




# Python编程案例教程

# 第5章 列表和元组

## 本章导读



在计算机应用领域中，常常会遇到需要处理大量数据的问题，其特点是：数据量很大，数据之间存在一定的内在关系。例如，一个班有100个学生，如果用字符串变量存储学生的名字，那么需要100个变量，可见用字符串变量来处理这样的数据将十分不方便，甚至不能处理。本章将介绍如何使用列表和元组数据结构解决这一类问题。





# 学习目标

- 理解列表的概念
  - 掌握列表的常见操作
  - 掌握元组的使用方法
  - 理解列表和元组的区别
- 

The background of the slide features an abstract design on the left side, composed of several overlapping, semi-transparent red polygons in various shades of red, creating a dynamic, layered effect. The right side of the slide is a solid white background.

**5.1 列 表**

5.2 元 组

5.3 典型案例

# 5.1 列 表

◆ 5.1.1 列表的概述

◆ 5.1.2 列表的遍历

◆ 5.1.3 列表的基本操作

### 5.1.1 列表的概述

列表（list）是包含0个或多个元素的有序序列，属于序列类型。

- ◆列表的长度和内容都是**可变**的，可自由对列表中的元素进行增加、删除或替换。
- ◆列表没有长度限制，**元素类型可以不同**，可以同时包含整数、实数、字符串等基本类型，也可以是列表、元组、字典、集合以及其他自定义类型的对象，使用非常灵活。

## 5.1.1 列表的概述

### 1、创建

创建方式一：

创建列表的方式很简单，只需将逗号分隔的不同元素使用方括号括起来即可。

例如：

```
>>> animal = ['elephant', 'monkey', 'snake', 'tiger']
```

```
>>> animal
```

```
['elephant', 'monkey', 'snake', 'tiger']
```

### 5.1.1 列表的概述

创建方式二:

使用**list()**将字符串、range对象，元组等对象转换为列表。

例如：

```
>>>list('hello world')
```

#将字符串转换为列表

```
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
```

```
>>>list(range(1,10,2))
```

#将range对象转换为列表

```
[1, 3, 5, 7, 9]
```



### 5.1.1 列表的概述

创建方式三：

使用乘法创建。

例如：

```
>>> a=[1]*10
```

#创建一个初始值为10个1元素的列表

### 5.1.1 列表的概述

创建方式四:

创建空列表。

例如 :

```
>>>a=[]
```

```
#创建一个初始值0个元素的列表
```

```
r=[]  
r.append("hello,")  
r.append("world!")
```

### 5.1.1 列表的概述

## 2、列表元素访问

与字符串的索引一样，列表索引也是从0开始的。我们可以通过下标索引的方式来访问列表中的值。

例如：

```
>>> animal = ['elephant', 'monkey', 'snake', 'tiger']
```

```
>>> animal[0]
```

```
'elephant'
```

## 5.1.1 列表的概述

### 3、列表属于引用型序列

例：列表的赋值与引用。

```
stu_1 = ['001', 'Wangwu', 98]    #使用数据赋值创建列表stu_1
stu_2 = stu_1    #stu_2是stu_1所对应数据的应用，stu_2并不包含真实数据
print(stu_1, stu_2)              #输出stu_1和stu_2
stu_1[0] = '002'                 #修改元素stu_1[0]的值为'002'
print(stu_1, stu_2)              #输出stu_1和stu_2
```

```
['001', 'Wangwu', 98] ['001', 'Wangwu', 98]
['002', 'Wangwu', 98] ['002', 'Wangwu', 98]
[Finished in 0.1s]
```



## 5.1.1 列表的概述

### 3、列表属于引用型序列

#### 提示

将一个变量赋值给另外一个变量都属于传址型；

将一个数值赋值给变量，系统一般都以重新分配地址创建新变量来处理。（特例是整数池0——256数据，及同值字符串变量赋值。）

## 5.1.2 列表的遍历

### ► 1 . 使用while循环遍历列表

使用while循环遍历列表，首先使用len( )需要获取列表的长度，将获取的列表长度作为while循环的判断条件。

## 5.1.2 列表的遍历

### ► 1 . 使用while循环遍历列表

例：使用while循环遍历列表。

```
animal = ['elephant', 'monkey', 'snake', 'tiger']
```

```
length = len(animal)
```

```
i = 0
```

```
while i < length:
```

```
    print(animal[i])
```

```
    i += 1
```

```
elephant  
monkey  
snake  
tiger  
[Finished in 0.2s]
```

