**你真的弄懂了Python中小数进位操作吗**

在日常的项目需求中，很多时候客户都有对小数点位数有一定的要求。那么使用Python来完成需求的时候，我们该怎么实现这些这个功能呢？都有哪些比较好的方法呢？我们来一起看看。（本人使用的Python是Python3版本）

1. **内置round方法**

Python的内置方法可以帮助我们保留指定位数的小数，该函数使用也很简单。

res\_1 = round(1.41401,3)  
res\_2 = round(-1.41401,3)

res\_3 = round(0.5)  
print(res\_1) *# 1.414*print(res\_2) *# -1.414*

print(res\_3) *# 0*

分别输出为1.414、-1.414和0。

实际上Python的round()函数可以接受两个参数round(value, ndigits)，第一个参数为实际操作数，第二个参数为实际保留几位，如果第二个参数不填，则默认保留到整数位。

1. **为什么不精确了呢？**

**细心的小伙伴已经发现了，在Python3中对0.5进行默认的四舍五入后变成了0，这是为什么呢？这实际涉及到Python中进位和取舍的问题。Python 3.X对于浮点数默认的是提供17位数字的精度。**

**这里想到一个问题，大家在Python3中尝试运行：**

print(0.1+0.1)  
print(0.1+0.1+0.1)

输出却为0.2、0.30000000000000004的疑问，这是为什么呢？

**要想弄清楚这个问题，要扩展的知识就比较多了，我们不再花费很多的篇幅来进行解释(如果有必要，也可以用一篇文章来介绍)，大家可以在计算机组成原理这门学科中找到答案。主要问题还是因为计算机只认识和处理和二进制的数值，在实际的计算中，所有的数据都必须为二进制，计算后又转换回来，就是在这个过程中造成数据的截断误差。感兴趣的小伙伴可以找一些资料研究一下原理**

在这里我们只需要知道在计算机中浮点数表示是遵循IEEE 754标准的，那么一个浮点数在计算机中转二进制后，再次呈现出来是怎样的：

from decimal import Decimal  
print(Decimal(0.125))  
print(**"{0:.30f}"**.format(0.125))  
print(Decimal(0.1))  
print(**"{0:.30f}"**.format(0.1))

输出为：

0.125

0.125000000000000000000000000000

0.1000000000000000055511151231257827021181583404541015625

0.100000000000000005551115123126

可以看出0.1在计算机中不能精确的表示，0.125可以。我们也可以简单的进行人工判断：如果：

其中X是被检查的数的小数部分，n1,n2,n3…nm是自然数。我们就任务这个数可以被精确的表示：比如:0.5=1/2^1、0.125=1/8=1/2^3因此他们都可以精确的表示，而0.1则不可以。

写到这里，我们以0.1为例说明其转化为二进制的过程，并说明为什么不能精确表示的原因：

0.1\*2 = 0.2 取整数部分为0；

0.2\*2 = 0.4 取整数部分为0

0.4\*2 = 0.8 取整数部分为0

0.8\*2 = 1.6 取整数部分为1

0.6\*2 = 1.2 取整数部分为1

0.2\*2 = 0.4 取整数部分为0

…

最后一次的乘法结果没有等于1，则不断进行下去：这样我们就可以取得其二进制表示：0.0001100110011001100110011…。可以看出其不能使用有限的数位来表示，因此其不能精确表示。再以0.125为例：

0.125\*2 = 0.25 取整数部分为0；

0.25\*2 = 0.5 取整数部分为0

0.5\*2 = 1 取整数部分为1

显然0.125的二进制表示为0.001，其可以被精确的表示。

如果一个小数的整数部分不为0，则将这个小数转换为二进制的时候，就多一些符号和整数部分的转换即可(这里不再赘述)

下面我这里给出一个Python代码，将一个十进制的数转换为二进制形式，并通过参数bit\_num控制保留的小数的位数。

def dec\_to\_bin(delnum,num\_bit):  
 def gen(x):  
 y = abs(x)  
 while y > 0:  
 yield y % 2  
 y = y >> 1  
 else:  
 if x == 0:  
 yield 0  
 *# 先判断是否是小数* dot\_exist = False  
 string\_delnum = str(delnum)  
 if **'.'** in string\_delnum:  
 dot\_exist = True  
 *# 存在小数的时候* if dot\_exist:  
 decimal\_convert = **''** integer\_convert = **''** int\_part, dec\_part = string\_delnum.split(**'.'**)  
 integer\_part = int(int\_part)  
 decimal\_part = Decimal(str(delnum)) - integer\_part  
 *# 处理整数部分* bin\_intpart = [i for i in gen(integer\_part)]  
 bin\_intpart.reverse()  
 if integer\_part >= 0:  
 integer\_convert = (**'%d'** \* len(bin\_intpart) % tuple(bin\_intpart))  
 else:  
 integer\_convert = (**'-'** + (**'%d'** \* len(bin\_intpart) % tuple(bin\_intpart)))  
 *# 处理小数部分* i = 0  
 *# while decimal\_part != 0 and i < num\_bit:* while i < num\_bit:  
 result = int(decimal\_part \* 2)  
 decimal\_part = decimal\_part \* 2 - result  
 decimal\_convert = decimal\_convert + str(abs(result))  
 i = i + 1  
 convert\_res = integer\_convert + **'.'** + decimal\_convert  
 return convert\_res  
 *# 只有整数部分的时候* else:  
 bin\_intpart = [i for i in gen(delnum)]  
 bin\_intpart.reverse()  
 if delnum >= 0:  
 convert\_res = (**'%d'** \* len(bin\_intpart) % tuple(bin\_intpart))  
 else:  
 convert\_res = (**'-'** + (**'%d'** \* len(bin\_intpart) % tuple(bin\_intpart)))  
 return convert\_res

