🡪

**List Comprehension**

List Comprehension的格式为：

**new\_list = [expression for member in iterable (if conditional)]**

其中包含三个元素。

第一部分为expression为调用一个方法或者表达式，接受member作为参数。**如果expression所代表的逻辑简单，可以直接在expression中插入代码，如果expression所代表的逻辑复杂，那么先额外定义一个函数，然后在expression中进行调用。**

第二部分为member，其为iterable可迭代对象的每个元素。

第三部分为iterable对象。

第四部分为条件，可以调用过滤方法或者表达式，接受member作为参数。

List Comprehension完成后的结果为一个列表，列表中元素为每一次第一部分所指定的方法或者表达式的返回结果。

例子：

txns = [1.09, 23.56, 57.84, 4.56, 6.78]

TAX\_RATE = .08  
def get\_price\_with\_tax(txn):  
 return txn \* (1 + TAX\_RATE)  
final\_prices = [get\_price\_with\_tax(i) for i in txns]  
print(final\_prices)

例子2，使用if条件语句：

sentence = 'the rocket came back from mars'

vowels = [i for i in sentence if i in 'aeiou']  
print(vowels)

例子3， 使用双if条件语句

num\_list = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]  
print(num\_list) # 结果为[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

**Set Comprehension**

Set Comprehension的格式用法与list类似，不同是最后得出的数据结构是set, 使用花括号{}.也意味着结果将自动去重。

**new\_set = {expression for member in iterable (if conditional)}**

**Dictionary Comprehension**

Dictionary Comprehension的格式用法和list类似，不同是得出的数据结构是字典, 使用花括号{}, 并且还要在表达式中表明key

**new\_dict = {key : expression for member in iterable (if conditional)}**

🡪**使用List comprehension代替简单for循环，map，filter函数**

在Python支持函数式特性编程特性中, Python提供了map, filter, reduce函数。但这些是Python早期支持函数式编程特性中引入。为了让代码更加的Pythonic, 我们可以使用list comprehension或者生成器表达式(generator expression)来代替简单的forloop(复杂的forloop无法代替)，以及map, filter, reduce函数。

总结来说，对于要执行map, filter的逻辑，我们可以有两种风格实现。

我们可以使用函数时风格的内置函数，map, filter.简单的逻辑直接匿名函数lambda,复杂的逻辑则额外定义函数，作为引用传入。

我们也可以使用具体Python风格的list comprehension进行处理。简单的逻辑直接expression插入代码,复杂的逻辑则额外定义函数，在expression中调用。

**使用list comprehension代替简单for循环函数**

例子：

numbers = []  
for i in range(1, 11): # 使用普通for循环，将一个个元素放入numbers list当中  
 numbers.append(i)  
numbers = [i for i in range(1, 11)] # 使用list comprehension，每一次左侧expression的返回值作为返回列表的元素，结果也为[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

**使用list comprehension或者生成器表达式（generator expression）代替map函数**

**例子1：使用list comprehension代替map函数**

# Generating a list with map  
list(map(function, iterable)) # 使用map函数写法，指定方法引用（非调用），返回map对象最后使用list生成list类型变量  
  
# Generating a list with a list comprehension  
[function(x) for x in iterable] # 使用list comprehension写法，左侧为expression，调用一个方法或者表达式，接受每个iterable元素x变量

**例子2：使用list comprehension代替map函数**

# Transformation function  
def square(number): # 指定一个方法，用于计算出传入变量的平方  
 return number \*\* 2  
  
numbers = [1, 2, 3, 4, 5, 6]  
  
# Using map()  
list(map(square, numbers)) # 使用map函数写法，指定方法引用（非调用）square，返回map对象最后使用list生成list类型变量， 结果为[1, 4, 9, 16, 25, 36]  
  
# Using a list comprehension  
[square(x) for x in numbers] # 使用list comprehension写法，左侧为expression，调用square(x)返回每一次轮询结果，x变量接受每个iterable元素，结果为[1, 4, 9, 16, 25, 36]。 不需要调用map函数，以及list转换类型。更加易读。

**例子3：使用生成器表达式（generator expression）代替map函数**

Python3中map函数的返回将会是一个map对象，map对象是一个迭代器。迭代器和生成器在Python中意味着内存使用效率很高。因为他们在轮询当中，逐一生成取出元素。而不像list那样，需要在内存中直接开辟能保存整个list大小的空间内存。

# Transformation function  
def square(number): # 定义函数用于计算传入参数的平方  
 return number \*\* 2  
  
numbers = [1, 2, 3, 4, 5, 6]  
  
# Using map()  
map\_obj = map(square, numbers) # 使用map函数，指定方法引用（非调用）square，返回map对象结果为<map object at 0x7f254d180a60>。 接下来，可以用list(map\_obj)将map对象这种迭代器变为list类型。或者直接进行轮询，由于是迭代器，所以不会直接开辟保存整个迭代器数据的内存空间，而是在轮询中逐一取出。  
  
# Using a generator expression  
gen\_exp = (square(x) for x in numbers) # 使用生成器表达式generator expression.与list comprehension相比，只是将左右的[]变为(). 生成的结果为<generator object <genexpr> at 0x7f254e056890>.接下来，可以用list(gen\_exp)将生成器对象变为list类型。或者直接进行轮询，由于是迭代器，所以不会直接开辟保存整个生成器数据的内存空间，而是在轮询中逐一取出。

**使用list comprehension或者生成器表达式（generator expression）代替filter函数**

例子：

numbers = range(10)  
odd\_numbers = list(filter(lambda x: x % 2 == 1, numbers)) # 利用filter函数，使用lambda自定义一个匿名函数，该匿名函数引用作为filter函数参数。  
print(odd\_numbers)

odd\_numbers = [i for i in range(10) if i % 2 == 1]# 利用list comprehension完整语法[expression for member in iterable (if conditional)]的右侧， 进入条件进行filter  
print(odd\_numbers)

🡪**使用内嵌List comprehension生成矩阵数据结构**

例子：

matrix = []  
for i in range(5): # 使用普通循环生成，需要使用两层for循环  
 row = []   
 for j in range(5):  
 row.append(i \* 5 + j)  
 matrix.append(row)  
print(matrix)  
# 结果为[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19], [20, 21, 22, 23, 24]]

matrix = [[i \* 5 + j for j in range(5)] for i in range(5)] # 使用nested list comprehension生成,在内层expression 语句中，再次使用list comprehesion  
print(matrix)

# 结果同样为[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19], [20, 21, 22, 23, 24]]