人工智能程序设计

M1 Python程序设计基础

2 数据类型

张 莉

数据类型

- 如何表示年龄?
- 如何表示性别?
- 如何分离一个长整数?
- 如何表示用户账号?
- …

人工智能程序设计

1 序列

序列

序列是一种最基本最重要的容器(container)，主要包括字符串，列表，元组和range对象

- 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')]

1. 索引

2. 标准类型运算

3. 通用序列类型操作

4. 序列类型函数

序列的索引

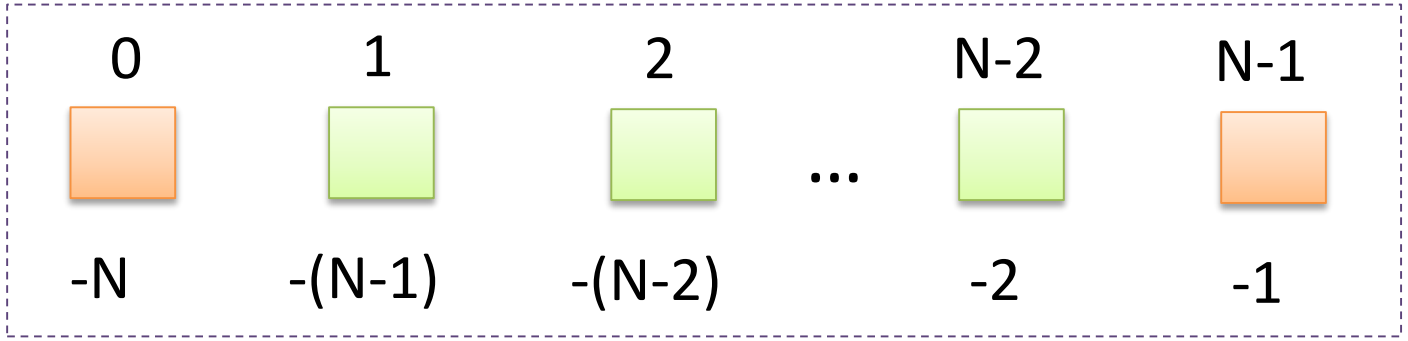
- 序列类型对象一般有多个成员组成，每个成员通常称为元素，每个元素都可以通过索引（index）进行访问，索引用方括号“[]”表示。

```
sequence[index]
```

序列的索引

	0	1	2	3	4	5	6
week	'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

标准类型运算符

值比较

对象身份比较

布尔运算

序列类型运算符

切片

重复

连接

成员判断

内建函数

序列类型转换内建函数

序列类型可用内建函数

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

Source

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

Source

```
>>> ['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

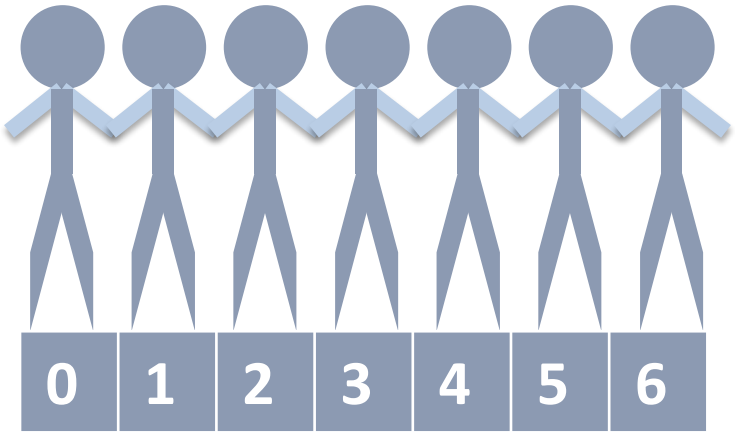
Source

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

序列类型运算符

x in s	成员判断
x not in s	
s + t	连接
s * n, n * s	重复
s[i]	切片
s[i:j]	
s[i:j:k]	

切片



索引值

Source

