Python 基础教程（第三版）

CH2 列表和元组

> - 数据结构：数据结构是以某种方式（如通过编号）组合起来的数据元素（如数、字符乃至其他数据结构）集合。

```
* 最基本的数据结构为序列（sequence），序列中的每个元素都有编号，即其位置或索引。
```

> - Python支持一种数据结构的基本概念，名为容器（container）。容器基本上就是可包含其他对象的对象。

```
* 两种主要的容器是序列（如列表和元组）和映射（如字典）。
* 在序列中，每个元素都有编号，而在映射中，每个元素都有名称（也叫键）。映射将在第4章详细讨论。
* 有一种既不是序列也不是映射的容器，它就是集合（set），将在第10章讨论。
```

1. **序列概述**
   - 列表和元组：列表是可以修改的，而元组不可。
2. **通用的序列操作**

   > 索引、切片、相加、相乘和成员资格检查。
   >
   > *迭代（iteration）见CH5*

   - 索引

     > Python没有专门用于表示字符的类型，因此一个字符就是只包含一个元素的字符串。
     >
     > 索引方式适用于所有序列

     - 如果函数调用返回一个序列，可直接对其执行索引操作。
       `fourth = input('Year: ')[3]`
   - 切片：访问特定范围内的元素
     - 简写：如果要从列表末尾开始数，可使用负数索引。
     - 如果切片结束于序列末尾，可省略第二个索引。

       `numbers[-3:]`
     - 如果切片始于序列开头，可省略第一个索引。
     - 要复制整个序列，可将两个索引都省略。
   - 步长
     - `numbers[0:10:2]`
     - `numbers[10:0:-2]`
     - 步长为负数时，第一个索引必须比第二个索引
   - 序列相加(同类型)
     - `[1, 2, 3] + [4, 5, 6]`

- `'Hello,' + 'world!'`
- **一般而言，不能拼接不同类型的序列。**

- 乘法
  - 'python' * 5
  - [42] * 10
- None、空列表和初始化
  - 空列表：`[]`
  - `[None] * 10`
- 成员资格

> 它检查是否满足指定的条件，并返回相应的值：满足时返回True，不满足时返回False。

  - 现在可使用运算符in来检查指定的字符串是否为另一个字符串的子串。
- 长度、最小值和最大值
  - len(numbers)
  - max(numbers)
  - min(numbers)

3. **列表**

i. 函数list

> 可将任何序列（而不仅仅是字符串）作为list的参数。
> `list('Hello')`
>
> `''.join(somelist)`

ii. 基本的列表操作

- 修改列表：给元素赋值
  - `x[1] = 2`

> 不能给不存在的元素赋值

- 删除元素
  - `del names[2]`
- 给切片赋值
  - `name[2:] = list('ar')`
  - 通过使用切片赋值，可将切片替换为长度与其不同的序列。
  - 在不替换原有元素的情况下插入新元素。
  - `numbers[1:1] = [2, 3, 4]`
  - `numbers[1:4] = []`与`del numbers[1:4]`等效。

iii. 列表方法

a. append

> - 方法append用于将一个对象附加到列表末尾。

> - append就地修改列表

b. clear

- 方法clear就地清空列表的内容。
- `lst.clear()`

c. copy

> 常规复制只是将另一个名称关联到列表
>
> 如：`b = a`

- `b = a.copy()`
- 类似于使用`a[:]`或`list(a)`，它们也都复制a。

d. count

> 方法count计算指定的元素在列表中出现了多少次。

- `['to', 'be', 'or', 'not', 'to', 'be'].count('to')`
- 必须给至少一个参数

e. extend

> 方法extend让你能够同时将多个值附加到列表末尾，为此可将这些值组成的序列作为参数提供给方法extend。

- 这可能看起来类似于拼接，但存在一个重要差别，那就是将**就地修改**被扩展的序列。
- 在常规拼接中，情况是返回一个全新的序列。
- 常规拼接必须使用a和b的副本创建一个新列表，`a = a + b`拼接的效率将比extend低。
- 要获得与extend相同的效果，可将列表赋给切片：

```
>>> a = [1, 2, 3]

>>> b = [4, 5, 6]

>>> a[len(a):] = b

>>> a

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

f. index

> 方法index在列表中查找指定值第一次出现的索引。

g. insert

> 方法insert用于将一个对象插入列表。

```
numbers.insert(3, 'four')

numbers[3:3] = ['four']
```

这虽巧妙，但可读性根本无法与使用insert媲美。

h. pop

> - 方法pop从列表中删除一个元素（末尾为最后一个元素），并返回这一元素。
> - pop是唯一既修改列表又返回一个非None值的列表方法。
> - `x.pop(index)`
> - `x.pop()`

i. remove

> - 方法remove用于删除第一个为指定值的元素。
> - 请注意：remove是就地修改且不返回值的方法之一。不同于pop的是，它修改列表，但不返回任何值。
> - `x.remove('be')`

j. reverse

> 方法reverse按相反的顺序排列列表中的元素,但不返回任何值（与remove和sort等方法一样）。

- `x.reverse()`
- 如果要按相反的顺序迭代序列，可使用函数reversed。这个函数不返回列表，而是返回一个迭代器。你可使用list将返回的对象转换为列表。
- `list(reversed(x))`

k. sort

> sort方法用于对列表就地排序(不返回任何值)。

- 为获取排序后的列表的副本，另一种方式是使用函数sorted。
- y = sorted(x)

> 这个函数可用于任何序列，但总是返回一个列表。

l. 高级排序

- 方法sort接受两个可选参数：key和reverse
  > - 参数key类似于参数cmp：你将其设置为一个用于排序的函数。然而，不会直接使用这个函数来判断一个元素是否比另一个元素小，而是使用它来为每个元素创建一个键，再根据这些键对元素进行排序

- `x.sort(key=len)`
- `x.sort(reverse=True)`
- 函数sorted也接受参数key和reverse。
- 在很多情况下，将参数key设置为一个自定义函数很有用。

> 如果你想更深入地了解排序，可以参阅文章"Sorting Mini-HOW TO"。

4. **元组**：不可修改的序列

> 与列表一样，元组也是序列，唯一的差别在于元组是不能修改的（你可能注意到了，字符串也不能修改）。

i. 表示只包含一个值的元组呢,必须在它后面加上逗号。

```
>>> 1, 2, 3
```

(1, 2, 3)

```
>>> 42,
```

(42,)

```
>>> (42,)
```

(42,)

```
>>> 3 * (40 + 2,)
```

(42, 42, 42)

> 函数tuple的工作原理与list很像：它将一个序列作为参数，并将其转换为元组。

```
>>> tuple([1, 2, 3])
```

(1, 2, 3)

```
>>> tuple('abc')
```

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

```
>>> tuple((1, 2, 3))
```

(1, 2, 3)

- 元组的创建及其元素的访问方式与其他序列相同。
- 元组的切片也是元组，就像列表的切片也是列表一样。
  *

5. **本章介绍的新函数**

函 数	描 述
len(seq)	返回序列的长度
list(seq)	将序列转换为列表
max(args)	返回序列或一组参数中的最大值
min(args)	返回序列和一组参数中的最小值
reversed(seq)	让你能够反向迭代序列
sorted(seq)	返回一个有序列表，其中包含指定序列中的所有元素
tuple(seq)	将序列转换为元组

```
print('value','...', sep='\n')
value
...
```