

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码 ？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式 [最近阅读](#)

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编程

# 17 生成器表达式和列表生成式

更新时间：2019-09-25 10:03:10



“一个不注意小事情的人，永远不会成功大事业。”  
——戴尔·卡耐基

## 列表生成式

如果我们想要构造一个包含指定元素或者具有某种规则的列表，比如 2 的指数幂序列 `[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]`，该怎么做？

- 最简单的办法，直接将这数原样写入代码来创建列表：

```
nums = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

这种办法当然是可行的，不过也有很大的局限。像这样直接将数据写入代码的做法，叫做硬编码。很多情况下我们不会使用硬编码的方式来创建列表（或者其它容器），因为列表中有啥数据往往在写代码时是不能确定的，通常在程序运行过程中通过计算得到，或从程序外部读入（比如从数据库 / 文件 / 网络中读入）。另外当数据量很大时，使用硬编码也是一件繁琐低效的事。

- 还有一种办法，创建一个空的列表，之后通过计算（或其它操作）获得各个元素，并添加到列表中：

```
nums = []
exponent = 1
while exponent <= 10:
    nums.append(2 ** exponent)
    exponent += 1
```

|   |   |
|---|---|
| <div>← 慕课专栏</div>                               | <div>≡ 你的第一本Python基础入门书 / 17 生成器表达式和列表生成式</div>   |
| <div>目录</div>                                   | <div>[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]</div>   |
| <div>第 1 章 入门准备</div>                           | <div>这样方式完全可行，也没有什么缺陷。</div>  |
| <div>01 开篇词：你为什么要学 Python？</div>                | <div>• 或者，对一个现有的可迭代对象中的各个元素做处理，构造出一个新的列表：</div>   |
| <div>02 我会怎样带你学 Python？</div>                   | <div><pre>nums = [] for i in range(1, 11):      # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]     nums.append(2 ** i)</pre></div> |
| <div>03 让 Python 在你的电脑上安家落户</div>               | <div></div>   |
| <div>04 如何运行 Python 代码？</div>                   | <div><pre>&gt;&gt;&gt; nums [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]</pre></div>                                    |
| <div>第 2 章 通用语言特性</div>                         | <div></div>   |
| <div>05 数据的名字和种类—变量和类型</div>                    | <div></div>   |
| <div>06 一串数据怎么存—列表和字符串</div>                    | <div>这段代码可以用一种简洁的方式写出，只需要一行代码：</div>  |
| <div>07 不只有一条路—分支和循环</div>                      | <div><pre>nums = [2 ** i for i in range(1, 11)]</pre></div>   |
| <div>08 将代码放进盒子—函数</div>                        | <div><pre>&gt;&gt;&gt; nums [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]</pre></div>                                    |
| <div>09 知错能改—错误处理、异常机制</div>                    | <div></div>   |
| <div>10 定制一个模子—类</div>                          | <div></div>   |
| <div>11 更大的代码盒子—模块和包</div>                      | <div>这行代码就是我们这个章节要所讲的<b>列表生成式</b>。顾名思义，列表生成式最终生成的是一个列表，它是用已有的可迭代对象来构造新列表的便捷方法。</div>                                  |
| <div>12 练习—密码生成器</div>                          | <div></div>   |
| <div>第 3 章 Python 进阶语言特性</div>                  | <div>提示：如果不清楚什么是可迭代对象，可以看一下上一篇文章《深入理解迭代器和生成器》。</div>  |
| <div>13 这么多的数据结构（一）：列表、元祖、字符串</div>             | <div></div>   |
| <div>14 这么多的数据结构（二）：字典、集合</div>                 | <div></div>   |
| <div>15 Python大法初体验：内置函数</div>                  | <div></div>   |
| <div>16 深入理解下迭代器和生成器</div>                      | <div></div>   |
| <div>17 生成器表达式和列表生成式 <a href="#">最近阅读</a></div> | <div></div>   |
| <div>18 把盒子升级为豪宅：函数进阶</div>                     | <div></div>   |
| <div>19 让你的模子更好用：类进阶</div>                      | <div></div>   |
| <div>20 从小独栋升级为别墅区：函数式编程</div>                  | <div></div>   |
| <div></div>                                     | <div>这个过程用伪代码来描述的话是这样的：</div>   |

|  |  |
|--|--|
