3.2 执行精确的浮点数运算¶

问题¶

你需要对浮点数执行精确的计算操作，并且不希望有任何小误差的出现。

解决方案¶

浮点数的一个普遍问题是它们并不能精确的表示十进制数。 并且，即使是最简单的数学运算也会产生小的误差，比如：

```
>>> a = 4.2
>>> b = 2.1
>>> a + b
6.300000000000001
>>> (a + b) == 6.3
False
>>>
```

这些错误是由底层CPU和IEEE 754标准通过自己的浮点单位去执行算术时的特征。 由于Python的浮点数据类型使用底层表示存储数据，因此你没办法去避免这样的误差。

如果你想更加精确(并能容忍一定的性能损耗)，你可以使用 `decimal` 模块：

```
>>> from decimal import Decimal
>>> a = Decimal('4.2')
>>> b = Decimal('2.1')
>>> a + b
Decimal('6.3')
>>> print(a + b)
6.3
>>> (a + b) == Decimal('6.3')
True
```

初看起来，上面的代码好像有点奇怪，比如我们用字符串来表示数字。 然而， `Decimal` 对象会像普通浮点数一样的工作(支持所有的常用数学运算)。 如果你打印它们或者在字符串格式化函数中使用它们，看起来跟普通数字没什么两样。

`decimal` 模块的一个主要特征是允许你控制计算的每一方面，包括数字位数和四舍五入运算。 为了这样做，你先得创建一个本地上下文并更改它的设置，比如：

```
>>> from decimal import localcontext
>>> a = Decimal('1.3')
>>> b = Decimal('1.7')
>>> print(a / b)
0.76470588235294117647058823529
>>> with localcontext() as ctx:
...     ctx.prec = 3
...     print(a / b)
...
0.765
>>> with localcontext() as ctx:
...     ctx.prec = 50
...     print(a / b)
...
0.76470588235294117647058823529411764705882352941176
>>>
```

讨论¶

`decimal` 模块实现了IBM的"通用小数运算规范"。不用说，有很多的配置选项这本书没有提到。

Python新手会倾向于使用 `decimal` 模块来处理浮点数的精确运算。 然而，先理解你的应用程序目的是非常重要的。 如

果你是在做科学计算或工程领域的计算、电脑绘图，或者是科学领域的大多数运算， 那么使用普通的浮点类型是比较普遍的做法。 其中一个原因是，在真实世界中很少会要求精确到普通浮点数能提供的17位精度。 因此，计算过程中的那么一点点的误差是被允许的。 第二点就是，原生的浮点数计算要快的多-有时候你在执行大量运算的时候速度也是非常重要的。

即便如此，你却不能完全忽略误差。数学家花了大量时间去研究各类算法，有些处理误差会比其他方法更好。 你也得注意下减法删除以及大数和小数的加分运算所带来的影响。比如：

```
>>> nums = [1.23e+18, 1, -1.23e+18]
>>> sum(nums) # Notice how 1 disappears
0.0
>>>
```

上面的错误可以利用 `math.fsum()` 所提供的更精确计算能力来解决：

```
>>> import math
>>> math.fsum(nums)
1.0
>>>
```

然而，对于其他的算法，你应该仔细研究它并理解它的误差产生来源。

总的来说， `decimal` 模块主要用在涉及到金融的领域。 在这类程序中，哪怕是一点小小的误差在计算过程中蔓延都是不允许的。 因此， `decimal` 模块为解决这类问题提供了方法。 当Python和数据库打交道的时候也通常会遇到 `Decimal` 对象，并且，通常也是在处理金融数据的时候。