## 5.1.2 列表的遍历

### ► 2 . 使用for循环遍历列表

使用for循环遍历列表的方式非常简单，只需要将要遍历的列表作为for循环表达式中的序列即可。

例：使用for循环遍历列表。

```
animal = ['elephant', 'monkey', 'snake', 'tiger']  
for name in animal:  
    print(name)
```

```
elephant  
monkey  
snake  
tiger  
[Finished in 0.2s]
```

3 lines, 148 characters selected



### 5.1.3 列表的基本操作

#### ► 1 . 增加元素

1) **append()方法**：使用append()方法可在列表末尾添加新的元素。

语法格式：

**list.append(obj)**

◆ list表示列表

◆ obj表示添加到列表末尾的对象

### 5.1.3 列表的基本操作

#### ► 1 . 增加元素

例如：

```
>>>add_list = [0,1,2,3]
```

```
>>>add_list.append(4)
```

```
>>>add_list
```

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

### 5.1.3 列表的基本操作

- 2) **extend()方法**：使用extend()方法可在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）。

语法格式：

**list.extend(seq)**

◆ list表示列表

◆ seq表示添加到列表中的元素列表

### 5.1.3 列表的基本操作

- 2) **extend()方法**：使用extend()方法可在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）。

例：比较extend()方法与append()方法添加列表元素的不同。

```
list_1 = [1,2,3]      #创建列表list_1并赋值
```

```
list_2 = [4,5,6]      #创建列表list_2并赋值
```

```
list_3 = [7,8,9]      #创建列表list_3并赋值
```

```
list_1.append(list_2)  #将list_2看作一个对象，整体打包添加到list_1对象中
```

```
print(list_1) #输出list_1
```

```
list_2.extend(list_3)  #将list_3看作一个序列，将这个序列和list_2序列合并
```

```
print(list_2) #输出list_2
```

```
[1, 2, 3, [4, 5, 6]]  
[4, 5, 6, 7, 8, 9]  
[Finished in 0.2s]
```

### 5.1.3 列表的基本操作

3) **insert()方法**：使用insert()方法可将指定对象插入到列表的指定位置。

语法格式：

`list.insert(index,obj)`

- ◆ list表示列表
- ◆ index表示对象需要插入的索引位置
- ◆ obj表示要插入列表中的对象

### 5.1.3 列表的基本操作

3) **insert()方法**：使用insert()方法可将指定对象插入到列表的指定位置。

例如：

```
>>> number = [1,2,4,5] #创建列表number并赋值
```

```
>>> number.insert(2,3) #使用insert()方法将元素3添加到列表索引为2的位置中
```

```
>>> number #打印number
```

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

### 5.1.3 列表的基本操作

#### ► 2 . 查找元素与计数

- 1) `index()`方法：用于返回指定元素在列表中首次出现的位置，如果该元素不在列表中则抛出异常。

语法格式：

`list.index(obj)`

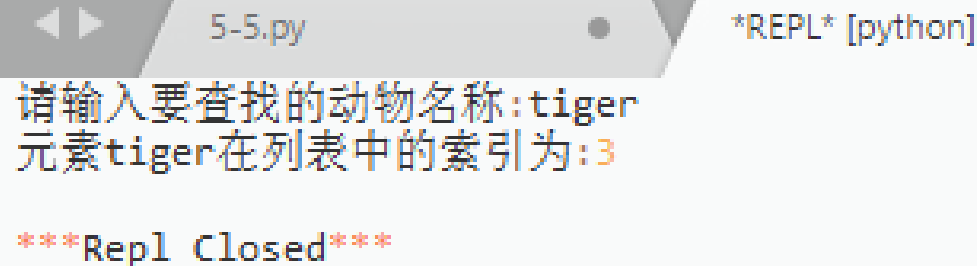
- ◆ list表示列表
- ◆ obj表示要查找的对象

### 5.1.3 列表的基本操作

#### ► 2. 查找元素与计数

例：在列表中查找元素，如果找到，输出该元素在列表中的索引位置，否则输出未找到。

```
animal = ['elephant', 'monkey', 'snake', 'tiger']           #创建列表animal
x = input('请输入要查找的动物名称:')                      #输入要查找的元素
if x in animal:                                             #查找是否存在
    a = animal.index(x)                                     #返回索引
    print('元素{0}在列表中的索引为:{1}'.format(x,a))      #输出索引号
else:
    print('列表中不存在该元素')
```



```
5-5.py *REPL* [python]
请输入要查找的动物名称:tiger
元素tiger在列表中的索引为:3
***Repl Closed***
```



