3.1 数字的四舍五入¶

问题¶

你想对浮点数执行指定精度的舍入运算。

解决方案¶

对于简单的舍入运算，使用内置的 `round(value, ndigits)` 函数即可。比如：

```
>>> round(1.23, 1)
1.2
>>> round(1.27, 1)
1.3
>>> round(-1.27, 1)
-1.3
>>> round(1.25361,3)
1.254
>>>
```

当一个值刚好在两个边界的中间的时候， `round` 函数返回离它最近的偶数。 也就是说，对1.5或者2.5的舍入运算都会得到2。

传给 `round()` 函数的 `ndigits` 参数可以是负数，这种情况下， 舍入运算会作用在十位、百位、千位等上面。比如：

```
>>> a = 1627731
>>> round(a, -1)
1627730
>>> round(a, -2)
1627700
>>> round(a, -3)
1628000
>>>
```

讨论¶

不要将舍入和格式化输出搞混淆了。 如果你的目的只是简单的输出一定宽度的数，你不需要使用 `round()` 函数。 而仅仅只需要在格式化的时候指定精度即可。比如：

```
>>> x = 1.23456
>>> format(x, '0.2f')
'1.23'
>>> format(x, '0.3f')
'1.235'
>>> 'value is {:0.3f}'.format(x)
'value is 1.235'
>>>
```

同样，不要试着去舍入浮点值来”修正”表面上看起来正确的问题。比如，你可能倾向于这样做：

```
>>> a = 2.1
>>> b = 4.2
>>> c = a + b
>>> c
6.300000000000001
>>> c = round(c, 2) # "Fix" result (???)
>>> c
6.3
>>>
```

对于大多数使用到浮点的程序，没有必要也不推荐这样做。 尽管在计算的时候会有一点点小的误差，但是这些小的误差是能被理解与容忍的。 如果不能允许这样的小误差(比如涉及到金融领域)，那么就得考虑使用 `decimal` 模块了，下一节我们会详细讨论。