

5.3. 使用list和tuple

5.4. 条件判断

5.5. 模式匹配

5.6. 循环

5.7. 使用dict和set

6. 函数

6.1. 调用函数

6.2. 定义函数

6.3. 函数的参数

6.4. 递归函数

7. 高级特性

7.1. 切片

7.2. 迭代

7.3. 列表生成式

7.4. 生成器

7.5. 迭代器

8. 函数式编程

8.1. 高阶函数

8.1.1. map/reduce

8.1.2. filter

8.1.3. sorted

8.2. 返回函数

8.3. 匿名函数

8.4. 装饰器

8.5. 偏函数

9. 模块

9.1. 使用模块

9.2. 安装第三方模块

10. 面向对象编程

10.1. 类和实例

10.2. 访问限制

10.3. 继承和多态

10.4. 获取对象信息

10.5. 实例属性和类属性

11. 面向对象高级编程

11.1. 使用__slots__

11.2. 使用@property

11.3. 多重继承

11.4. 定制类

11.5. 使用枚举类

11.6. 使用元类

12. 错误、调试和测试

12.1. 错误处理

12.2. 调试

12.3. 单元测试

12.4. 文档测试

13. IO编程

列表生成式



廖雪峰



资深软件开发工程师，业余马拉松选手。

列表生成式即List Comprehensions，是Python内置的非常简单却强大的可以用来创建list的生成式。

举个例子，要生成list `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]` 可以用 `list(range(1, 11))`：

```
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

但如果要生成 `[1x1, 2x2, 3x3, ..., 10x10]` 怎么做？方法一是循环：

```
>>> L = []
>>> for x in range(1, 11):
...     L.append(x * x)
...
>>> L
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

但是循环太繁琐，而列表生成式则可以用一行语句代替循环生成上面的list：

```
>>> [x * x for x in range(1, 11)]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

写列表生成式时，把要生成的元素 `x * x` 放到前面，后面跟 `for` 循环，就可以把list创建出来，十分有用，多写几次，很快就可以熟悉这种语法。

for循环后面还可以加上if判断，这样我们就可以筛选出仅偶数的平方：

```
>>> [x * x for x in range(1, 11) if x % 2 == 0]
[4, 16, 36, 64, 100]
```

还可以使用两层循环，可以生成全排列：

```
>>> [m + n for m in 'ABC' for n in 'XYZ']
['AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'CX', 'CY', 'CZ']
```

三层和三层以上的循环就很少用到了。

运用列表生成式，可以写出非常简洁的代码。例如，列出当前目录下的所有文件和目录名，可以通过一行代码实现：

```
>>> import os # 导入os模块，模块的概念后面讲到
>>> [d for d in os.listdir('.')] # os.listdir可以列出文件和目录
['.emacs.d', '.ssh', '.Trash', 'Adlm', 'Applications', 'Desktop', 'Documents', 'Downloads', 'Libr
```

`for` 循环其实可以同时使用两个甚至多个变量，比如 `dict` 的 `items()` 可以同时迭代key和value：

```
>>> d = {'x': 'A', 'y': 'B', 'z': 'C'}
>>> for k, v in d.items():
...     print(k, '=', v)
...
y = B
x = A
z = C
```

因此，列表生成式也可以使用两个变量来生成list：

```
>>> d = {'x': 'A', 'y': 'B', 'z': 'C'}
>>> [k + '=' + v for k, v in d.items()]
['y=B', 'x=A', 'z=C']
```

最后把一个list中所有的字符串变成小写：

5.3. 使用list和tuple

5.4. 条件判断

5.5. 模式匹配

5.6. 循环

5.7. 使用dict和set

6. 函数

6.1. 调用函数

6.2. 定义函数

6.3. 函数的参数

6.4. 递归函数

7. 高级特性

7.1. 切片

7.2. 迭代

7.3. 列表生成式

7.4. 生成器

7.5. 迭代器

8. 函数式编程

8.1. 高阶函数

8.1.1. map/reduce

8.1.2. filter

8.1.3. sorted

8.2. 返回函数

8.3. 匿名函数

8.4. 装饰器

8.5. 偏函数

9. 模块

9.1. 使用模块

9.2. 安装第三方模块

10. 面向对象编程

10.1. 类和实例

10.2. 访问限制

10.3. 继承和多态

10.4. 获取对象信息

10.5. 实例属性和类属性

11. 面向对象高级编程

11.1. 使用__slots__

11.2. 使用@property

11.3. 多重继承

11.4. 定制类

11.5. 使用枚举类

11.6. 使用元类

12. 错误、调试和测试

12.1. 错误处理

12.2. 调试

12.3. 单元测试

12.4. 文档测试

13. IO编程