### 5.1.3 列表的基本操作

2) `count()`方法：统计指定元素在列表中出现的次数。

语法格式：

`list.count(obj)`

◆ `list`表示列表

◆ `obj`表示列表中要统计的对象

### 5.1.3 列表的基本操作

2) `count()`方法：统计指定元素在列表中出现的次数。

例如：

```
>>>x = [1,2,1,2,1,2] #创建列表
```

```
>>>x.count(1) #使用count()方法统计1在变量x中的出现次数
```

```
3
```

### 5.1.3 列表的基本操作

#### ▶ 3 . 删除元素

- 1) **del命令**：可根据索引删除列表中的元素，还可以使用分片的方式删除列表中的元素。

### 5.1.3 列表的基本操作

#### ► 3. 删除元素

例如：

```
>>> number = [1,2,3,4,5] #创建列表number并赋值
```

```
>>> del number[2] #使用del命令删除number中索引为2的元素
```

```
>>> number #打印number
```

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

```
>>> number = [1,2,3,4,5] #创建列表number并赋值
```

```
>>> del number[1:3] #使用del命令删除索引从1到3（不包括3）的元素
```

```
>>> number #打印number
```

```
[1, 4, 5]
```

### 5.1.3 列表的基本操作

- 2) **pop()方法**：用于移除列表中的一个元素（默认为最后一个元素），并且返回该元素的值。

语法规则：

`list.pop([obj])`

◆ list表示列表

◆ obj为可选参数，表示移除列表元素的索引值，默认为-1，删除最后一个列表值

### 5.1.3 列表的基本操作

2) **pop()方法**：用于移除列表中的一个元素（默认为最后一个元素），并且返回该元素的值。

例如：

```
>>> number = [1,2,3,4,5] #创建列表number并赋值
```

```
>>> number.pop()          #使用pop()方法删除number中的元素5
```

```
>>> number                #打印number    结果[1, 2, 3, 4]
```

```
>>> number.pop(0)         #使用pop()方法删除number中索引为0的元素 结果1
```

```
>>> number                #打印number    结果[2, 3, 4]
```

### 5.1.3 列表的基本操作

3) `remove()`方法：用于移除列表中某个值的第一个匹配项。

语法规则：

`list.remove(obj)`

◆ `list`表示列表

◆ `obj`表示列表中要移除的对象

### 5.1.3 列表的基本操作

3) `remove()`方法：用于移除列表中某个值的第一个匹配项。

例：删除列表中所有的指定元素。

```
x = ['123','abc','xyz','abc','python']    #创建列表x并赋值
while 'abc' in x:                          #循环
    x.remove('abc')                        #使用remove()方法删除x中值为'abc'的元素
print(x)                                  #输出x
```

```
['123', 'xyz', 'python']
[Finished in 0.2s]
```



### 5.1.3 列表的基本操作

#### ► 4 . 分片赋值

- ◆ 分片赋值就是以分片的形式给序列赋值，可以一次为多个元素赋值。
- ◆ 分片赋值还可以在不替换任何原有元素的情况下插入新的元素。
- ◆ 分片赋值也可以用来删除元素。

### 5.1.3 列表的基本操作

#### ► 4 . 分片赋值

例如：

```
>>>x = [1, 2, 3, 4]    #创建列表并赋值
```

```
>>>x                  #打印列表x
```

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

```
>>>x[2:] = [5, 6, 7]#替换变量x索引从2开始到最后的元素
```

```
>>>x                  #打印变量x
```

```
[1, 2, 5, 6, 7]
```

### 5.1.3 列表的基本操作

#### ► 4 . 分片赋值

例如：

```
>>>x = [1,7]           #创建列表并赋值
```

```
>>>x[1:1] = [2,3,4,5,6]   #在变量x中索引为1的位置插入列表[2,3,4,5,6]
```

```
>>>x                     #打印变量x
```

```
[1, 2, 3, 4, 5, 6, 7]
```

