第3章

Python列表

Python列表是一种强大和灵活的数据结构，可以用来存储和处理多个元素。掌握列表的操作和方法是Python编程的基础之一。

3.1 列表的定义

Python列表是一种数据结构，可以包含任意数量的不同数据类型的元素，如数字、字符串、布尔值、列表、函数等。列表使用方括号[]来定义，每个元素之间用逗号分隔。列表支持索引和切片操作，可以对列表元素进行增加、删除、修改和访问等操作。

根据Python官方文档的描述，列表是一种有序的数据集合，用于存储多个元素。列表中的元素可以是不同的数据类型，包括数字、字符串、布尔值等。列表是可变的，可以通过索引进行访问和修改，也可以通过方法进行增加和删除操作。

列表具有有序、可变、可重复等特点。列表可以存储不同类型的元素，并且可以根据需要进行增加、删除、修改和访问操作。掌握列表的基本概念和操作方法，可以有效地处理和操作序列数据。

3.2 列表的基本操作

列表是一种有序的可变序列，允许包含任意类型的元素，并且可以根据需要对列表中的元素进行增加、删除和修改。

Python列表的有序性表现为：列表中的元素按照被添加的顺序排列，即列表中元素的位置是确定的，可以使用索引访问列表中的任何元素。Python列表的可变性表现为：可以更改列表中的元素，列表中的元素个数可以变化。Python列表的可重复性表现为：列表中的元素是可重复的。Python列表的常用操作如表3-1所示。

表3-1 Python列表的常用操作（含部分内置函数与列表方法）

操作符/函数/方法	功 能 描 述	举 例
+	两个列表合并	[1,2]+[3,4]→[1,2,3,4]
*	重复列表元素	[1,2]*3→[1,2,1,2,1,2]
in	判断元素是否在列表中	0 in [1,2]→False
len(seq)	返回对象seq（如列表、字符串、元组、字典、集合等）的元素个数（长度）	len([1,2])→2
list(seq)	将seq转换为列表	list((1,2))→[1,2]

续表

操作符/函数/方法	功 能 描 述	举　　例
all(iterable)	返回一个布尔值。如果 iterable 的元素都为真(或 iterable 自身为空)则返回 True,否则返回 False	all([0,1,2])→False
any(iterable)	返回一个布尔值。如果 iterable 的任一元素为真则返回 True,如果 iterable 的所有元素均为假(或 iterable 自身为空)则返回 False	any([0,1,2])→True
max(iterable)	返回可迭代对象 iterable 中最大的元素	max([0,1,2])→2
min(iterable)	返回可迭代对象 iterable 中最小的元素	min([0,1,2])→0
range(start,stop,step)	返回可序列对象(整数范围的整数序列)。其中,start(可选)的默认值为 0；stop 是整数上限(整数序列的最大值是 stop－1)；step 是步长(整数序列的间隔)	list(range(3))→[0,1,2] list(range(1,3))→ [1,2] ist(range(1,9,2))→[1,3,5,7]
sum(iterable[,start])	返回可迭代对象 iterable 从 start 位置开始向右所有元素的和,start 默认值为 0	sum([1,2,3],1)→7 sum([1,2,3])→6
enumerate(lst)	获取列表中每个元素的索引和值,返回值的类型是元组	numbers ＝[3,6,9] for index, value in enumerate (numbers): 　　print("Index:",index, "Value:",value)
sorted(iterable[,cmp [,key[,reverse＝False]]])	将可迭代对象 iterable 进行排序,返回一个新的列表。可选参数 cmp 是一个带有两个参数的比较函数,它根据第一个参数小于、等于还是大于第二个参数来返回负数、零或正数,默认值为 None。可选参数 key 是一个带有参数的函数,用于从列表元素中选出一个比较关键字,默认值是 None。reverse 若为 True,则列表逆序排序	sorted([2,3,1])→[1,2,3]
list. append(x)	添加一个元素 x 到列表的末尾。相当于 list＝list＋[x]	list1＝[1,2] list1. append(3) list1 的值为[1,2,3]
list. extend(aList)	将列表 aList 中的元素添加到原列表的末尾。相当于 list＝list＋ aList	list1＝[1,2] aList＝[3,4] list1. extend(aList) list1 的值为[1,2,3,4]
list. insert(i,x)	将在列表 list 的第 i 处插入一个元素 x	list1＝[1,2] list1. insert(1,0) list1 的值为[1,0,2]
list. remove(x)	将删除列表中第一个值为 x 的元素。如果没有这样的元素则报错	list1＝[0,1,2,1] list1. remove(1) list1 的值为[0,2,1]
list. pop([i])	将弹出列表中位置为 i 的元素(即从列表中删除该元素并返回)。如果不指定 i,则删除最后一个元素	list1＝[0,1,2,1] list1. pop(1) list1 的值为[0,2,1]
list. index(x,[start], [stop])	在索引值 start 和 stop 指定的范围内,返回列表中第一个值为 x 的元素的索引(下标)。如果没有这样的元素则报错。start 的默认值是 0,stop 的默认值是 len(list)	list＝[0,1,2,1] list. index(1)的值是 1

