第 3 章

Python 序列结构

数据结构(data structure)是相互之间存在一种或多种特定关系的数据元素的集合,这些数据元素可以是数字或字符,也可以是其他类型的数据结构。

Python 中常见的数据结构可以统称为容器(container)。序列(如列表和元组)、映射(如字典)以及集合(set)是主要的三类容器。

在 Python 语言中,序列(sequence)是最基本的数据结构。序列中,给每一个元素分配一个序列号(即元素的位置),该位置称为索引。Python 中包含 6 种内建序列,即列表、元组、字符串、Unicode 字符串、buffer 对象和 xrange 对象。本书介绍最常用的两种:列表与元组。

3.1 列　　表

列表(list)是 Python 语言中最通用的序列数据结构之一。列表是一个没有固定长度、用来表示任意类型对象、位置相关的有序集合。列表的数据项不需要具有相同的类型,常用的列表操作主要包括索引、分片、连接、乘法和检查成员等。列表中的每个元素都分配一个数字——它的位置(索引),第一个索引是 0,第二个索引是 1,依次类推。

3.1.1 列表的基本操作

1. 创建列表

创建一个列表,只要把用逗号分隔的不同的数据项用方括号括起来即可。例如:

```
>>>list1 =['Google', 'python', 2018, 2019]
>>>list2 =[1, 2, 3, 4, 5 ]
>>>list3 =["a", "b", "c", "d"]
```

2. 访问列表

可以使用下标索引来访问列表中的值,同样也可以使用方括号的形式截取字符。例如:

```
>>>list1 =['Google', 'python', 2018, 2019]
>>>list2 =[1, 2, 3, 4, 5, 6, 7 ]
>>>print ("list1[0]:", list1[0])            #输出结果为 list1[0]:Google
>>>print ("list2[1:5]:", list2[1:5])        #输出结果为 list2[1:5]:[2,3,4,5]
```

3. 列表元素赋值

列表元素的赋值主要包括列表整体赋值和列表指定位置赋值两种方法。例如:

```
>>>x=[1,2,3,4,5]
>>>x
[1,2,3,4,5]
>>>x[2]=6
>>>x
[1,2,6,4,5]
```

注意:程序设计中不能对不存在的位置进行赋值。在上例中,列表 x 内只包含 5 个元素,如果运行"x[5]=6",则会出现"IndexError:list assignment index out of range"的错误提示,提出索引超出范围。

4. 列表元素删除

可以使用 del 语句删除列表的元素。例如:

```
>>>list =['Google', 'python', 2018, 2019]
>>>list
['Google', 'python', 2018, 2019]
>>>del list1[2]
>>>list1
['Google', 'python', 2019]
```

与列表元素赋值相似,列表元素的删除只能针对已有元素进行删除,否则也会产生索引超出范围的错误提示。

5. 列表分片赋值

分片操作可以用来访问一定范围内的元素,也可以用来提取序列的一部分内容。分片操作通过使用冒号相隔的两个索引来实现,第一个索引的元素包含在片内,第二个索引的元素不包含在片内。例如:

```
>>>list1=[1,2,3,4,5,6,7]
>>>print(list1[1:3])
[2,3]                                    #输出分片结果
>>>list2= list[1:3]                     #分片并赋值
>>>print(list2)
[2,3]
```

注意:在 list 偏移访问中,与 C 语言中数组类似,同样是用 list[0]表示列表第一个元素;2、3 分别是列表中第二、三个元素,索引分别是 1 和 2,由此可以看出,对于 list[x:y]的切片段为序号 x 到 y−1 的内容。

6. 列表组合

操作符"+"用于组合列表,它的作用是把"+"两边的列表组合起来得到一个新的列表。例如:

```
>>>list1=[1,2,3]
>>>list2=[4,5,6]
>>>list3=list1+list2
```

```
>>>print(list3)
[1,2,3,4,5,6]
```

7. 列表重复

操作符“*”用于重复列表,它的作用是对列表中的元素重复指定次数。例如:

```
>>>list1=["python is easy"]
>>>list2=list1*4
>>>print(list2)
['python is easy', 'python is easy', 'python is easy', 'python is easy']
```