### 5.1.3 列表的基本操作

#### ► 4 . 分片赋值

例如：

```
>>>x = [1, 2, 3, 4, 5, 6, 7] #创建列表并赋值
```

```
>>>x[1:6] = [] #用空列表将x中索引1到索引6之间的元素替换
```

```
>>> x #打印变量x
```

```
[1, 7]
```

### 5.1.3 列表的基本操作

#### ► 4 . 分片赋值

例如：补充

```
>>>x = [1, 2, 3, 4, 5, 6, 7] #创建列表并赋值
```

```
>>>y=x[:] #使用切片实现克隆
```

```
>>> y          #打印变量y
```

```
[1, 2, 3, 4, 5, 6, 7]
```

注意：切片产生的列表为新列表

### 5.1.3 列表的基本操作

#### ► 5. 列表元素转置

`reverse()`方法：用于将列表中的元素反向存放。  
语法格式：

`list.reverse()`

◆ `list`表示列表，该方法没有参数，没有返回值

### 5.1.3 列表的基本操作

#### ► 5 . 列表元素转置

例如：

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

```
#创建列表并赋值
```

```
>>>x.reverse()
```

```
#使用reverse()方法将变量x中的元素反向存放
```

```
>>>x
```

```
#打印变量x
```

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

### 5.1.3 列表的基本操作

## ► 6 . 列表元素排序

- 1) **sort()方法**：用于对原列表进行排序（默认为升序排序），排序后的新列表会覆盖原列表。

语法规则：

**list.sort([key=None],[reverse=False])**

- ◆ list表示列表
- ◆ key为可选参数，如果指定了该参数，会使用该方法进行排序
- ◆ reverse为可选参数，表示是否反向排序，默认为False



### 5.1.3 列表的基本操作

#### ► 6 . 列表元素排序

- 1) **sort()方法**：用于对原列表进行排序（默认为升序排序），排序后的新列表会覆盖原列表。

例：给定任意n个整数，按由小到大次序对其进行排序，并输出结果。

`x = [3,2,1,5,4]`                      #创建列表并赋值

`x.sort()`                      #使用sort()方法对变量x中的元素进行升序排序

`print(x)`                      #输出列表x

```
[1, 2, 3, 4, 5]  
[Finished in 0.3s]
```

### 5.1.3 列表的基本操作

例：将多个字符串按照长度逆序排序后输出。

```
x = ['a','abc','ab','abcd']    #创建列表并赋值
x.sort(key=len,reverse=True)    #使用sort()方法对变量x中
元素进行长度逆序排序
print(x)                        #输出列表x
```

```
['abcd', 'abc', 'ab', 'a']
[Finished in 0.2s]
```

### 5.1.3 列表的基本操作

例：将多个字符串按照不区分大小写排序后输出。

```
x = ['a','Ac','ab','Gbcd']      #创建列表并赋值
x.sort(key=str.lower,reverse=True)  #使用sort()方法对变
量x中元素进行不区分大小写排序
print(x)                          #输出列表x
```

### 5.1.3 列表的基本操作

- 2) `sorted()`函数：与`sort()`方法不同，内置函数`sorted()`返回新列表，并不对原列表进行任何修改。

语法规则：

`sorted(iterable[,key=None][,reverse=False])`

- ◆ `iterable`表示可迭代对象，在这里就是列表名
- ◆ 参数`key`和`reverse`的用法与`sort()`方法中的相同

### 5.1.3 列表的基本操作

2) `sorted()`函数：与`sort()`方法不同，内置函数`sorted()`返回新列表，并不对原列表进行任何修改。

例如：

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

```
#创建列表并赋值
```

```
>>>y=sorted(x)
```

```
#将x中的元素升序排序后赋给变量y
```

```
>>>x
```

```
#输出x    [1, 5, 2, 3, 4]
```

```
>>>y
```

```
#输出y    [1, 2, 3, 4, 5]
```

### 5.1.3 列表的基本操作

#### 2) 元素过滤：

例如：

```
li = ["a", "mpilgrim", "foo", "b", "c", "b", "d", "d"]  
newli=[elem for elem in li if len(elem) > 1]  
print(newli)
```

### 5.1.3 列表的基本操作

例如：通过itemgetter对元组列表排序

```
from operator import itemgetter  
student_tuples=[('john', 'A', 15), ('dave', 'B1', 10), ('jane', 'B2', 10)]  
sorted(student_tuples, key=itemgetter(2,1))
```