```
>>> 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

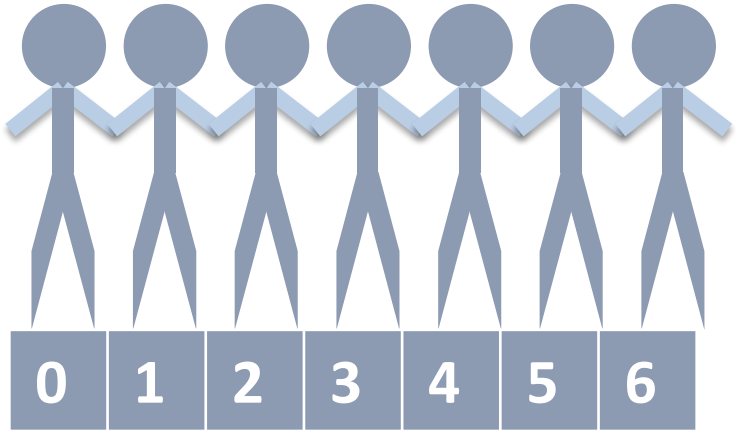
```
>>> aList[-2: -1]
['Sat.']
>>> aList[-2: -3]
[]
>>> aList[-2:]
['Sat.', 'Sun.']
>>> aList[:]
['Mon.', 'Tues.', 'Wed.', 'Thur.', 'Fri.', 'Sat.', 'Sun.']
```

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

```
>>> 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
```

序列类型转换内建函数

list()
str()
tuple()

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)
```

序列类型其他常用内建函数

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

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 0x00000180243 61B70>
>>> 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

Source

max()和min()

```
>>> 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')]
```

人工智能程序设计

2 字符串

字符串的表示形式

Source

```
>>> 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

字符串的创建和访问

Source

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

创建方式：

赋值

直接输出

访问方式：

切片

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

```
>>> hStr
'The IBM Company'
>>> hStr = ''
>>> hStr
''
>>> 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
```

字符串的表示形式

转义
字符

Source

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

常用转义字符

字符	说明
\t	横向制表符
\n	换行
\r	回车
\e	转义
\"	双引号
\'	单引号
\\	反斜杠
\(在行尾时)	续行符

\OOO　八进制数OOO代表的字符
\xXX　十六进制数XX代表的字符

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'
```

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']
>>> '2020.1.1'.split('.')
['2020', '1', '1']
```

字符串的应用

使用以下语句存储一个字符串：mark = 'My GPA is: 3.5.'，从字符串mark中提取出GPA的值（3.5），结果为浮点类型。

File

```
# Filename: gpa_find.py
mark = 'My GPA is: 3.5.'
gpa_temp = mark.split(':')[1]
gpa = float(gpa_temp[:-1])
print(gpa)
```

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]
```

人工智能程序设计

3 列表

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')
```

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]
('BA', 'The Boeing Company', '184.76')
>>> pList[1][1]
'The Boeing Company'
```

列表的操作

Source

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

deepcopy()

深拷贝

Source

```
>>> import copy
>>> a = [1, 2, [5, 4]]
>>> c = copy.deepcopy(a)   # copy.copy也是浅拷贝
>>> 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
```

人工智能程序设计

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] = 'Alibiabia'
>>> print(aList)
['AXP', 'Alibiabia', 'CAT']
>>> aTuple1[1]= 'Alibiabia'
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    aTuple1[1]= 'Alibiabia'
NameError: name 'aTuple1' is not defined
>>> 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

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

元组作为函数特殊返回类型

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)
```

人工智能程序设计

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))
[]
```

人工智能程序设计

6 字典

为什么要使用字典?

某公司人事部门让技术部门用Python构建一个简易的员工信息表，包含员工的姓名和工资信息。根据信息表查询员工Linlin的工资。

Source

```
>>> names = ['Mayue', 'Lilin', 'Wuyun']
>>> salaries = [3000, 4500, 8000]
>>> print(salaries[names.index('Lilin')])
4500
```

salaries['Lilin']

字典

- 什么是字典?——一种映射类型
  - 键（key）
  - 值（value）
  - key-value对

键是唯一的:
数字
字符串
元组

字典

- aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}

key	value
'Mayue'	3000
'Lilin'	4500
'Wuyun'	8000

Source