续表

操作符/函数/方法	功 能 描 述	举 例
list.count(x)	返回列表中 x 出现的次数	list=[0,1,2,1] list.count(1)的值是 2
list.reverse()	反转列表中所有元素的位置	list1=[0,1,2,3] list1.reverse() list1 的值为[3,2,1,0]
list.sort(cmp=None, key=None,reverse=False)	将列表重新排序。参数含义与内置函数 sorted()一致	list1=[0,1,3,2] list1.sort(reverse=True) list1 的值为[3,2,1,0]

以下是一些常见的 Python 列表基本操作的示例。

(1) 创建一个列表。

① 使用[]创建列表。

【例 3-1】 使用[]创建列表的示例。

参考源码如下。

```
#创建一个包含数字的列表
numbers = [1,2,3,4,5]
#创建一个包含字符串的列表
fruits = ["apple","banana","orange","grape"]
#创建一个包含布尔值的列表
flags = [True,False,True,False]
#创建一个包含函数的列表,函数名不要加引号。如果加了引号,则是字符串,不是函数名
funcs = [print,len,sum,max]
#创建一个包含数字、字符串、布尔值和函数名的列表
mixLst = [1,'jxufe',True,print]
```

② 使用内置函数 list()创建列表的示例。

【例 3-2】 使用内置函数 list()创建列表示例。

参考源码如下。

```
aList = list('abc')
print(aList)                    #结果:['a','b','c']
```

(2) 访问列表中的元素。

列表的有序性意味着列表元素的顺序是固定的,即元素在列表中的位置是固定的。列表中的每个元素都有一个与之对应的索引,索引从 0 开始,依次递增。通过索引,可以访问和操作列表中的元素。

例如,myList=[1,2,3,4,5]

在这个列表中,元素 1 的索引值是 0,元素 2 的索引值是 1,以此类推。

element=myList[2] #获取索引值为 2 的元素,即 3。

myList[3]=10 #将索引值为 3 的元素修改为 10。

这些操作都是基于元素在列表中的位置进行的。

【例 3-3】 访问列表中元素的示例。

参考源码如下。

```
numbers = [1,2,3,4,5]
fruits = ["apple","banana","orange","grape"]
#访问列表中的元素
```

```
print(numbers[0])                          #输出结果为 1
print(numbers[3])                          #输出结果为 4
print(fruits[2])                           #输出结果为 "orange"
```

（3）修改列表中的元素。

【例 3-4】 修改列表中的元素示例。

参考源码如下。

```
numbers = [1,2,3,4,5]
#修改列表中的元素
numbers[0] = 10
numbers[3] = 40
```

（4）添加元素到列表末尾。

【例 3-5】 添加元素到列表末尾示例。

参考源码如下。

```
numbers = [10,2,3,40,5]
#添加元素到列表末尾
numbers.append(6)
print(numbers)                             #输出结果为[10,2,3,40,5,6]
```

（5）在列表指定位置插入元素。

【例 3-6】 在列表指定位置插入元素的示例。

参考源码如下。

```
numbers = [10,2,3,40,5,6]
#在指定位置插入元素,列表 numbers 的第二位(索引值为 1)插入新元素
numbers.insert(1,20)
print(numbers)                             #输出结果为 [10,20,2,3,40,5,6]
```