| <div>← 慕课专栏</div> <div>目录</div> <div>第 1 章 入门准备</div> <div>01 开篇词：你为什么要学 Python ?</div> <div>02 我会怎样带你学 Python ?</div> <div>03 让 Python 在你的电脑上安家落户</div> <div>04 如何运行 Python 代码？</div> <div>第 2 章 通用语言特性</div> <div>05 数据的名字和种类—变量和类型</div> <div>06 一串数据怎么存—列表和字符串</div> <div>07 不只有一条路—分支和循环</div> <div>08 将代码放进盒子—函数</div> <div>09 知错能改—错误处理、异常机制</div> <div>10 定制一个模子—类</div> <div>11 更大的代码盒子—模块和包</div> <div>12 练习—密码生成器</div> <div>第 3 章 Python 进阶语言特性</div> <div>13 这么多的数据结构（一）：列表、元组、字符串</div> <div>14 这么多的数据结构（二）：字典、集合</div> <div>15 Python大法初体验：内置函数</div> <div>16 深入理解下迭代器和生成器</div> <div>17 生成器表达式和列表生成式 <a href="#">最近阅读</a></div> <div>18 把盒子升级为豪宅：函数进阶</div> <div>19 让你的模子更好用：类进阶</div> <div>20 从小独栋升级为别墅区：函数式编</div> | <div>for 项 in 可迭代对象:<br/>    新项 = 对项的操作(项)<br/>列表.append(新项)</div> <div>来看一个例子：</div> <div>这里有个列表： ['a', 'b', 'c', 'd', 'e'] ，怎样把其中的每个小写字母转换为大写？可以这样：</div> <div>[char.upper() for char in ['a', 'b', 'c', 'd', 'e']]</div> <div>&gt;&gt;&gt; [char.upper() for char in [ 'a' , 'b' , 'c' , 'd' , 'e' ]]<br/>[ 'A' , 'B' , 'C' , 'D' , 'E' ]</div> <div>如果你不能一下子理解，不妨比较一下用 for 循环来实现的版本。它们之间是等价的：</div> <div>chars = []<br/>for char in ['a', 'b', 'c', 'd', 'e']:<br/>    chars.append(char.upper())</div> <div>&gt;&gt;&gt; chars<br/>[ 'A' , 'B' , 'C' , 'D' , 'E' ]</div> <div>再来谈谈 [对项的操作 for 项 in 可迭代对象] 中的 对项的操作 ，这个操作它可以简单，也可以很复杂。</div> <div>简单来看，我们可以直接使用 项 本身而不做任何处理。如：</div> <div>&gt;&gt;&gt; [char for char in 'ABCDEF']<br/>[ 'A' , 'B' , 'C' , 'D' , 'E' , 'F' ]</div> <div>当然如果是要得到这个结果，我们应该直接使用 list('ABCDEF')：</div> <div>&gt;&gt;&gt; list( 'ABCDEF' )<br/>[ 'A' , 'B' , 'C' , 'D' , 'E' , 'F' ]</div> <div>复杂来看，我们可以对 项 进行一系列的处理。如分别将 'abcde' 中每个字母的大写形式和小写形式放到元组中：</div> <div>[(char.upper(), char) for char in 'abcde']</div> |
|--|--|

|   |  |
|---|--|
| <div>← 慕课专栏</div> <div>三 你的第一本Python基础入门书 / 17 生成器表达式和列表生成式</div> |  |
| 目录  | <pre>[( 'A' , 'a' ), ( 'B' , 'b' ), ( 'C' , 'c' ), ( 'D' , 'd' ), ( 'E' , 'e' )]</pre>   |
| 第 1 章 入门准备  |  |
| 01 开篇词：你为什么要学 Python ？  | 这里我们将每个 <code>char</code> 转换为了 <code>(char.upper(), char)</code> ，并且其中 <code>char</code> 被多次用到。  |
| 02 我会怎样带你学 Python ？   | 上个例子等价于：   |
| 03 让 Python 在你的电脑上安家落户  | <pre>result = [] for char in 'abcde':     result.append((char.upper(), char))</pre>  |
| 04 如何运行 Python 代码 ？   | <pre>&gt;&gt;&gt; result [( 'A' , 'a' ), ( 'B' , 'b' ), ( 'C' , 'c' ), ( 'D' , 'd' ), ( 'E' , 'e' )]</pre>   |
| 第 2 章 通用语言特性  | 列表生成式中使用 <code>if</code>   |
| 05 数据的名字和种类—变量和类型   | 在列表生成式的中，每次迭代的 <code>项</code> 是可以被筛选过滤的，使用 <code>if</code> 关键字。如：  |
| 06 一串数据怎么存—列表和字符串   | <pre>[对项的操作 for 项 in 可迭代对象 if 对项的判断]</pre>   |
| 07 不只有一条路—分支和循环   | 它的阅读顺序是： <code>for 项 in 可迭代对象</code> -> <code>if 对项的判断</code> -> <code>对项的操作</code> 。  |
| 08 将代码放进盒子—函数   | 每次迭代时所取出的 <code>项</code> ，要先经过 <code>对项的判断</code> ，如果结果为 <code>True</code> ，才会由 <code>对项的操作</code> 处理。如果 <code>对项的判断</code> 的结果为 <code>False</code> ，后续 <code>对项的操作</code> 会被跳过，此时最终列表的长度也会减少。 |
| 09 知错能改—错误处理、异常机制   | 举个例子， <code>[2 ** i for i in range(1, 11)]</code> 可以生成出 20 ~ 210 间的整数，如果我们只想要其中的奇数次方的值，该怎么做？   |
| 10 定制一个模子—类   | 这时就可以在列表中使用 <code>if</code> 关键字：   |
| 11 更大的代码盒子—模块和包   | <pre>[2 ** i for i in range(1, 11) if i % 2 == 1]</pre>  |
| 12 练习—密码生成器   | <pre>&gt;&gt;&gt; [2 ** i for i in range(1, 11) if i % 2 == 1] [2, 8, 32, 128, 512]</pre>  |
| 第 3 章 Python 进阶语言特性   | 这里的阅读顺序是：  |
| 13 这么多的数据结构（一）：列表、元祖、字符串  | <ol style="list-style-type: none"><li>1. <code>for e in range(1, 11)</code></li><li>2. <code>if e % 2 == 1</code></li><li>3. <code>2 ** e</code></li></ol>                                     |
| 14 这么多的数据结构（二）：字典、集合  | 上述代码等价于：   |
| 15 Python大法初体验：内置函数   | <pre>nums = [] for i in range(1, 11):</pre>  |
| 16 深入理解下迭代器和生成器   |  |
| 17 生成器表达式和列表生成式 <a href="#">最近阅读</a>                              |  |
| 18 把盒子升级为豪宅：函数进阶  |  |
| 19 让你的模子更好用：类进阶   |  |
| 20 从小独栋升级为别墅区：函数式编程   |  |

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式 [最近阅读](#)

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编