```
>>> L = ['Hello', 'World', 'IBM', 'Apple']
>>> [s.lower() for s in L]
['hello', 'world', 'ibm', 'apple']
```

if ... else

使用列表生成式的时候，有些童鞋经常搞不清楚 `if...else` 的用法。

例如，以下代码正常输出偶数：

```
>>> [x for x in range(1, 11) if x % 2 == 0]
[2, 4, 6, 8, 10]
```

但是，我们不能在最后的 `if` 加上 `else`：

```
>>> [x for x in range(1, 11) if x % 2 == 0 else 0]
File "<stdin>", line 1
    [x for x in range(1, 11) if x % 2 == 0 else 0]
                                   ^
SyntaxError: invalid syntax
```

这是因为跟在 `for` 后面的 `if` 是一个筛选条件，不能带 `else`，否则如何筛选？

另一些童鞋发现把 `if` 写在 `for` 前面必须加 `else`，否则报错：

```
>>> [x if x % 2 == 0 for x in range(1, 11)]
File "<stdin>", line 1
    [x if x % 2 == 0 for x in range(1, 11)]
      ^
SyntaxError: invalid syntax
```

这是因为 `for` 前面的部分是一个表达式，它必须根据 `x` 计算出一个结果。因此，考察表达式：`x if x % 2 == 0`，它无法根据 `x` 计算出结果，因为缺少 `else`，必须加上 `else`：

```
>>> [x if x % 2 == 0 else -x for x in range(1, 11)]
[-1, 2, -3, 4, -5, 6, -7, 8, -9, 10]
```

上述 `for` 前面的表达式 `x if x % 2 == 0 else -x` 才能根据 `x` 计算出确定的结果。

可见，在一个列表生成式中，`for` 前面的 `if ... else` 是表达式，而 `for` 后面的 `if` 是过滤条件，不能带 `else`。

练习

如果list中既包含字符串，又包含整数，由于非字符串类型没有 `lower()` 方法，所以列表生成式会报错：

```
>>> L = ['Hello', 'World', 18, 'Apple', None]
>>> [s.lower() for s in L]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<stdin>", line 1, in <listcomp>
AttributeError: 'int' object has no attribute 'lower'
```

使用内建的 `isinstance` 函数可以判断一个变量是不是字符串：

```
>>> x = 'abc'
>>> y = 123
>>> isinstance(x, str)
True
>>> isinstance(y, str)
False
```

请修改列表生成式，通过添加 `if` 语句保证列表生成式能正确地执行：

```
L1 = ['Hello', 'World', 18, 'Apple', None]
L2 = ???

# 测试：
print(L2)
if L2 == ['hello', 'world', 'apple']:
```

- 5.3. 使用list和tuple
- 5.4. 条件判断
- 5.5. 模式匹配
- 5.6. 循环
- 5.7. 使用dict和set

6. 函数

- 6.1. 调用函数
- 6.2. 定义函数
- 6.3. 函数的参数
- 6.4. 递归函数

7. 高级特性

- 7.1. 切片
- 7.2. 迭代
- 7.3. 列表生成式
- 7.4. 生成器

```
print('测试通过!')
else:
    print('测试失败!')
```

参考源码

do_list_compr.py

小结

运用列表生成式，可以快速生成list，可以通过一个list推导出另一个list，而代码却十分简洁。

« 迭代

生成器 »



廖雪峰的官方网站 [Java教程](#) [Python教程](#) [JavaScript教程](#) [SQL教程](#) [手写Spring](#) [手写Tomcat](#) [区块链教程](#) [Git教程](#) [Makefile教程](#) [博客](#)



8. 函数式编程

- 8.1. 高阶函数
 - 8.1.1. map/reduce
 - 8.1.2. filter
 - 8.1.3. sorted
- 8.2. 返回函数
- 8.3. 匿名函数
- 8.4. 装饰器
- 8.5. 偏函数

9. 模块

- 9.1. 使用模块
- 9.2. 安装第三方模块

10. 面向对象编程

- 10.1. 类和实例
- 10.2. 访问限制
- 10.3. 继承和多态
- 10.4. 获取对象信息
- 10.5. 实例属性和类属性

11. 面向对象高级编程

- 11.1. 使用__slots__
- 11.2. 使用@property
- 11.3. 多重继承
- 11.4. 定制类
- 11.5. 使用枚举类
- 11.6. 使用元类

12. 错误、调试和测试

- 12.1. 错误处理
- 12.2. 调试
- 12.3. 单元测试
- 12.4. 文档测试

13. IO编程

Comments

Loading comments...

©liaoxuefeng.com - 微博 - GitHub - License