## 5.2 元 组

◆ 5.2.1 元组的基本操作

◆ 5.2.2 元组的内置函数



元组 ( tuple ) : 与列表类似, 不同之处在于不能修改元组的元素, 元组使用圆括号包含元素, 而列表使用方括号包含元素。

创建元组 : 只需要在圆括号中添加元素, 并使用逗号分隔元素即可, 也可省略圆括号, 直接使用逗号分隔元素。

例如：

```
>>>x=('a', 'b', 1, 2,3)    #创建元组x
```

```
>>>x                      #输出x
```

```
('a', 'b', 1, 2,3)
```

```
>>>y='a','b','c','d'      #创建元组y时省略了圆括号
```

```
>>>y                      #输出y
```

```
('a','b','c','d')
```

例如：

提示

(1) 如果要定义一个空元组，可以用没有包含内容的圆括号来表示。

(2) Python在定义只有1个元素的元组时，在元素后面必须加一个逗号“，”。

```
>>>x = () #创建空元组x
```

```
>>>x = (1,) #创建只有一个元素的元组x
```

## 5.2.1 元组的基本操作

### ► 1. 访问元组

与列表类似，可以使用下标索引来访问元组中的值。



例如：

```
>>>x = ('a',1,3.14)
```

```
#创建元组x
```

```
>>>x[0]
```

```
#输出元组索引为0的元素
```

```
'a'
```

```
>>>x[1]
```

```
#输出元组索引为1的元素
```

```
1
```

```
>>>x[2]
```

```
#输出元组索引为2的元素
```

```
3.14
```

## 5.2.1 元组的基本操作

### ► 2 . 合并元组

元组中的元素值是不允许修改的，但我们可以对元组进行连接组合。



例如：

```
>>>x = (1, 2, 3)
```

```
#创建元组x
```

```
>>>y = ('a', 'b')
```

```
#创建元组y
```

```
>>>z = x + y
```

```
#将元组x和y进行连接组合后赋给z
```

```
>>>z
```

```
#输出z
```

```
(1, 2, 3, 'a', 'b')
```

## 5.2.1 元组的基本操作

### ► 3 . 元组的遍历

与列表一样，也可以使用for循环来遍历元组中的所有元素。

例：元组的遍历。

```
x=(1, 2, 3, 4, 5)                                #创建元组x
```

```
for n in x:                                       #循环
```

```
    print(n,' ',end="")                          #输出元素
```

```
1 2 3 4 5 [Finished in 0.2s]
```

## 5.2.2 元组的内置函数

Python提供的元组内置函数有len()、max()、min()和tuple()。

tuple()函数的功能是以一个序列为参数，并把它转换为元组，如果参数本身是元组，则原样返回该参数。

例如：

```
>>> tuple([1,2,3])           #将列表转换为元组
```

```
(1, 2, 3)
```

```
>>> tuple('abc')             #将字符串转换为元组
```

```
('a', 'b', 'c')
```

```
>>> tuple((1,2,3))           #参数为元组
```

```
(1, 2, 3)
```

The background features a complex, abstract design of overlapping red and pink triangles and polygons, creating a dynamic, low-poly aesthetic. The colors range from deep red to light pink, with some areas appearing more translucent than others.

## 5.3 典型案例



## 5.3.1 简易计算器

例：编写一个简易计算器，要求根据输入的数字和四则运算符号，计算运算结果并输出。

```
operator_list = ['+', '-', '*', '/']          #创建列表并赋值四则运算符
number_1 = float(input("请输入第一个操作数：")) #获取第一个操作数
operator = input("请输入运算符：")           #获取运算符
number_2 = float(input("请输入第二个操作数：")) #获取第二个操作数
#判断用户输入的运算符并进行计算，将结果赋值给变量result
if operator not in operator_list:             #输入的运算符不是四则运算符
    print("输入的运算符有误，请输入四则运算符！") #输出提示语
else:                                         #输入的运算符属于四则运算符
    if operator == '+':                      #运算符为 "+"
        result = number_1 + number_2        #两数相加的值赋给变量result
    elif operator == '-':                    #运算符为 "-"
        result = number_1 - number_2        #两数相减的值赋给变量result
    elif operator == '*':                    #运算符为 "*"
        result = number_1 * number_2        #两数相乘的值赋给变量result
    elif operator == '/':                    #运算符为 "/"
        result = number_1 / number_2        #两数相除的值赋给变量result
    print(number_1, operator, number_2, "=", result) #将两个操作数的运算结果输出
```