```
>>> sorted(aInfo)
['Lilin', 'Mayue', 'Wuyun']
```

1. 创建字典

2. 生成字典

3. 字典的基本操作

4. 字典的函数与方法

创建字典

直接创建

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
```

创建字典

用dict()函数创建

Source

```
>>> info = [('Mayue', 3000), ('Lilin', 4500), ('Wuyun', 8000)]
>>> bInfo = dict(info)
>>> print(bInfo)
{'Lilin': 4500, 'Wuyun': 8000, 'Mayue': 3000}
>>> cInfo = dict([['Mayue', 3000], ['Lilin', 4500], ['Wuyun', 8000]])
>>> dInfo = dict(Mayue = 3000, Lilin = 4500, Wuyun = 8000)
>>> eInfo = dict(('Mayue', 3000), ('Lilin', 4500), ('Wuyun', 8000)))
```

创建字典

用方法fromkeys(seq[, value])创建

Source

```
>>> gInfo = {}.fromkeys(('Mayue', 'Lilin', 'Wuyun'), 3000)
>>> print(gInfo)
{'Lilin': 3000, 'Mayue': 3000, 'Wuyun': 3000}
```

创建员工信息表时将所有员工的工资默认值设置为3000

生成字典

已知有姓名列表和工资列表，如何生成字典类型的员工信息表?

Source

```
>>> names = ['Mayue', 'Lilin', 'Wuyun']
>>> salaries = [3000, 4500, 8000]
>>> dict(zip(names,salaries))
{'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
```

生成字典

对于几个公司的财经数据，如何构造公司代码和股票价格的字典?

```
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')]
```

生成字典

对于几个公司的财经数据，如何构造公司代码和股票价格的字典?

```
aDict = {'AXP': '78.51', 'BA': '184.76', 'CAT ': '96.39',
'CSCO': '33.71', 'CVX': '106.09'}
```

算法分析：可用循环将公司代码和股票价格分别append到一个新列表中，再利用zip()和dict()函数将这两个列表转化成字典；其他方法。

字典的基本操作

- 键值查找
- 字典更新
- 添加元素
- 成员判断
- 删除元素

1. 键值查找

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> aInfo['Lilin']
4500
```

2. 字典更新

Source

```
>>> aInfo['Lilin'] = 9999
>>> aInfo
{'Wuyun': 8000, 'Mayue': 3000, 'Lilin': 9999}
```

3. 添加元素

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> aInfo['Liuxi'] = 6000
>>> aInfo
{'Wuyun': 8000, 'Liuxi': 6000, 'Mayue': 3000, 'Lilin': 9999}
```

4. 成员判断

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> 'Liuyun' in aInfo
False
```

5. 删除元素

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> del aInfo['Lilin']
>>> aInfo
{'Mayue': 3000, 'Wuyun': 8000}
```

字典的内建函数

dict()
len()
hash()

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> len(aInfo)
3
>>> hash('Mayue')
7716305958664889313
```

字典的内建函数

Source

```
>>> hash('Wangdachui')
7716305958664889313
>>> testList = [1, 2, 3]
>>> hash(testList)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    hash(testList)
TypeError: unhashable type: 'list'
```

字典方法

clear()	copy()	fromkeys()	get()	items()
keys()	pop()	setdefault()	update()	values()

字典方法

keys()

values()

items()

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> aInfo.keys()
dict_keys(['Mayue', 'Lilin', 'Wuyun'])
>>> aInfo.values()
dict_values([3000, 4500, 8000])
>>> aInfo.items()
dict_items([('Mayue', 3000), ('Lilin', 4500), ('Wuyun', 8000)])
```

字典方法

get()

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> print(aInfo.get('Lilin'))
4500
>>> print(aInfo.get('Qiqi'))
None
>>> print(aInfo.get('Qiqi', 7000))
7000
>>> aInfo
{'Lilin': 4500, 'Mayue': 3000, 'Wuyun': 8000}
```

字典方法

下面两个程序都通过键查找值，区别在哪里？你更喜欢哪一个？

Source

```
>>> stock = {'AXP': 78.51, 'BA': 184.76}
>>> stock['AAA']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'AAA'
```

Source