3.1.2　列表的常用方法

方法是一个与对象有着密切关联的函数,方法的调用格式为

```
对象.方法(参数)
```

列表的常用方法和函数见表 3-1。

表 3-1　列表的常用方法和函数

方法和函数	说　明
count()	统计某元素在列表中出现的次数
append()	在列表末尾追加新的对象
extend()	在列表的末尾一次性追加另一个序列中的多个值
insert()	将对象插入列表中
pop()	移除列表中的一个元素,并返回该元素的值
remove()	用于移除列表中某个值的第一个匹配项
reverse()	将列表中的元素反向存储
sort()	对列表进行排序
index()	在列表中找出某个值第一次出现的位置
cmp()	用于比较两个列表的元素
clear()	清空列表
copy()	复制列表
len()	返回列表元素个数
max()	返回列表元素中的最大值
min()	返回列表元素中的最小值
list()	将元组转换为列表
in	判断列表是否存在指定元素

1. count()

count()方法可以用于统计列表中某元素出现的次数。其语法格式为

```
对象.count(obj)
```

其中,obj 表示列表中统计的对象。该方法返回元素在列表中出现的次数。例如:

```
>>>list1=['h','a','p','p','y']
>>>list1.count('p')
2
```

count()方法可以统计列表中任意元素的出现次数,该元素可以是数字、字母、字符串甚至是其他列表。例如:

```
>>>list1=[[7,1],2,2,[1,7]]
>>>list1.count([7,1])
1
>>>list1.count(2)
2
```

2. append()

append()方法用于在列表末尾追加新的对象。其语法格式为

```
对象.append(obj)
```

其中,obj 表示添加到列表末尾的对象。该方法无返回值,但是会修改原来的列表。例如:

```
>>>list1=["Google","python"]
>>>list1.append("baidu")
>>>list1
['Google','python', 'baidu']
```

3. extend()

extend()方法可以在列表的末尾一次性追加一个新的序列中的值。与序列的连接操作不同,使用该方法修改了被扩展的序列,而连接只是返回一个新的序列。其语法格式为

```
对象.extend(seq)
```

其中,seq 表示元素列表,它可以是列表、元组、集合、字典,若为字典,仅会将键(key)作为元素依次添加至原列表的末尾。该方法没有返回值,但会在已存在的列表中添加新的列表内容。例如:

```
>>>list1=['a','b','c']
>>>list2=['d','e']
>>>list1.extend(list2)
>>>list1
['a','b','c','d','e']
>>>list3=['a','b','c']
>>>list4=['d','e']
>>>list3+list4                                  #使用连接,返回新的序列
['a','b','c','d','e']
>>>list3
['a','b','c']
```

4. insert()

insert()方法可以在指定位置插入新的元素。其语法格式为

```
对象.insert(index,obj)
```

其中,index 表示对象 obj 需要插入的索引位置;obj 表示要插入列表中的对象。该方法没有返回值,但会在列表指定位置插入对象。例如:

```
>>>list1=['Google','python','baidu']
>>>list1.insert(1,'taobao')
```

```
>>>list1
['Google','taobao','python','baidu']
```

5. pop()

pop()方法用于移除列表中的一个元素(默认最后一个元素),并且返回该元素的值。其语法格式为

```
对象.pop(index)
```

其中,index 为可选参数,表示要移除列表元素的索引值不能超过列表的总长度,默认 index 的值为－1,删除最后一个列表值。该方法返回从列表中移除的元素对象。例如:

```
>>>list1=['Google','python','baidu']
>>>list1.pop(1)
'python'
>>>list1.pop()
'baidu'
```

6. remove()

remove()方法用于删除列表中某元素值的第一个匹配项。其语法格式为

```
对象.remove(obj)
```

其中,obj 表示列表中要移除的对象。该方法没有返回值,但是会移除列表中的某个值的第一个匹配项。例如:

```
>>>list1=['Google','python','baidu','taobao','python']
>>>list1.remove('baidu')
>>>list1
['Google','python','taobao','python']
>>>list1.remove('python')
>>>list1
['Google','taobao','python']
```