（6）删除列表中的元素。

【例 3-7】 删除列表中的元素示例。

参考源码如下。

```
numbers = [10,20,2,3,40,5,6]
#删除列表中的元素
del numbers[0]
print(numbers)                             #输出结果为[20,2,3,40,5,6]
numbers.remove(40)
print(numbers)                             #输出结果为[20,2,3,5,6]
```

Python 列表提供了切片操作来访问和操作列表中的子序列，即切片操作可以用于从列表中提取一部分元素，或者对列表中的一部分元素进行修改，包括以下 4 种。

（1）基本切片操作。

用来获取列表中的一段连续的子序列。

```
语法：lst[start:end]
```

其中：

```
start 表示起始索引,如果省略 start,则默认为 0。
end 表示结束索引(不包括该索引对应的元素),如果省略 end,则默认为列表的长度。
```

例如，lst[1:3]表示获取列表中从第 2 个元素到第 3 个元素的子序列。

(2) 步长切片操作。

用来获取列表中以指定步长间隔的子序列。

```
语法：lst[start:end:step]
```

其中：

```
start 表示起始索引。
end 表示结束索引(不包括该索引对应的元素)。
step 表示步长。
```

例如，lst[0:6:2]表示获取列表中从第 1 个元素～第 6 个元素以步长为 2 的子序列。

小技巧：可以利用指定步长切片操作，对列表中的元素进行逆序操作。

参考源码如下。

```
lst = [1,2,3,4,5,6,7]
lstRev = lst[::-1]                    #利用步长切片操作,步长为-1,进行逆序操作
print('lst is:{}'.format(lst))
print(f'lstRev is:{lstRev}')
```

执行结果：

```
lst is:[1,2,3,4,5,6,7]
lstRev is:[7,6,5,4,3,2,1]
```

(3) 反向切片操作。

反向切片操作可以用来获取列表中倒数的子序列。

```
语法：lst[-start:-end]
```

其中：

```
start 表示起始索引。
end 表示结束索引(不包括该索引对应的元素)。
```

例如，lst[－3:－1]表示获取列表中倒数第 3 个元素～倒数第 2 个元素的子序列。

(4) 省略切片操作。

省略切片操作可以用来获取列表中的全部元素。

```
语法：lst[:]
```

例如，lst[:]表示获取列表中的全部元素。

注意：切片操作不会修改原始列表，而是返回一个新的列表。同时，在使用切片操作时，需要确保起始索引和结束索引都在列表的范围内，否则会导致 IndexError 异常。

以下是一些常见的 Python 列表切片操作的示例。

【例 3-8】 访问列表中的一部分元素的示例。

参考源码如下。

```
#访问列表中的一部分元素
numbers = [1,2,3,4,5,6,7,8,9,10]
print(numbers[2:6])                    #输出结果为[3,4,5,6]
print(numbers[:5])                     #输出结果为[1,2,3,4,5]
print(numbers[-3:-1])                  #输出结果为[8,9]
```

```
print(numbers[5:])                    #输出结果为[6,7,8,9,10]
print(numbers[:])                     #输出结果为[1,2,3,4,5,6,7,8,9,10]
```

这个例程创建了一个名为 numbers 的列表，其中包含 1～10 的整数。使用切片操作来访问列表中的一部分元素，例如 numbers[2:6]表示从索引 2～索引 5(不包括索引 6)的元素，输出结果为[3,4,5,6]。

通过灵活运用切片操作，可以从大量数据中自主地提取出所选的信息，进行进一步的分析和处理。

使用列表切片时的注意事项：

(1) 索引值应是整数类型的数据。

当索引值为非负整数时，表示从列表的第一个元素开始，向右计数(即正向计算)；当索引值为负整数时，表示从列表的后一个元素开始，向左计数(即反序计算)；当索引值为浮点数时，会报错。例如，列表 lst=[1,2,3,4,5,6]，lst[1.5]会报错。

(2) 使用负数的结束索引。

索引值为－1 时，表示列表倒数第一个元素。lst[－1]可访问列表 lst 的倒数第一个元素 6。但在切片操作中，lst[1:－1]和 lst[－5:－1]的结果相同，均为[2,3,4,5]。

