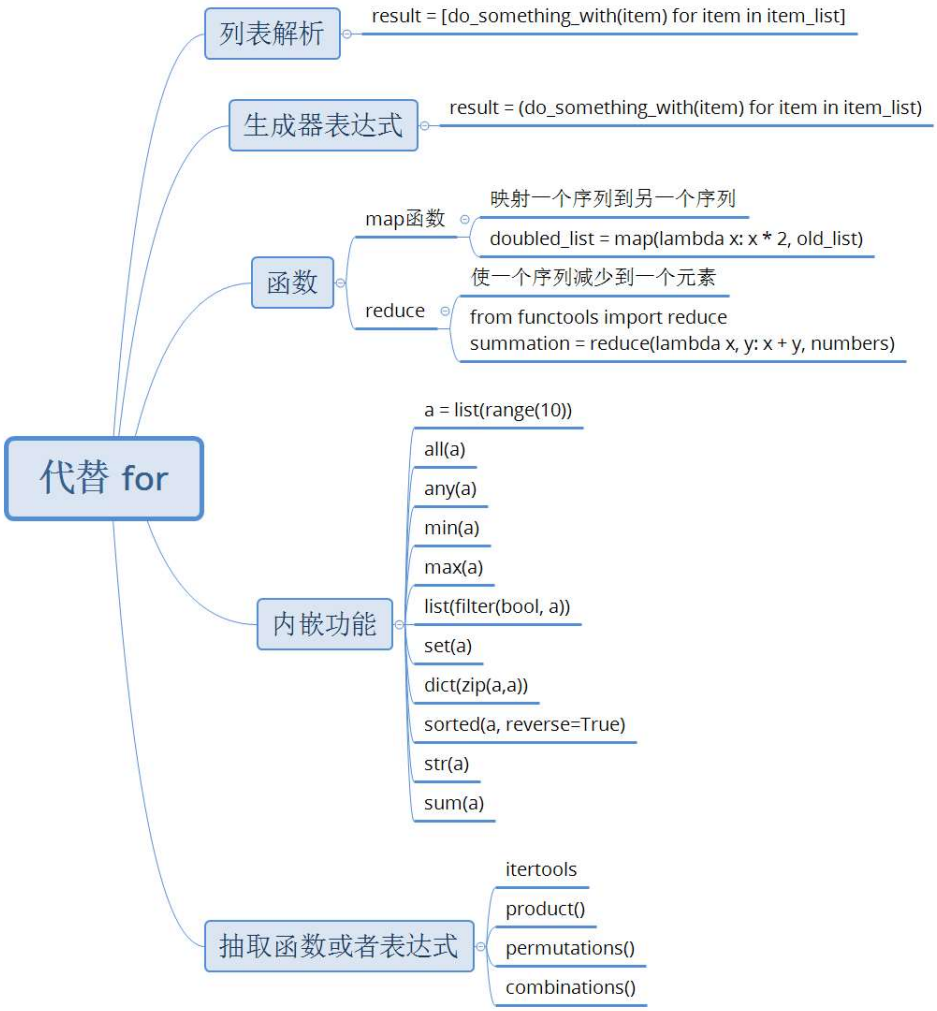
挑战不用 for 循环



不到处写for循环你将会获得什么

1. 更少的代码行数
2. 更好的代码阅读性
3. 只将缩进用于管理代码文本

Let’s see the code skeleton below:

看看下面这段代码的构架：

```
# 1
with ...:
    for ...:
        if ...:
```

```
            try:
            except:
        else:
```

这个例子使用了多层嵌套的代码，**这是非常难以阅读的。**我在这段代码中发现它无差别使用缩进把管理逻辑（with, try-except）和业务逻辑（for, if）混在一起。如果你遵守只对管理逻辑使用缩进的规范，那么核心业务逻辑应该立刻脱离出来

“扁平结构比嵌套结构更好” –《Python之禅》

为了避免for循环，你可以使用这些工具

1. 列表解析/生成器表达式

看一个简单的例子，这个例子主要是根据一个已经存在的序列编译一个新序列：

```
result = []
for item in item_list:
    new_item = do_something_with(item)
    result.append(item)
```

如果你喜欢MapReduce，那你可以使用map，或者Python的列表解析：

```
result = [do_something_with(item) for item in item_list]
```

同样的，如果你只是想要获取一个迭代器，你可以使用语法几乎相通的生成器表达式。

```
result = (do_something_with(item) for item in item_list)
```

2. 函数

站在更高阶、更函数化的变成方式考虑一下，如果你想映射一个序列到另一个序列，直接调用map函数。（也可用列表解析来替代。）

```
doubled_list = map(lambda x: x * 2, old_list)
```

如果你想使一个序列减少到一个元素，使用reduce

```
from functools import reduce
summation = reduce(lambda x, y: x + y, numbers)
```

另外，Python中大量的内嵌功能会（我不知道这是好事还是坏事，你选一个，不加这个句子有点难懂）消耗迭代器：

```
>>> a = list(range(10))
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> all(a)
False
>>> any(a)
True
>>> max(a)
9
>>> min(a)
0
>>> list(filter(bool, a))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> set(a)
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> dict(zip(a,a))
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9}
>>> sorted(a, reverse=True)
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> str(a)
'[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]'
>>> sum(a)
45
```

3. 抽取函数或者表达式

上面的两种方法很好地处理了较为简单的逻辑，那更复杂的逻辑怎么办呢？作为一个程序员，我们会把困难的事情抽象成函数，这种方式也可以用在这里。如果你写下了这种代码：

```
results = []
for item in item_list:
    # setups
    # condition
    # processing
    # calculation
    results.append(result)
```

显然你赋予了一段代码太多的责任。为了改进，我建议你这样做：

```
def process_item(item):
    # setups
    # condition
    # processing
    # calculation
    return result


results = [process_item(item) for item in item_list]
```

嵌套的for循环怎么样？

```
results = []
for i in range(10):
    for j in range(i):
        results.append((i, j))
```

列表解析可以帮助你：

```
results = [(i, j)
           for i in range(10)
           for j in range(i)]
```

如果你要保存很多的内部状态怎么办呢?

```
# finding the max prior to the current item
a = [3, 4, 6, 2, 1, 9, 0, 7, 5, 8]
results = []
current_max = 0
for i in a:
    current_max = max(i, current_max)
    results.append(current_max)

# results = [3, 4, 6, 6, 6, 9, 9, 9, 9, 9]
```

让我们提取一个表达式来实现这些:

```
def max_generator(numbers):
    current_max = 0
    for i in numbers:
        current_max = max(i, current_max)
        yield current_max

a = [3, 4, 6, 2, 1, 9, 0, 7, 5, 8]
results = list(max_generator(a))
```

“等等，你刚刚在那个函数的表达式中使用了一个for循环，这是欺骗!”

好吧，自作聪明的家伙，试试下面的这个。

4. 你自己不要写for循环，itertools会为你代劳

这个模块真是妙。我相信这个模块能覆盖80%你想写下for循环的时候。例如，上一个例子可以这样改写:

```
from itertools import accumulate
a = [3, 4, 6, 2, 1, 9, 0, 7, 5, 8]
```

```
resutls = list(accumulate(a, max))
```

另外，如果你在迭代组合的序列，还有product()，permutations()，combinations()可以用。

结论

1. 大多数情况下是不需要写for循环的。
2. 应该避免使用for循环，这样会使得代码有更好的阅读性。

行动

1. 再看一遍你的代码，找出任何以前凭直觉写下for循环的地方，再次思考一下，不用for循环再写一遍是不是有意义的。
2. 分享你很难不使用for循环的例子。