

RE: prePEP: Decimal data type

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2003-11/1168.html>

From: Tim Peters (tim.one_at_comcast.net)

Date: 11/06/03

Date: Thu, 6 Nov 2003 10:43:52 -0500

To: <python-list@python.org>

[Alex Martelli]

> *European Union directives (adopted as laws by member states) mandate*
> *the rounding procedure to be used in computations involving Euros*
> *(round to closest Eurocent, always round up on half-Eurocent*
> *results); they very explicitly mention that this may give a*
> *1-Eurocent discrepancy compared to "exact" arithmetic, and give*
> *examples; they establish that such a 1-cent discrepancy that comes*
> *from following exactly the prescribed rules is NOT legal cause for*
> *any lawsuit whatsoever; they earnestly recommend that all computing*
> *equipment and programs follow these same rules to avoid the huge*
> *headaches that would result in trying to reconcile accounts*
> *otherwise.*
>
> *Thus, for most accounting programs intended to run within the EU (not*
> *just in Euros: these provisions also apply to the other non-Euro*
> *currencies, as far as EU law is concerned), I do NOT think it would*
> *be a good thing for the programmer to have to remember to round*
> *explicitly -- the legal mandate is about rounding rules and it's*
> *quite easy to avoid the "fail to meet the requirements", as they seem*
> *designed to be easy to meet.*

I'd be astonished if the rules were consistent enough so that a type implementing a fixed number of decimal digits "after the decimal point" would be of real use for non-experts ... OK, here from:

<http://www.eubusiness.com/emu/retail8.htm>

The conversion of prices into euros will require the use of a six-significant-digit conversion rate (six digits disregarding initial zeros)

which should not be rounded or truncated. This rate will be irrevocably fixed on 1 January 1999 and defined in the form of one euro expressed in national currencies.

So that part mandates a *floating* point input (not meaning binary floating point, but "6 digits disregarding initial zeros" is the essence of floating

point — the total number of digits isn't fixed, nor is the # of digits after the decimal point fixed, just the # of *significant* digits).

To convert from national currencies to the euro, one has to divide by the conversion rate. To convert from the euro to the national currency, one has to multiply by the conversion rate. The use of inverse rates is forbidden.

Neutral.

To convert from one national currency to another, amounts must be first converted into euros and then into the second national currency. The euro amount must be rounded to three decimal places. The national currency should then be rounded to two decimals.

So no single fixed-point discipline can suffice: in one direction they want rounding to 3 digits after the decimal point, in the other to 2 digits, and one of the inputs is a floating-point value with no fixed number of digits after the decimal point.

This is all very easily done with IBM's proposed arithmetic, but it requires the programmer to specify explicitly how many places after the decimal point they want after each * and / operation. Since that value changes in arbitrarily (from the POV of arithmetic semantics) mandated ways, no class can guess what's required from step to step.

Again, this isn't an issue for + or -: if x and y have the same number of digits after the decimal point, then x+y and x-y are exact, and inherit that same number of fractional digits (provided only the user hasn't specified a suicidally low precision — and they can easily know whether they have, because the "rounded result" flag would get set in the context).

> *Whether Decimal itself allows an optional rounding-policy (including
> of course "no rounding" as a possibility) is one issue (I guess that
> might violate some ANSI or IEEE standard...?)*

The IBM spec requires that the user be able to specify rounding mode, in the context. It sounds like the "round half up" mode is what the EU requires (not from the link above, but from what you said). Note that Cowlshaw's site has a "telco benchmark" wherein different steps require different rounding disciplines; I don't think that's uncommon either; I was able to get the exact result for that benchmark pretty easily using my FixedPoint class, but I enjoyed the luxury of having expert knowledge of the potential traps in advance. I expect most FixedPoint users would struggle with it.

> *but I most surely do want to be able to use such policies in whatever
> arithmetic type underlies Money -- so I hope Decimal is at least
> designed so that _subclasses_ can easily provide such customized
> rounding (e.g., feature-testing for a __round__ specialmethod, not
> defined in the baseclass Decimal if need be, but offering the needed
> hook for subclasses to add the functionality).*

It has to support some half dozen specific rounding modes already (the spec requires them). The mandated modes have been reviewed by many people now over several years, so it's tempting to call YAGNI on adding more rounding complexity. My FixedPoint does support user-defined rounding modes; alas, all evidence to date suggests I'm the only person to date able to define one correctly <wink -- but doing rounding exactly right requires exact understanding of every stinking detail of every stinking endcase -- it's not easy the first time someone tries it>.