```
>>> stock = {'AXP': 78.51, 'BA': 184.76}
>>> print(stock.get('AAA'))
None
```

字典方法

setdefault()

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> aInfo.setdefault('Lilin', None)
# 与aInfo.get('Lilin')和aInfo.setdefault('Lilin')效果一样
4500
>>> aInfo.setdefault('Jinhe', None)
# 与aInfo.setdefault('Jinhe')效果一样
>>> aInfo.setdefault('Qiqi', 8000)
8000
>>> aInfo
{'Jinhe': None, 'Lilin': 4500, 'Mayue': 3000, 'Qiqi': 8000, 'Wuyun': 8000}
```

字典方法

copy()

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> aInfoBackup = aInfo.copy()
>>> aInfoBackup
{'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
```

字典方法

pop()

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> aInfo.pop('Lilin')
4500
>>> aInfo
{'Mayue': 3000, 'Wuyun': 8000}
```

字典方法

clear()

Source

```
>>> aInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> aInfo.clear()
>>> aInfo
{}
```

字典方法

update()

Source

```
>>> anfo={}
>>> bInfo = {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> aInfo.update(bInfo)
>>> aInfo
{'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> cInfo = {'Mayue': 4000, 'Wanqi':6000, 'Lilin': 9999}
>>> aInfo
{'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
>>> aInfo.update(cInfo)
>>> aInfo
{'Mayue': 4000, 'Lilin': 9999, 'Wanqi': 6000, 'Wuyun': 8000}
```

字典方法简单应用

已知有员工姓名和工资信息表{'Wangdachui':3000, 'Niuyun':2000, 'Linling':4500, 'Tianqi':8000}，如何单独输出员工姓名和工资金额?

Source

```
>>> aInfo = {'Wangdachui': 3000, 'Niuyun': 2000, 'Lilin': 4500, 'Tianqi': 8000}
>>> aInfo.keys()
dict_keys(['Tianqi', 'Wangdachui', 'Niuyun', 'Lilin'])
>>> aInfo.values()
dict_values([8000, 3000, 2000, 4500])
```

字典方法简单应用

人事部门有两份人员和工资信息表，第一份是原有信息，第二份是公司中有工资更改人员和新进人员的信息，如何处理可以较快地获得完整的信息表?

Source

```
>>> aInfo = {'Wangdachui': 3000, 'Niuyun': 2000, 'Lilin': 4500}
>>> bInfo = {'Wangdachui': 4000, 'Niuyun': 9999, 'Wangzi': 6000}
>>> aInfo.update(bInfo)
>>> aInfo
{'Wangzi': 6000, 'Lilin': 4500, 'Wangdachui': 4000, 'Niuyun': 9999}
```

字典应用

创建一个字典，键保存用户名，值保存密码。设计一个登录检查程序，只有用户名和密码都正确的用户才能通过登录检查程序。

```
d  = {'dazhuang':'123', 'xiaomi':'456'}
name, passwd = input('enter the account: ').split(',')
if d.get(name) == passwd:
    print('Bingo')
else:
    print('Nani')
```

人工智能程序设计

7 集合

集合

人事部门的一份工资信息表登记时由于工作人员的疏忽有部分姓名重复登记了，如何快速解决这个问题?

Source

```
>>> names = ['Wangdachui', 'Niuyun', 'Wangzi', 'Wangdachui', 'Lilin', 'Niuyun']
>>> namesSet = set(names)
>>> namesSet
{'Wangzi', 'Wangdachui', 'Niuyun', 'Lilin'}
```

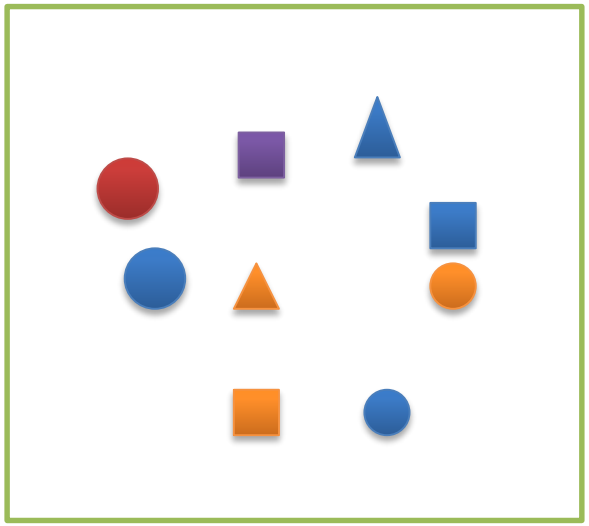
集合

- 什么是集合?