3.3 列表推导式

列表推导式是一种在 Python 中创建列表的简洁方法。它使用现有序列中元素的某些子集来创建新列表。列表推导式可以使用 for 循环(见 7.4.1 节)和 if 语句(见 7.3 节)来筛选元素，也可以使用循环嵌套(见 7.4.4 节)来创建复杂的列表。

以下是列表推导式的一般形式。

```
[expression for item in iterable if condition]
```

其中，

expression 是任何有效的 Python 表达式，它将对每个元素进行求值并将结果存储在新的列表中。

iterable 是任何可迭代对象，例如，列表、元组、字符串等。

condition 是一个对列表元素进行筛选的条件表达式。

列表推导式由方括号[]包围，并使用 for 循环和条件判断组成。for 循环迭代现有序列中的每个元素，并将每个元素传递给 expression 表达式进行计算，把每个计算结果作为新列表中的一个元素。

(1) 省略 if 语句的列表推导式，生成一个列表。

【例 3-9】 利用列表推导式创建一个包含前 10 个整数的列表的示例。

参考源码如下。

```
numbers = [x for x in range(10)]
print(numbers) #[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

在本例中，使用 for 循环迭代 range(10)中的每个元素，并将每个元素存储在 numbers 列表中。

(2) 使用 if 语句的列表推导式，生成一个列表。

【例 3-10】 创建一个包含所有偶数的列表示例。

参考源码如下。

```
evenNumbers = [x for x in range(10) if x % 2 == 0]
print(evenNumbers)
```

执行结果：

```
[0,2,4,6,8]
```

在本例中，使用 if 语句来过滤 range(10)中的所有奇数，并将所有偶数存储在 evenNumbers 列表中。

(3) 使用循环嵌套的列表推导式，生成一个列表。

【例 3-11】 创建一个包含三位数的列表的示例。

参考源码如下。

```
itNumbers = [x * 100 + y * 10 + z for x in range(1,2) for y in range(3) for z in range(4)]
print(itNumbers)
```

执行结果：

```
[0,1,2,3,10,11,12,13,20,21,22,23,100,101,102,103,110,111,112,113,120,121, 122,123]
```

在本例中，使用三个嵌套循环来创建一个包含所有三位数的列表。第一个循环迭代 range(1,2)，第二个循环迭代 range(3)，第三个循环迭代 range(4)。for 循环的结果被传递给表达式，该表达式将对每个元素进行求值并将结果存储在新的列表中。

【例 3-12】 使用列表生成式来创建一个 3×4 的矩阵的示例。

参考源码如下。

```
testArray = [[x + y for x in range(3)] for y in range(4)]
print(testArray)
```

执行结果：

```
[[0,1,2],[1,2,3],[2,3,4],[3,4,5]]
```

在金融领域，列表生成式有很多应用，可以用来创建股票价格数据集、计算收益率、计算波动率等。

以下是三个列表推导式在金融场景下的应用。

【例 3-13】 使用列表推导式选取符合条件的股票名称的示例。

参考源码如下。

```
stocks = [{'stockName':'name1','marketCap':500},{'stockName':'name2','marketCap':300},{'stockName':
'name3','marketCap':360},{'stockName': 'name4','marketCap':550}]
selectedStocks = [stock['stockName'] for stock in stocks if stock['marketCap']> 360]
print(selectedStocks)
```

运行结果：

```
['name1','name4']
```

【例 3-14】 使用列表推导式计算股票收益率的示例。

$$收益率=(当天的收盘价/昨天的收盘价)-1$$

假设有一个包含每天股票价格的列表 stockPrices，可以使用列表生成式计算股票的收益

率列表。

参考源码如下。

```
stockPrices = [100.01,99.50,99.80,100.02,100.5]
returns = [(price - stockPrices[i - 1]) / stockPrices[i - 1] for i, price in enumerate
(stockPrices) if i > 0]
print(returns)
```

运行结果：

```
[ - 0.005099490050994951,0.0030150753768843934, 0.002204408817635259,0.004799040191961648]
```

注：内置函数 enumerate()的功能，见 8.5 节中的表 8-1。

【例 3-15】 使用列表推导式按股票价格排序的示例。

参考源码如下。

```
stockPrices = {'name1':150,'name2':1100,'name3': 300, 'name4':2500}
sortedPrices = [code for code, price in sorted(stockPrices.items(), key = lambda x: x[1])]
# lambda x: x[1]lambda 函数，详见 8.6 节
print(sortedPrices)
```

运行结果：