```
>>> nums
[2, 8, 32, 128, 512]
```

列表生成式中嵌套 for

列表生成式中的 `for` 中还可以再嵌套 `for` 。如：

```
[对项1和(或)项2的操作 for 项1 in 可迭代对象1 for 项2 in 可迭代对象2]
```

它等价于：

```
列表 = []
for 项1 in 可迭代对象1:
    for 项2 in 可迭代对象2:
        新项 = 对项1和(或)项2的操作()
        列表.append(新项)
```

看起来有点复杂，我们看个例子：

```
nums = [1, 2, 3]
chars = ['a', 'b', 'c']
[c * n for n in nums for c in chars]
```

```
>>> nums = [1, 2, 3]
>>> chars = ['a', 'b', 'c']
>>> [c * n for n in nums for c in chars]
['a', 'b', 'c', 'aa', 'bb', 'cc', 'aaa', 'bbb', 'ccc']
```

它等价于：

```
nums = [1, 2, 3]
chars = ['a', 'b', 'c']
result = []
for n in nums:
    for c in chars:
        result.append(c * n)
```

```
>>> result
['a', 'b', 'c', 'aa', 'bb', 'cc', 'aaa', 'bbb', 'ccc']
```

[对项1和(或)项2的操作 for 项1 in 可迭代对象1 for 项2 in 可迭代对象2] 中的 可迭代对象2 可以是项1 本身。也就是可以写成：

```
[对项和(或)子项的操作 for 项 in 可迭代对象 for 子项 in 项]
```

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码 ？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式 [最近阅读](#)

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编程

```
strings = ['aa', 'bb', 'cc']
[char for string in strings for char in string]
```

```
>>> strings = [ 'aa' , 'bb' , 'cc' ]
>>> [char for string in strings for char in string]
[ 'a' , 'a' , 'b' , 'b' , 'c' , 'c' ]
```

它等价于：

```
strings = ['aa', 'bb', 'cc']
result = []
for string in strings:
    for char in string:
        result.append(char)
```

```
>>> result
[ 'a' , 'a' , 'b' , 'b' , 'c' , 'c' ]
```

字典生成式

便捷地构造列表可以使用列表生成式，同样的，想要通过已有的可迭代对象来便捷地构造字典，可以使用字典生成式。

字典生成式的写法是：

```
{键: 值 for 项 in 可迭代对象}
```

和列表生成式非常相似，不同之处在于它使用的是花括号（ {} ），另外还使用 键: 值 形式。

举个例子，有字符串 'abcde'，以每个小写字母作为键，对应大写字母作为值的来构造个字典：

