# PCS114　FP 初体验

## 概述

Python 不是一门纯粹的函式编程语言，它也加入了一些可以有效提高编程效率的函式编程特性。

函式编程，英文写作 Functional Programming，原意并非以函式来组织程序结构，直白地说，FP 是指将程序的行为抽象出来，作为基本的元素，以此组织程序。它来自于与图灵机同时提出的 Lambda 计算模型。相对于具有高度可操作性（现代我们所见的计算机基本都是图灵机）的图灵机模型，Lambda 模型在数学上更为优美，因此，产生了 FP 语言这样的技术流派，通过严格定义程序的行为来使程序可控性更好，更为优美。相对来说，FP 语言也需要较深入的学习。

## 应用

在函式编程中，最著名的特色就是高序（High Order）。简单地说，就是定制一个算法，按规则来指定容器中的每一个元素。最常用的 High Order 为：

- 映射，也就是将算法施于每个元素，将返回值合并为一个新的容器。
- 过滤，将算法施于每个元素，将返回值为真的元素合并为一个新的容器。
- 合并，将算法（可能携带一个初值）依次施于每个元素，将返回值作为下一步计算的参数之一，与下一个元素再计算，直至最终成为一个总的结果。

Python 通过 map、filter、reduce 三个内置函式来实现这三个 High Order 功能。

## 使用 map

函式 map 至少需要两个参数，第一个是一个函式，第二个是传入函式的参数。例如：

```
def foo(x):
    return x*x

print map(foo, range(10))
```

以上代码得到 10 以内的自然数平方表。

```
~$ python pcs-114-1.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

map 允许接收三个或三个以上的参数，从第二个开始，每个参数都接收一个线性容器（或迭代对象），将每个元素提取出来做为第一个参数（函式）的参数列表。如果各个容器的长度不一样，短缺的部分用 None 补齐。例如可以这样使用：

```
def foo(x, y):
    return x**y

print map(foo, range(10), range(10))
~$ python pcs-114-2.py
[1, 1, 4, 27, 256, 3125, 46656, 823543, 16777216, 387420489]
```

这个计算结果会很大，所以最好不要用太大的数来测试，不过得益于 Python 的长整数，仍然可以得到计算结果。如果直接这样用：map(foo, range(10), range(20))，会收到异常，因为定义的这个 foo 不接受 None。

如果 map 的第一个参数为 None，那么返回原来的序列，如果传入了多个序列，会将其中每个位置的参数打包成 tuple。如：

```
print map(None, range(10), range(10))
~$ python pcs-114-3.py
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8),
(9, 9)]
```

## 使用 filter

英文单词 filter 的字面意义有过滤的意思，实际也如此。例如可以用下面的方法得到 100 以内的偶数列：

```
def foo(x):
    return x%2==0
print filter(foo, range(100))
~$ python pcs-114-4.py
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38,
40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76,
78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98]
```

如果 filter 的第一个参数是 None，则返回序列中所有的真值。例如 True、非空序列，非零数值等。

## 使用 reduce

filter 可以方便的实现有条件的选择，但是如果想要筛法计算素数列，用 filter 就不够高效了。计算 N 是否为素数最有效的方法应该只计算已得到的素数列中，小于 N 的平方根的那部分。而 filter 并不保存前一步的计算状态（从 FP 理论上讲，每个 filter 运算之间是无关的，它们可以并行执行）。需要将上一步计算的结果作为当前计算的一部分时，Python 的内置函式 reduce 就派上用场了。

```
    def foo(perms, x):
        i = 0
        while perms[i]**2 <= x:
            if x%perms[i] == 0:
                return perms
            else:
                i += 1
        else:
            perms.append(x)
        return perms

    print reduce(foo, range(5, 100, 2), [2, 3])
~$ python pcs-114-5.py
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97]
```

因为 1、2、3、5 这几个数已经确定为素数，并且大于 2 的偶数显然不是素数，所以这里对传统筛法做了很普通的优化。这里可以看到，通过 reduce，方便地实现了计算结果的重用，节省了大量的 CPU 周期。在实际应用中，reduce 可以用来实现统计计算或时序依赖的遍历行为。

## lambda 表达式

FP 的理论基础，称之为 Lambda 模型。在很多 FP 语言中，Lambda 这个伟大的名字就代表了函式。而 Python 中，lambda 是一个关键字，是一个简单的匿名函式定义方式。它允许将一个表达式定义为一个可调用的对象，可以用它赋值或传递给其他函式。

前面讨论了一个通过 map 生成乘方表的例子，实际上那个例子中的 foo 函式只有一行，

完全可以用 lambda 代替它：