### 程序运行效果



## 5.3.2 矩阵相加

例：已知两个3行3列的矩阵，实现其对应位置的数据相加，并返回一个新矩阵。

```
X = [[12,7,3],[4,5,6],[7,8,9]]           #定义矩阵X
Y = [[5,8,1],[6,7,3],[4,5,9]]           #定义矩阵Y
result = [[0,0,0],[0,0,0],[0,0,0]]       #定义新矩阵
for i in range(len(X)):                  #循环控制行
    for j in range(len(X[0])):           #循环控制列
        result[i][j] = X[i][j] + Y[i][j] #计算新矩阵中的元素值
for r in result:                         #遍历输出新矩阵中的元素
    print(r)
```

程序运行效果

```
[17, 15, 4]
[10, 12, 9]
[11, 13, 18]
[Finished in 0.1s]
```



Line 8, Column 13

Spaces: 2

### 5.1.1 列表的概述

```
def MyList_Connect(List1,List2):
def MyList_Append(SrcList,Index,AppendValue)
def MyList_Insert(SrcList,Index,InsertValue):
def MyList_InsertMoreValue(SrcList,Index,InsertValuesList):
def MyList_InsertMoreValue(SrcList,Index,InsertValuesList):
def MyList_isContain(SrcList,FindValue):
def MyList_FindFirstIndex(SrcList,FindValue):
def MyList_FindLastIndex(SrcList,FindValue):
def MyList_FindIndexList(SrcList,FindValue):
def MyList_FindCount(SrcList,FindValue):
def MyList_RemoveAt(SrcList,DelIndex):
def MyList_RemoveBetween(SrcList,beginIndex,endIndex,isDelEndIndex=True):
def MyList_RemoveFirst(SrcList,DelValue):
def MyList_RemoveLast(SrcList,DelValue):
def MyList_RemoveMore(SrcList,DelValue):
def MyList_RemoveAll(SrcList):
def MyList_Clear(SrcList):
```

### 5.1.1 列表的概述

```
def MyList_ReplaceFirst(SrcList,oldValue,newValue):
def MyList_ReplaceLast(SrcList,oldValue,newValue):
def MyList_ReplaceAll(SrcList,oldValue,newValue):
def MyList_ReplaceFromIndex(SrcList,FromIndex,ListValue):
def
MyList_ReplaceBetweenIndex(SrcList,beginIndex,endIndex,ListValue,isReplaceEndIndex=True):
def MyList_Copy(SrcList,beginIndex=0,endIndex=-1,isCopyEndIndex=True):
def MyList_Reverse(SrcList):
def MyList_Sort(SrcList,isDesc=False,isCopySrcList=True):
def MyList_SortStrList_ByLen(SrcStrList,isDesc=False,isCopySrcList=True):
def MyList_SortStrList_ByNoneCase(SrcStrList,isDesc=False,isCopySrcList=True):
def
MyList_SortTupleList(SrcTupleList,order_itemgetter1,order_itemgetter2,order_itemgetter3,order
_itemgetter4,order_itemgetter5,isDesc=False):
def MyList_Sort(SrcList,isDesc=False,isCopySrcList=True):
```

## 5.1.1 列表的概述

```
from operator import itemgetter, attrgetter
data = [('老王', 18, 175, 75), ('阿汤哥', 15, 165, 70),
        ('罗宾森', 23, 180, 100), ('小风', 10, 171, 60),
        ('黄佬', 20, 175, 65),]
get_c_d = itemgetter(2, 3)
for value in data:
    print(get_c_d(value))
print("-----")

for value in sorted(data, key=itemgetter(2, 3)):
    print(value)
print("-----")
from collections import namedtuple
size = namedtuple('size', 'height weight')
stu = namedtuple('stu', 'name age size')
data_stu = [stu(name, age, size(height, weight)) for name, age, height, weight in data]
get_name_age = attrgetter('name', 'size.height')
for value in sorted(data_stu, key=attrgetter('size.height')):
    print(get_name_age(value))
print("-----")
```

**本章结束！**

