sun.']
>>> week.reverse()
>>> week
['Sun.', 'Sat.', 'Fri.', 'Thur.', 'Wed.', 'Tues.', 'Mon.']
```

列表的方法

列表.reverse()

- 列表的方法
- 在原列表上直接翻转，并得到逆序列表，改变原列表内容。

reversed()

- 序列类型的内建函数
- 返回的是序列逆序排序后的迭代器，原列表内容不变。

字符串和元组（字符串和元组都是不可变的）没有reverse()方法

列表的方法

sort()

Source

```
>>> jScores = [9, 9, 8.5, 10, 7, 8, 8, 9, 8, 10]
>>> jScores.sort()
>>> jScores
[7, 8, 8, 8, 8.5, 9, 9, 9, 10, 10]
>>> numList = [3, 11, 5, 8, 16, 1]
>>> fruitList = ['apple', 'banana', 'pear', 'lemon', 'avocado']
>>> numList.sort(reverse = True)
>>> numList
[16, 11, 8, 5, 3, 1]
>>> fruitList.sort(key = len)
>>> fruitList
['pear', 'apple', 'lemon', 'banana', 'avocado']
```

列表的方法

列表.sort()

- 列表的方法
- 对原列表排序，改变原列表内容。

sorted()

- 序列类型的内建函数
- 返回的是排序后的新列表，原列表内容不变。

字符串和元组（字符串和元组都是不可变的）没有sort()方法

列表的应用

某学校组织了一场校园歌手比赛，每个歌手的得分由10名评委和观众决定，最终得分的规则是去掉10名评委所打分数的一个最高分和一个最低分，再加上所有观众评委分数后的平均值。评委打出的10个分数为：9、9、8.5、10、7、8、8、9、8和10，观众评委打出的综合评分为9，请计算该歌手的最终得分。

列表的应用

File

```
# Filename: scoring.py
jScores = [9, 9, 8.5, 10, 7, 8, 8, 9, 8, 10]
aScore = 9
jScores.sort()
jScores.pop()
jScores.pop(0)
jScores.append(aScore)
aveScore = sum(jScores)/len(jScores)
print(aveScore)
```

```
[7, 8, 8, 8, 8.5, 9, 9, 9, 10, 10]
[8, 8, 8, 8.5, 9, 9, 9, 10]
[8, 8, 8, 8.5, 9, 9, 9, 10, 9]
8.72222222222
```

列表的应用

有一份参加Python课程的学号名单B01,B02,B03,B05,B08,B10，请计算共有多少同学参与了本课程。请分别用列表和字符串的方法来解决这个问题。

列表的应用

File

```
# Filename: count.py
lst = ['B01','B02','B03','B05','B08','B10']
s = "B01,B02,B03,B05,B08,B10"
num1 = len(lst)
print(num1)
num2 = s.count(',') + 1
num3 = len(s.split(','))
print(num2, num3)
```

3.4

元组

元组的创建

圆括号

( )

Source

```
>>> aTuple = (1, 2, 3)
>>> aTuple
(1, 2, 3)
>>> 2020,
(2020,)
>>> k = 1, 2, 3
>>> k
(1, 2, 3)
```

元组的操作

序列通用：
切片、求长度

( )

Source

```
>>> bTuple = (['Monday', 1], 2,3)
>>> bTuple
(['Monday', 1], 2, 3)
>>> bTuple[0][1]
1
>>> len(bTuple)
3
>>> bTuple[1:]
(2, 3)
```

元组的操作

元组不可变

( )

Source

```
>>> aList = ['AXP', 'BA', 'CAT']
>>> aTuple = ('AXP', 'BA', 'CAT')
>>> aList[1] = 'INTC'
>>> print(aList)
['AXP', 'INTC', 'CAT']
>>> aTuple[1]= 'INTC'
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    aTuple[1]= 'INTC'
TypeError: 'tuple' object does not support item
assignment
>>> aTuple.sort()
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    aTuple.sort()
AttributeError: 'tuple' object has no attribute 'sort'
```

元组

Source

```
>>> aList = [3, 5, 2, 4]
>>> aList
[3, 5, 2, 4]
>>> sorted(aList)
[2, 3, 4, 5]
>>> aList
[3, 5, 2, 4]
>>> aList.sort()
>>> aList
[2, 3, 4, 5]
```

Source

```
>>> aTuple = (3, 5, 2, 4)
>>> sorted(aTuple)
[2, 3, 4, 5]
>>> aTuple
(3, 5, 2, 4)
>>> aTuple.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
```

元组

sort()

- 元组没有sort方法。

sorted()

- 序列的内建函数
- 返回排序新列表，原列表内容不变

3.4.2 元组的其他特性和作用

元组特性

元组的
可变元素可变

( )

Source

```
>>> bTuple = (1, 2, [3, 4])
>>> bTuple[2] = [5, 6]
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    bTuple[2] = [5, 6]
TypeError: 'tuple' object does not support item assignment
>>> bTuple[2][0] = 5
>>> bTuple
(1, 2, [5, 4])
```

元组的作用

元组用在什么地方？

在映射类型中当作键使用

函数的特殊类型参数

未明确定义的一组对象

函数返回一组值、末明确定义列表还是元组

返回对象的个数	返回类型
0	None
1	object
>1	tuple

Source

```
>>> def foo():
        return 1, 2, 3
>>> foo()
(1, 2, 3)
>>> 1,2,3
(1, 2, 3)
```

3.5

range对象

range对象

- 用range()函数生成range对象，执行时一边计算一边产生值（类似一个生成器），生成一个不可变的整数序列

```
range(start, end, step=1)
range(start, end)
range(end)
```

range对象

Source

```
>>> list(range(3, 11))
[3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(11))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(3, 11, 2))
[3, 5, 7, 9]
```

Source

```
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> list(range(0))
[]
>>> list(range(1, 0))
[]
```

小结

- 序列
- 字符串
- 列表
- 元组
- range对象