  一个无序不重复的元素的组合

  – 可变集合（set）

  – 不可变集合（frozenset）



去重

集合

Source

```
>>> names = ['Mayue', 'Lilin', 'Wanqi', 'Mayue', 'Lilin']
>>> names
['Mayue', 'Lilin', 'Wanqi', 'Mayue', 'Lilin']
>>> nameset = set(names)
>>> nameset
{'Mayue', 'Wanqi', 'Lilin'}
>>> type(nameset)
<class 'set'>
```

集合的创建

大括号

{}

Source

```
>>> Set = {1, 2, 3}
>>> aSet = set('hello')
>>> aSet
{'h', 'e', 'l', 'o'}
>>> fSet = frozenset('hello')
>>> fSet
frozenset({'h', 'e', 'l', 'o'})
>>> type(aSet)
<class 'set'>
>>> type(fSet)
<class 'frozenset'>
```

集合的基本操作

Source

```
>>> aSet = set('sunrise')
>>> bSet = set('sunset')
>>> 'u' in aSet
True
>>> aSet == bSet
False
>>> aSet < bSet
False
>>> set('sun') < aSet
True
```

数学符号	Python符号
∈	in
∉	not in
=	==
≠	!=
⊂	<
⊆	<=
⊃	>
⊇	>=

标准类型运算符

集合的基本操作

Source

```
>>> aSet = set('sunrise')
>>> bSet = set('sunset')
>>> aSet & bSet
{'u', 's', 'e', 'n'}
>>> aSet | bSet
{'e', 'i', 'n', 's', 'r', 'u', 't'}
>>> aSet - bSet
{'i', 'r'}
```

Source

```
>>> aSet = set('sunrise')
>>> bSet = set('sunset')
>>> aSet ^ bSet
{'i', 'r', 't'}
>>> aSet -= set('sun')
>>> aSet
{'e', 'i', 'r'}
```

数学符号	Python符号
∩	&
∪	|
- 或 \	-
Δ	^

集合类型运算符

运算符可复合

&= |= -= ^=

集合方法

面向
所有集合

issubset(t)
issuperset(t)
union(t)
intersection(t)
difference(t)
symmetric_difference(t)
copy()

Source

```
>>> aSet = set('sunrise')
>>> bSet = set('sunset')
```

Source

```
>>> aSet.issubset(bSet)
False
>>> aSet.intersection(bSet)
{'u', 's', 'e', 'n'}
>>> aSet.difference(bSet)
{'i', 'r'}
>>> aSet.symmetric_difference(bSet)
{'i', 't', 'r'}
>>> cSet = aSet.copy()
>>> cSet
{'s', 'r', 'e', 'i', 'u', 'n'}
```

集合方法

面向
可变集合

update(t)
intersection_update(t)
difference_update(t)
symmetric_difference_update(t)
add(obj)
remove(obj)
discard(obj)
pop()
clear()

集合方法

面向可变集合

Source

```
>>> aSet = set('sunrise')
>>> aSet.add('!')
>>> aSet
{'!', 'e', 'i', 'n', 's', 'r', 'u'}
>>> aSet.remove('!')
>>> aSet
{'e', 'i', 'n', 's', 'r', 'u'}
>>> aSet.discard('a')
>>> aSet
{'s', 'u', 'e', 'i', 'n', 'r'}
```

Source

```
>>> aSet.remove('a')
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in
<module>
    aSet.remove('a')
KeyError: 'a'
>>> aSet.update('Yeah')
>>> aSet
{'a', 'e', 'i', 'h', 'n', 's', 'r', 'u', 'Y'}
>>> aSet.clear()
>>> aSet
set()
```

M1.2 小结

01 序列

02 字符串

03 列表

04 元组

05 range对象

06 字典

07 集合