1.19 转换并同时计算数据¶

问题¶

你需要在数据序列上执行聚集函数（比如 `sum()` , `min()` , `max()` ）， 但是首先你需要先转换或者过滤数据

解决方案¶

一个非常优雅的方式去结合数据计算与转换就是使用一个生成器表达式参数。 比如，如果你想计算平方和，可以像下面这样做：

```
nums = [1, 2, 3, 4, 5]
s = sum(x * x for x in nums)
```

下面是更多的例子：

```
# Determine if any .py files exist in a directory
import os
files = os.listdir('dirname')
if any(name.endswith('.py') for name in files):
    print('There be python!')
else:
    print('Sorry, no python.')
# Output a tuple as CSV
s = ('ACME', 50, 123.45)
print(','.join(str(x) for x in s))
# Data reduction across fields of a data structure
portfolio = [
    {'name':'GOOG', 'shares': 50},
    {'name':'YHOO', 'shares': 75},
    {'name':'AOL', 'shares': 20},
    {'name':'SCOX', 'shares': 65}
]
min_shares = min(s['shares'] for s in portfolio)
```

讨论¶

上面的示例向你演示了当生成器表达式作为一个单独参数传递给函数时候的巧妙语法（你并不需要多加一个括号）。 比如，下面这些语句是等效的：

```
s = sum((x * x for x in nums)) # 显式的传递一个生成器表达式对象
s = sum(x * x for x in nums) # 更加优雅的实现方式，省略了括号
```

使用一个生成器表达式作为参数会比先创建一个临时列表更加高效和优雅。 比如，如果你不使用生成器表达式的话，你可能会考虑使用下面的实现方式：

```
nums = [1, 2, 3, 4, 5]
s = sum([x * x for x in nums])
```

这种方式同样可以达到想要的效果，但是它会多一个步骤，先创建一个额外的列表。 对于小型列表可能没什么关系，但是如果元素数量非常大的时候， 它会创建一个巨大的仅仅被使用一次就被丢弃的临时数据结构。而生成器方案会以迭代的方式转换数据，因此更省内存。

在使用一些聚集函数比如 `min()` 和 `max()` 的时候你可能更加倾向于使用生成器版本， 它们接受的一个 key 关键字参数或许对你很有帮助。 比如，在上面的证券例子中，你可能会考虑下面的实现版本：

```
# Original: Returns 20
min_shares = min(s['shares'] for s in portfolio)
# Alternative: Returns {'name': 'AOL', 'shares': 20}
min_shares = min(portfolio, key=lambda s: s['shares'])
```