```
['name1', 'name3', 'name2', 'name4']
```

注：内置函数 sorted()的功能，见 8.5 节中的表 8-1。lambda 函数，详见 8.6 节。

列表推导式通常用于处理大量数据或筛选数据等场景，但它也有一些注意事项需要注意。

(1) 在列表推导式中，列表元素的类型由列表推导式中的表达式的计算结果类型决定。例如，下面的代码会创建一个包含字符串"a"到"z"的列表。因为表达式 chr(i)的结果类型是字符串，所以列表元素的类型也是字符串。

```
letters = [chr(i) for i in range(97, 123)]
```

(2) 列表推导式可以包含多个表达式和条件语句，可以使用类似于 if…else 的语法进行条件筛选。但是列表生成器中的代码不能包含赋值语句或 yield 语句(详见 8.8 节)。

(3) 与生成器不同，列表推导式返回的是一个列表对象。如果需要将其转换为其他数据类型，可以使用内置函数 tuple()、set()等。

(4) 当列表推导式中的可迭代对象为空时，会返回一个空列表。

列表推导式是一种非常强大的工具，但它也可能会降低程序的性能。如果列表推导式中的表达式很复杂，那么列表推导式可能会导致程序运行速度变慢。

3.4 列表在金融领域的应用

列表在金融场景下有很多应用场景，如创建股票价格列表、创建交易量列表、创建收益率列表、创建波动率列表、创建技术指标列表、创建回测结果列表、创建交易策略列表等。

(1) 股票价格列表。Python 的列表类型可以用来表示一组股票的价格数据，如 stockPrices=[10.5,12.3,11.7,13.2]，这样可以方便地对股票价格进行存储和处理。例如，计算平均价格、最高价格、最低价格等。

(2) 财务报表数据。Python 的列表类型可以用来表示财务报表的数据，如 financialData=[1000,1200,1100,1300]，表示每个季度的收入。例如，计算总收入、收入变化等。

(3) 交易记录列表。Python 的列表类型可以用来表示一组交易记录的数据，如 tradeRecords=["2021-08-01 购买股票 A500 股","2021-08-02 卖出股票 B600 股"]，通过访问 tradeRecords 中的数据，可以进行各种分析处理，如筛选特定日期的交易记录、统计交易数量等。

(4) 客户信息列表。Python 的列表类型可以用来表示一组客户信息的数据，如 customerInfo=["张三","李四","王五"]，通过编程可以对客户信息进行存储和处理，例如，查找特定客户、统计客户数量等。

(5) 指标数据集合。Python 的列表类型可以用来表示一组指标数据，如 indicators=[0.1,0.2,0.15,0.3]，通过对 indicators 的访问，可以进行各种处理，如计算平均值、标准差、相关性分析等。

例如：

```
#创建一个包含股票价格的列表
stockPrices = [10.5,11.2,12.1,10.8,11.5,12.2,13.1,14.5,15.2,16.1]
```

创建了一个名为 stockPrices 的列表，其中包含 10 支股票价格。可以使用索引来访问列表中的每个元素，例如：

```
#访问列表中的元素
print(stockPrices[0])                    #输出结果为 10.5
print(stockPrices[5])                    #输出结果为 12.2
print(stockPrices[-1])                   #输出结果为 16.1
```

除了访问列表中的元素之外，还可以使用各种内置函数来处理列表。例如，可以使用 sum()函数来计算列表中所有元素的总和，使用 len()函数来计算列表中元素的数量，使用 sort()函数来按升序或降序排序列表中的元素等。

```
#计算列表中所有元素的总和
total = sum(stockPrices)
print(total)

#计算列表中元素的数量
count = len(stockPrices)
print(count)

#按升序排序列表中的元素
stockPricesSort()
print(stockPrices)

#按降序排序列表中的元素
stockPrices.sort(reverse=True)
print(stockPrices)
```

以下是 5 个典型的列表及操作在有关金融方面应用的示例。

【例 3-16】 假设有一支股票价格的列表，使用 Python 的内置函数和列表操作来进行统计分析的示例。

参考源码如下。