```
print map(lambda x:x**2, range(10))
~$ python pcs-114-6.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Python 中的 lambda 不像真正的 FP 语言那么显要，它只是一个语法糖，用来生成一个匿名的函式对象。Lambda 只接受一个单行表达式，该表达式的结果就是返回值。Python 的设计者出于可读性和语法的一致性考虑，没有支持多行匿名函式的定义，但是 Python 可以嵌套定义函式。

## 推导式

这项技术在 Python 中的正式命名是“List Comprehensions”，在 Python Tutorial 的简体中文版中译作列表推导式。其基本形式很简单，例如，前面生成平方表的例子还可以进一步简化为：

```
   print [x**2 for x in range(10)]
~$ python pcs-114-7.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

列表推导式分为三部分，最左边是生成每个元素的表达式，然后是 for 迭代过程，最右边可以设定一个 if 过判断作为过滤条件。例如通过下面的方式查找 1000 以内，所有左右对称的数：

```
    def isSymmetry(it):
        if it < 10:
            return True
        s = str(it)
        mpoint = len(s)/2 + 1
        for idx in range(1, mpoint):
            if s[idx-1] != s[-idx]:
                return False
        else:
            return True

    print [x for x in range(1000) if isSymmetry(x)]
~$ python pcs-114-8.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, 101, 111,
121, 131, 141, 151, 161, 171, 181, 191, 202, 212, 222, 232, 242, 252, 262,
272, 282, 292, 303, 313, 323, 333, 343, 353, 363, 373, 383, 393, 404, 414,
424, 434, 444, 454, 464, 474, 484, 494, 505, 515, 525, 535, 545, 555, 565,
575, 585, 595, 606, 616, 626, 636, 646, 656, 666, 676, 686, 696, 707, 717,
727, 737, 747, 757, 767, 777, 787, 797, 808, 818, 828, 838, 848, 858, 868,
878, 888, 898, 909, 919, 929, 939, 949, 959, 969, 979, 989, 999]
```

列表推导式甚至还可以一次性执行多个 for 迭代，这会生成它们的全排列，例如可以通过以下方法生成一个九九乘法表：

```
    for s in ["%s * %s = %s"%(x, y, x*y) for x in range(1, 10) for y in range(1,
10)]:
      print s
~$ python pcs-114-9.py
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
...
8 * 8 = 64
8 * 9 = 72
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
```

如果只希望输出不重复的那些，不妨尝试以下代码：

```
   for s in ["%s * %s = %s"%(x, y, x*y) for x in range(1, 10) for y in range(1,
10) if x<=y]:
      print s
~$ python pcs-114-10.py
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
...
8 * 8 = 64
```

```
8 * 9 = 72
9 * 9 = 81
```

因为列表推导式可以统一实现 map 和 filter，而 reduce 可以转成等价的 for，Python 的发明人 Gudio 曾经想取消 map、filter 和 reduce 这三个函式。虽然在广大使用者的呼声下这三个函式还是保留了下来，但是大家也一致认可，列表推导通常可以用更简洁的表达从线性容器中有选择地获取并计算元素，所以应该优先使用列表推导式。

因为列表推导式非常好用，所以在 Python 2.5 之后，它做了进一步的扩展，如果一个函式接受一个可迭代对象作为参数，那么可以给它传递一个不带中括号的推导式，在 Python Tuorial 中称其为“迭代推导式”，因为它不需要一次生成整个列表，只需要将可迭代对象传递给函式，如果推导式返回大量元素，这样做显然可以节省内存，提高速度。例如字典的构造函式可以接受一个可迭代对象，从中得到（key, value）元组序列来生成字典，在使用 Python 2.5 时经常可以见到形如以下的代码：

```
   d = dict(("%s * %s"%(x, y), x * y) for x in range(1, 10) for y in range(1,
10))
   for k, v in d.iteritems():
      print "%s = %s"%(k, v)
~$ python pcs-114-11.py
8 * 4 = 32
7 * 4 = 28
5 * 2 = 10
6 * 4 = 24
4 * 2 = 8
4 * 6 = 24
8 * 7 = 56
1 * 6 = 6
5 * 6 = 30
5 * 9 = 45
...
4 * 3 = 12
8 * 3 = 24
9 * 9 = 81
8 * 8 = 64
8 * 2 = 16
```