```
{char: char.upper() for char in 'abcde'}
```

```
>>> {char: char.upper() for char in 'abcde' }
{ 'a': 'A' , 'b': 'B' , 'c': 'C' , 'd': 'D' , 'e': 'E' }
```

同样的，字典生成式中也可以使用 if 和嵌套 for，使用方法参照列表生成式。

集合生成式

想要通过已有的可迭代对象来构造集合，可以使用集合生成式。

|   |  |
|---|--|
| <div><div>← 慕课专栏</div><div>三 你的第一本Python基础入门书 / 17 生成器表达式和列表生成式</div></div> |  |
| 目录  | (对项的操作 for 项 in 可迭代对象)   |
| 第 1 章 入门准备  | 例如：  |
| 01 开篇词：你为什么要学 Python？   | {char.lower() for char in 'ABCDABCD'}  |
| 02 我会怎样带你学 Python？  | <pre>&gt;&gt;&gt; {char.lower() for char in 'ABCDABCD'} {'c', 'a', 'd', 'b'}</pre>   |
| 03 让 Python 在你的电脑上安家落户  |  |
| 04 如何运行 Python 代码？  | 提示：通过这个例子也能看到集合的重要特性——无序且无重复。  |
| 第 2 章 通用语言特性  |  |
| 05 数据的名字和种类—变量和类型   | 同样的，集合生成式也可以使用 if 和嵌套 for。   |
| 06 一串数据怎么存—列表和字符串   |  |
| 07 不只有一条路—分支和循环   | 生成器表达式   |
| 08 将代码放进盒子—函数   | 上面有列表生成式、字典生成式、集合生成式，那么是不是也有「元组生成式」？是不是用圆括号来表示就可以了？  |
| 09 知错能改—错误处理、异常机制   | 不是的，Python 中并没有「元组生成式」！虽然 Python 中确实有类似的圆括号的写法：  |
| 10 定制一个模子—类   | (对项的操作 for 项 in 可迭代对象)   |
| 11 更大的代码盒子—模块和包   | 但这可不是什么「元组生成式」，而是我们上一章节学习过的生成器表达式。   |
| 12 练习—密码生成器   | 生成器表达式是一种创建生成器的便捷方法。虽然写法上和列表生成式、字典生成式、集合生成式相似，却有着本质的不同，因为它创建出来的是生成器，而不是列表、字典、集合这类容器。   |
| 第 3 章 Python 进阶语言特性   | (char.lower() for char in 'ABCDEF')  |
| 13 这么多的数据结构（一）：列表、元组、字符串  | <pre>&gt;&gt;&gt; g = (char.lower() for char in 'ABCDEF') &gt;&gt;&gt; g &lt;generator object at 0x103da6c78&gt; &gt;&gt;&gt; next(g) 'a' &gt;&gt;&gt; next(g) 'b'</pre> |
| 14 这么多的数据结构（二）：字典、集合  |  |
| 15 Python大法初体验：内置函数   | 提示：如果你对生成器有些遗忘，不妨看下前一篇文章《深入理解迭代器和生成器》。   |
| 16 深入理解下迭代器和生成器   |  |
| 17 生成器表达式和列表生成式 <a href="#">最近阅读</a>  | 生成器表达式中同样可以使用 if 和嵌套 for，使用方法和列表生成式相同。   |
| 18 把盒子升级为豪宅：函数进阶  |  |
| 19 让你的模子更好用：类进阶   |  |
| 20 从小独栋升级为别墅区：函数式编程   |  |

|   |  |
|---|--|
| <div><div>← 慕课专栏</div><div>≡ 你的第一本Python基础入门书 / 17 生成器表达式和列表生成式</div></div> |  |
| 目录  |  |
| 第 1 章 入门准备  | 精选留言 0   |
| 01 开篇词：你为什么要学 Python ？  | <div>欢迎在这里发表留言，作者筛选后可公开显示</div> <div><div>！</div><div>目前暂无任何讨论</div></div> |
| 02 我会怎样带你学 Python ？   |  |
| 03 让 Python 在你的电脑上安家落户  |  |
| 04 如何运行 Python 代码 ？   |  |
| 第 2 章 通用语言特性  | <div>千学不如一看，千看不如一练</div>   |
| 05 数据的名字和种类—变量和类型   |  |
| 06 一串数据怎么存—列表和字符串   |  |
| 07 不只有一条路—分支和循环   |  |
| 08 将代码放进盒子—函数   |  |
| 09 知错能改—错误处理、异常机制   |  |
| 10 定制一个模子—类   |  |
| 11 更大的代码盒子—模块和包   |  |
| 12 练习—密码生成器   |  |
| 第 3 章 Python 进阶语言特性   |  |
| 13 这么多的数据结构（一）：列表、元祖、字符串  |  |
| 14 这么多的数据结构（二）：字典、集合  |  |
| 15 Python大法初体验：内置函数   |  |
| 16 深入理解下迭代器和生成器   |  |
| 17 生成器表达式和列表生成式 <a href="#">最近阅读</a>  |  |
| 18 把盒子升级为豪宅：函数进阶  |  |
| 19 让你的模子更好用：类进阶   |  |
| 20 从小独栋升级为别墅区：函数式编  |  |