```
#股票价格列表
stockPrices = [10.2,11.5,12.3,11.8,10.5,9.8,10.1,11.2,12.5,14.2]
#计算平均价格
averagePrice = sum(stockPrices) / len(stockPrices)
```

```
print("平均价格：", averagePrice)
#计算最高价格
maxPrice = max(stockPrices)
print("最高价格：", maxPrice)
#计算最低价格
minPrice = min(stockPrices)
print("最低价格：",minPrice)
#获取最近5天的价格
recentPrices = stockPrices[-5:]
print("最近5天的价格：",recentPrices)
#获取涨幅超过10%的日期
riseDates = [i for i in range(len(stockPrices)) if stockPrices[i]/ stockPrices[i-1]>1.1]
print("涨幅超过10%的日期：", riseDates)
```

执行结果：

```
平均价格：11.41
最高价格：14.2
最低价格：9.8
最近5天的价格：[9.8,10.1,11.2,12.5,14.2]
涨幅超过10%的日期：[1,7,8,9]
```

【例 3-17】 对财务报表进行计算总收入、净利润率的示例。

净利润率的计算公式：

$$\text{净利润率} = \text{净收入} / \text{总收入}$$

参考源码如下。

```
#定义财务报表,其中的列表元素是字典(详见第5章)
financialStatements = [
    {'year':2018,'revenue':1000000,'cost':800000,'netIncome':200000},
    {'year':2019,'revenue':1200000,'cost':900000,'netIncome':300000},
    {'year':2020,'revenue':1500000,'cost':1000000,'netIncome':400000}
]

#计算总收入和净利润率
totalRevenue = sum([fs['revenue'] for fs in financialStatements])
totalCost = sum([fs['cost'] for fs in financialStatements])
totalNetIncome = sum([fs['netIncome'] for fs in financialStatements])
profitMargin = totalNetIncome / totalRevenue
#输出结果
print("财务报表分析：")
print("总收入：{}元".format(totalRevenue))
print("净利润率：{:.2%}".format(profitMargin))
```

执行结果：

```
财务报表分析：
总收入：3700000元
净利润率：24.32%
```

在本例中，定义了一个包含多个财务报表的列表 financialStatements，列表元素（财务报表）是一个字典（字典的概念和操作见第5章），每个财务报表包含年份、收入、成本和净利润等信息。使用列表推导式和 sum()函数，计算所有财务报表的总收入、总成本和总净利润。将总净利润除以总收入，得到净利润率。最后，使用 print()函数输出结果。

【例 3-18】 假设有一支股票交易记录的列表，使用 Python 的列表推导式和条件判断来

筛选出符合条件的交易记录的示例。

参考源码如下。

```
#股票交易记录列表
tradingRecords = [
   {'date':'2021-01-01','amount':100,'price':10.2,'type':'buy'},
   {'date':'2021-01-05','amount':200,'price': 11.5,'type':'buy'},
   {'date':'2021-01-10','amount':300,'price': 12.3,'type':'sell'},
   {'date':'2021-01-15','amount':400,'price': 11.8,'type':'buy'},
   {'date':'2021-01-20','amount':500,'price': 10.5,'type':'sell'},
   {'date':'2021-01-25','amount':600,'price': 9.8,'type':'sell'}
]
#筛选出买入交易记录
buyRecords = [record for record in tradingRecords if record['type'] == 'buy']
print("买入交易记录: ", buyRecords)
#筛选出卖出交易记录
sellRecords = [record for record in tradingRecords if record['type'] == 'sell']
print("卖出交易记录: ", sellRecords)
#筛选出交易金额大于1000的交易记录
highAmountRecords = [record for record in tradingRecords if record['amount'] * record['price'] >
1000]
print("交易金额大于1000的交易记录: ", highAmountRecords)
```

执行结果:

