A Numeric Mystery

One way to simplify a decimal number x (*e.g.* 23.456) is to discard all but n digits (*e.g.* 3 digits), keeping only the most significant nonzero digit and the following n-1 digits (*e.g.* 23.456 simplifies that way to 23.5). As a text-editing task, this is quite easy to do. As a numeric task it's trickier.

Here's a way to do it numerically, without ever seeing the text form of the numbers: Start by converting x to a "normalized" form, whose highest non-zero digit occurs immediately after the decimal point (*e.g* 23.456 normalizes to 0.23456). To accomplish this — without looking — we can use the number ilx computed as follows:

Start by creating the normalized form of x:

1. Compute the integer part ilx of $\log_{10}(x)$ (*e.g.* ilx = int($\log_{10}$(23.456)) = int(1.37025) = 1)
2. If ilx >= 0 then add 1 to it (*e.g.* ilx becomes 2)
3. Raise 10 to the power ilx (*e.g.* $10^2 = 100$)
4. Form norm(x) by dividing x by 10^{ilx} (*e.g.* x ÷ 10^{ilx} = 23.456 ÷ 100 = 0.23456 = norm(x))

Unix's rounding function rounds its argument to the nearest integer, so next we'll shift the desired n digits of x into the integer position:

5. Raise 10 to the nth power (*e.g.* 10^3 = 1,000)
6. Multiply Norm(x) by 10^n to isolate the desired n digits of x in the integer part (*e.g.* norm(x)· 10^n = 0.23456 · 1,000 = 234.56)
7. Round that result to the nearest integer (*e.g.* round(234.56) = 235.)

Next, normalize that rounded version then shift it to its original position:

8. Divide by 10^n to form the normalized result (*e.g.* 235. ÷ 10^n = 0.235)
9. Finally, multiply by 10^{ils} to undo the normalization (e.g. 0.235 · 10^{ilx} = 0.235 · 10^2 = 23.5 *voila*! n digits!)

This algorithm works pretty well:

x	n	n digits	log(x)
23.4567	3	23.5	1.3702669
9.019019	3	9.02	0.95515930
2.34567	3	2.35	0.37026691
1.108768	3	1.11	0.044840683
0.9019019	3	0.9	-0.044840698
0.234567	3	0.23	-0.62973309
0.0234567	3	0.0235	-1.6297331
0.00234567	3	0.00235	-2.6297331

But, oops: The 3-digit form of 0.234567 should be 0.235, and the 3-digit form of 0.9019019 should be 0.902. Hence, the mystery: When and why does the algorithm fail? Which steps need to be changed?

A correct Frrr version of the algorithm

Keep n digits of x starting with leading nonzero digit
(for decimal x and 1 <= n <= 8)
Setup: F1 = n+x/0
Result: n-digit-form in F2num

Make sure x is decimal, not an integer:

```
@< {{ SkipWhen F1num isDecimal }}>

@< {{ hp( put/- NOT DECIMAL-/ h1n
   PUT/- F1num must be decimal, not integer -/ hShow ) @}}>
```

Save the desired precision in s1 as 10-to-the-n:

```
@<{{ hp( get f1i
   00c __exp10 sto 1 ) }}>
```

1. Compute lx = log(x) into s2

```
@<{{ hp( get f1n
   00c log10 sto 2 )
   SkipWhen <s>2 LT0 }}>
```

2. If log x >= 0 modify it by adding 1 to it:

```
@<{{ hp( rcl 2 1 00+ dupx sto 2 ) }}>
```

3. Raise 10 to the power ilx = int(s2)

```
@<{{ hp( rcl 2 00c trunc ooc __exp10 sto 2  ) }}>
```

4. Normalize x by dividing it by 10 to the power ilx

```
@<{{ hp( get f1n rcl 2 00/ ) }}>
```

5,6. Isolate the desired n digits of x as an integer

```
@<{{ hp( rcl 1 00* sto 3 ) }}>
```

7,8. Round, then normalize that result into s3

```
@<{{ hp( rcl 3  00c round  rcl 1 00/ sto 3 ) }}
```

9. Undo the original normalization

```
@<{{ hp( rcl 3 rcl 2 00* sto 3 ) }}
```

Display the result in F2num, then repeat from the start

```
@<{{ hp( rcl 3  00put f2n ) @}}
```