7. reverse()

reverse()方法可以实现列表的反向存放。其语法格式为

```
对象.reverse()
```

该方法没有返回值,但是会对列表的元素进行反向排序。例如:

```
>>>list1=['Google','python','baidu','taobao']
>>>list1.reverse()
>>>list1
['taobao','baidu','python','Google']
```

8. sort()

sort()方法用于对原列表进行排序,如果指定参数,则使用指定的比较函数进行排序。其语法格式为

```
对象.sort(key= none,reverse= False)
```

其中,key 是用来进行比较的元素,只有一个参数,具体函数的参数取自可迭代对象中,指定可迭代对象中的一个元素来进行排序;reverse 表示排序规则,如果值为 True 表示降序,值为 False 表示升序,默认为升序。该方法没有返回值,但是会对列表的对象进行排序。例如:

```
>>>list1=['Google','python','baidu','taobao']
>>>list1.sort()                              #按 ASCII 值进行比较排序
>>>list1
['Google','baidu','python','taobao']
>>>list2=['e','a','f','o','i']
>>>list2.sort(reverse=True)                  #降序排列
>>>list2
['o','i','f','e','a']
>>>list3= ['luo','lu','l']
>>>list3.sort(key= len)                      #指定长度进行升序排列
>>>list3
['l','lu','luo']
```

说明:数字、字符串按照 ASCII 值的大小进行排序,中文按照 unicode 从小到大排序。

9. 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。其语法格式为

```
对象.index(obj)
```

其中,obj 表示查找的对象。该方法返回查找对象的索引位置,如果没有找到对象则抛出异常。例如:

```
>>>list1=['Google','python','baidu','taobao']
>>>list1.index('python')
1
```

10. clear()

clear()方法用于清空列表,类似于 del a[:]。其语法格式为

```
对象.clear()
```

该方法没有返回值。例如:

```
>>>list1=['Google','python','baidu','taobao']
>>>list1.clear()
>>>list1
[]
```

11. copy()

copy()方法用于复制列表,类似于 a[:]。其语法格式为

```
对象.copy()
```

该方法返回复制后的新列表。例如:

```
>>>list1=['Google','python','baidu','taobao']
>>>list1.copy()
['Google','python','baidu','taobao']
```

12. len()

len()函数返回列表元素个数。其语法格式为

```
len(list)
```

其中,list 表示要计算元素个数的列表。例如:

```
>>>list1=['Google','python','baidu','taobao']
>>>len(list1)
4
```

13. max()

max()函数返回列表元素中的最大值。其语法格式为

```
max(list)
```

其中,list 表示要返回最大值的列表。例如:

```
>>>list1=['Google','python','baidu','taobao']
>>>max(list1)
'taobao'
>>>list2=[456,251,702]
>>>max(list2)
702
```

14. min()

min()函数返回列表元素中的最小值。其语法格式为

```
min(list)
```

其中,list 表示要返回最小值的列表。例如:

```
>>>list1=['Google','python','baidu','taobao']
>>>min(list1)
Google
>>>list2=[456,251,702]
>>>min(list2)
251
```

15. list()

list()函数用于将元组或字符串转换为列表。其语法格式为

```
list(seq)
```

其中,seq 表示要转换为列表的元组或字符串。该函数返回的是列表。例如:

```
>>>a=('Google','python','baidu',123)
>>>list1=list(a)
```

```
>>>list1
['Google','python','baidu',123]
>>>str="hello python"
>>>list2=list(str)
>>>list2
['h','e','l','l','o',' ','p','y','t','h','o','n']
```

16. in

in 操作符用于判断元素是否存在列表中,如果元素在列表里则返回 True,否则返回 False。not in 操作符刚好相反,如果元素在列表里则返回 False,否则返回 True。其语法格式为

```
x in l
```

其中,x 表示需要判断是否存在的元素;l 表示列表。例如:

```
>>>list1=[1,2,3,4,5]
>>>1 in list1
True
>>>10 in list1
False
>>>10 not in list1
True
```

3.1.3 与列表相关的函数

1. sum()

sum()函数用于返回列表中所有元素之和,列表中的元素必须为数值。

```
>>>sum([1,2,3])
6
```

2. zip()

zip()函数用于将多个列表中元素重新组合为元组,并返回包含这些元组的 zip 对象。

```
>>>list1=["a","b","c"]
>>>list2=[1,2,3]
>>>list3=zip(list1,list2)
>>>type(list3)
<class 'zip'>
>>>list(list3)                                    #将 zip 对象转换成列表
[('a', 1), ('b', 2), ('c', 3)]
```