print(dec\_to\_bin(-3.1,20)) *#-11.00011001100110011001*  
print(dec\_to\_bin(3.0,20)) *#11.00000000000000000000*  
print(dec\_to\_bin(3.1415,3))*#11.001*

1. **如何正确的进行四舍五入？**

我们使用decimal模块就能很好的解决这个问题，官方链接如下：

<https://docs.python.org/3/library/decimal.html#module-decimal>

decimal中有不同的进位方式，不同modes解释我们可以从以下界面获取：

<https://docs.python.org/3/library/decimal.html#decimal.ROUND_CEILING>

decimal可以通过指定rounding参数来确定进位方式。如果没有指定rounding参数，那么默认使用上下文提供的进位方式，上下文的进位方式我们可以这样来查看：

from decimal import getcontext  
print(getcontext().rounding)

输出ROUND\_HALF\_EVEN，即Round to nearest with ties going to nearest even integer方式进行进位。

我们我们要进行正确的四舍五入，我们将rounding指定为ROUND\_HALF\_UP即可。我们来看代码：

from decimal import Decimal  
from decimal import ROUND\_HALF\_UP,ROUND\_HALF\_EVEN  
num\_1 = Decimal(**'0.125'**).quantize(Decimal(**'0.00'**), rounding=ROUND\_HALF\_UP)  
num\_2 = str(Decimal(**'0.375'**).quantize(Decimal(**'0.00'**), rounding=ROUND\_HALF\_EVEN))  
print(num\_1)  
print(num\_2)  
print(type(num\_1))  
print(type(num\_2))

输出：

0.13

0.38

<class 'decimal.Decimal'>

<class 'str'>

代码解释：Decimal(**'0.00'**)指定保留两位小数的意思，rounding指定进位方式，注意第三行代码返回的是一个decimal.Decimal对象。ROUND\_HALF\_UP表示Round to nearest with ties going away from zero的进位方式，即真正的四舍五入。

注意，Decimal是可以接受浮点数为参数的，我们来看一下：

num\_3 = Decimal(11.245).quantize(Decimal(**'0.00'**), rounding=ROUND\_HALF\_UP)  
num\_4 = Decimal(**'11.245'**).quantize(Decimal(**'0.00'**), rounding=ROUND\_HALF\_UP)  
print(num\_3)  
print(num\_4)

输出：

11.24

11.25

是不是感觉很奇怪了？是这样的，官网有这样一段话：

“If value is a float, the binary floating point value is losslessly converted to its exact decimal equivalent. This conversion can often require 53 or more digits of precision. For example, Decimal(float('1.1')) converts to Decimal('1.100000000000000088817841970012523233890533447265625').”

也就是说，当我们传入一个不能精确的表示的数时，经Decimal后转换被转成了一个不精确的数了（有截断误差），此时可能不能进行正确的四舍五入了。

最后，我这里给出一个能将任意浮点数精确四舍五入的方法：

from decimal import Decimal,ROUND\_HALF\_UP  
def right\_round(num,keep\_n):  
 if isinstance(num,float):  
 num = str(num)  
 return Decimal(num).quantize((Decimal(**'0.'** + **'0'**\*keep\_n)),rounding=ROUND\_HALF\_UP)  
  
print(right\_round(1.245,2))

right\_round()接收两个参数，一个是转换的数，一个是位数即将这个数四舍五入保留几位小数。

1. **Pandas中的表示**

在Pandas也是可以对数据保留到指定小数的方法，我们来看一下：

import pandas as pd  
import numpy as np  
*# 设置精度表示*pd.set\_option(**'precision'**,2)  
test\_data = pd.Series([1.2345,11.275,11.245,11.2451])  
print(test\_data.round(2))  
print(np.round(test\_data, decimals=2))

两个输出结果均为：

0 1.23

1 11.28

2 11.24

3 11.25

dtype: float64

可以看出，pandas和numpy的进位方式和Python中原生的round()方式时一致的。

1. **总结**

以上就是本次推文Python中四舍五入保留小数的全部内容，在一些要求精度的项目中，希望本文能给大家带来帮助！