```
买入交易记录: [{'date': '2021-01-01', 'amount': 100, 'price': 10.2, 'type': 'buy'}, {'date': '2021-
01-05', 'amount': 200, 'price': 11.5, 'type': 'buy'}, {'date': '2021-01-15', 'amount': 400, 'price':
11.8, 'type': 'buy'}]
卖出交易记录: [{'date':'2021-01-10','amount':300,'price':12.3,'type': 'sell'},{'date':'2021-
01-20','amount':500,'price':10.5,'type':'sell'},{'date':'2021-01-25','amount':600,'price':9.8,
'type':'sell'}]
交易金额大于1000的交易记录: [{'date':'2021-01-01','amount':100,'price':10.2, 'type':'buy'},
{'date':'2021-01-05','amount':200,'price':11.5,'type':'buy'},{'date':'2021-01-10','amount':
300,'price':12.3,'type':'sell'},{'date':'2021-01-15','amount':400,'price':11.8,'type':'buy'},
{'date':'2021-01-20','amount':500,'price':10.5,'type':'sell'},{'date':'2021-01-25','amount':600,
'price':9.8,'type':'sell'}]
```

在本例中,定义了一个包含交易记录列表tradingRecords,列表元素(交易记录)是一个字典(字典的概念和操作见第5章),每个记录包含交易时间、交易数量、交易方向等信息。使用列表推导式和条件判断,筛选并分别生成了三个列表:买入交易记录、卖出交易记录和交易金额大于1000的交易记录。最后,使用print()函数输出结果。

【例3-19】 利用列表计算平均值、标准差等的示例。

参考源码如下。

```
#定义指标数据
data = [10,12,8,14,15,11,9,13,16,12]
#计算平均值
average = sum(data) / len(data)
#计算标准差
variance = sum([(x - average) ** 2 for x in data]) / len(data)
std_deviation = variance ** 0.5
#输出结果
print("指标数据分析: ")
print("平均值: {:.2f}".format(average))
print("标准差: {:.2f}".format(std_deviation))
```

执行结果:

```
指标数据分析:
平均值: 12.00
标准差: 2.45
```

在本例中，定义了一个包含多个指标数据的列表 data，例如，公司的每月销售额或每日股票收盘价等。使用 sum()函数和 len()函数，计算所有指标数据的平均值。使用列表推导式和 sum()函数，计算所有指标数据的方差，并将方差开方得到标准差。使用 print()函数输出结果。

【例 3-20】 查找特定客户、统计客户数量的示例。

参考源码如下。

```
#定义客户列表
customers = [{'name':'张三','age':30,'gender':'男','email':'zhangsan@example.com'},{'name':'李
四','age':25,'gender':'女','email':'lisi@example.com'},{'name':'王五','age':40,'gender':'男', 'email':'wangwu
@example.com'},{'name':'赵六','age':35,'gender':'女','email':'zhaoliu@example.com'}]
#查找特定客户
targetCustomer = '张三'
for customer in customers:
    if customer['name'] == targetCustomer:
       print("找到客户: ", customer)
       break
else:
   print("未找到客户: ",targetCustomer)
#统计客户数量
numCustomers = len(customers)
print(f"客户数量: {numCustomers}")
```

执行结果：

```
找到客户: {'name':'张三','age':30,'gender':'男','email': 'zhangsan@example.com'}
客户数量: 4
```

在本例中，定义了一个包含多个客户信息的列表 customers，每个客户包含姓名、年龄、性别和电子邮件等信息。使用 for 语句遍历所有客户信息，查找特定的客户。如果找到了目标客户，则输出该客户的信息；否则输出“未找到客户”和目标客户的姓名。使用 len()函数统计客户数量，得到客户数量并输出。

这些例程说明了 Python 列表在金融方面的应用，通过灵活运用列表操作和条件判断，可以进行各种复杂的数据处理和分析。使用 Python 列表时需要注意以下 4 点。

(1) 列表是可变的数据类型，列表中的元素可以是任何数据类型，包括其他列表。可以随时添加、删除或修改其中的元素。修改列表中的元素可能会影响到其他使用该列表的代码。

(2) 列表是有序的数据类型，可以使用索引访问列表元素。列表的索引从 0 开始，可以使用负数索引反序访问列表的元素，也可以使用循环语句来迭代访问列表中的每个元素。

(3) 列表使用切片操作来获取列表的子集。可以使用 list[start:end]来获取从索引 start 到 end-1 的元素。列表的长度可以使用 len()函数获取。

(4) 列表的方法包括添加、删除、排序、反转等操作，需要根据具体需求选择合适的方法。列表可以使用内置函数进行操作：Python 提供了许多内置函数来操作列表，例如，len()用于获取列表的元素个数，append()用于在列表末尾添加元素，remove()用于删除指定元素等。