3. enumerate()

enumerate()函数用于返回包含若干下标和元素的迭代对象。

```
>>>list1=["a","b","c","d","e"]
>>>list2=enumerate(list1)
>>>type(list2)
<class 'enumerate'>
```

```
>>>list(list2)                                    #将 enumerate 对象转换成列表
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
```

3.1.4　列表推导式

列表推导式又称列表解析式，是 Python 的一种独有特性。推导式是可以从一个数据序列构建另一个新的数据序列的结构体。使用推导式可以简单高效地处理一个可迭代对象，并生成结果列表。其语法格式为

```
[expr for i1,in 序列 1...for iN in 序列 N]          #迭代序列里所有内容,并计算生成列表
[expr for i1,in 序列 1...for iN in 序列 N if cond_expr]  #按条件迭代,并计算生成列表
```

表达式 expr 使用每次迭代内容计算生成一个列表。如果指定了条件表达式 cond_expr，则只有满足条件的元素参与迭代。例如：

```
>>>[i ** 3 for i in range(5)]                      #range()函数用于创建一个整数列表
[0,1,8,27,64]
>>>[i for i in range(10) if i%2!=0]
[1,3,5,7,9]
>>>[(a,b,a * b) for a in range(1,3) for b in range(1,3) if a>=b]
[(1,1,1),(2,1,2),(2,2,4)]
```

3.2　元　　组

序列数据结构的另一个重要类型是元组，元组元素可以存储不同类型的数据，包括字符串、数字，甚至是元组。元组与列表有两点不同：①元组定义后，它的元素就不能改变；②元组使用小括号，列表使用中括号。

3.2.1　元组的创建

元组的创建非常简单，可以直接用逗号分隔来创建一个元组。例如：

```
>>>tup1="a","b","c","d"
>>>tup1
('a','b','c','d')
```

大多数情况下，元组元素是用圆括号括起来的。例如：

```
>>>tup2=(1,2,3,4)
>>>tup2
(1,2,3,4)
```

注意：即使只创建包含一个元素的元组，也需要在创建时加上逗号分隔符。例如：

```
>>>tup3=(50)                                      #tup3 是整型
>>>tup3
50
>>>tup4=(50, )                                    #tup4 是元组
>>>tup4
```

```
(50, )
```

除了以上两种创建元组的方法外,还可以用 tuple()函数将一个序列作为参数,并将其转换成元组。例如:

```
>>>tup5=tuple([1,2,3,4])
>>>tup5
(1,2,3,4)
>>>tup6=tuple('python')
>>>tup6
('p','y','t','h','o','n')
```

3.2.2 元组的基本操作

元组的操作主要是元组的创建和元组元素的访问,其他操作与列表类似。

1. 元组元素的访问

与列表相似,元组可以直接通过索引来访问元组中的值。例如:

```
>>>tup1=('Google','python','baidu','taobao')
>>>tup1[1]
'python'
>>>tup2=(1,2,3,4,5,6,7,8)
>>>tup2[1:5]
(2,3,4,5)
```

2. 元组元素的排序

与列表不同,元组的内容不能发生改变,因此适用于列表的 sort()方法并不适用于元组。元组的排序只能先将元组通过 list()方法转换成列表,然后对列表进行排序,再将列表通过 tuple()方法转换成元组。例如:

```
>>>te1=('Google','python','baidu','taobao')
>>>te2=list(te1)
>>>te2.sort()
>>>te1=tuple(te2)
>>>te1
('Google','baidu','python','taobao')
```

3. 元组的修改

元组中的元素值是不允许修改的,但可以对元组进行连接组合。例如:

```
>>>tup1=(12,34,56)
>>>tup2=('abc','xyz')
>>>tup3=tup1+tup2
>>>tup3
(12,34,56,'abc','xyz')
>>>tup1[0]=100                                    #修改元组元素操作是非法的
TypeError:'tuple'object does not support item assignments
```