What is wrong with the FORMAT BIF?

Walter Pachl, 5 May 2025

Some, well a long time ago I came across a difference in the outputs of a Rosetta Code task when run with ooRexx and Regina. I investigated a little and the reason was that the FORMAT bif gave different results for these two Interpreters. Recently I came up with the idea to systematically explore what the problems with FORMAT are and to present the results.
Finally I will show what I would consider a correct implementation of this versatile function.

FORMAT(number) is to return the result of the expression (number+0)

Let’s see what happens:

```
Parse arg d f
parse Version v
If d<>'' Then Numeric Digits d
If f<>'' Then Numeric Form Engineering
Say v
Say 'x                  (x+0)                format(x)                  (x/1)'
x=1.00000000;          Say left(x,18) left((x+0),20) left(format(x),26) (x/1)
x=1.234567895;         Say left(x,18) left((x+0),20) left(format(x),26) (x/1)
x=1.234567895e-8;      Say left(x,18) left((x+0),20) left(format(x),26) (x/1)
x=0.00000001234567895; Say left(x,18) left((x+0),20) left(format(x),26) (x/1)

F:\_0505>regina f1
REXX-Regina_3.9.6(MT) 5.00 29 Apr 2024
x                  (x+0)                format(x)           (x/1)
1.00000000         1.00000000           1.00000000           1
1.234567895        1.23456790           1.23456790           1.2345679
1.234567895E-8     1.23456790E-8        1.23456790E-8        1.2345679E-8
0.0000000123456789 1.23456790E-8        1.23456790E-8        1.2345679E-8

F:\_0505>rexx f1
REXX-ooRexx_5.1.0(MT)_64-bit 6.05 7 Feb 2023
x                  (x+0)                format(x)             (x/1)
1.00000000         1.00000000           1.00000000             1
1.234567895        1.23456790           1.23456790             1.2345679
1.234567895E-8     0.0000000123456790   0.00000001234567905000 0.000000012345679
0.0000000123456789 0.0000000123456790   0.00000001234567905000 0.000000012345679
```

(a) x+0 keeps zeroes in the decimal places, x/1 drops them
(b) Regina uses exponential notation when it shouldnt‘t
(c) ooRexx obeys the rule that exponential notation ist to be used only when the length oft the decimal fraction exceeds twice the value of digits(). Its result for FORMAT is, however, simply wrong.

The full definition of FORMAT is as follows (from the ANSI Standard)

9.4.2 FORMAT
FORMAT formats its first argument (number)
The second argument (before) specifies the number of characters to
be used for the integer part
The third argument (after) specifies the number of characters for the decimal part.
The fourth argument (expp) specifies the number of characters for the exponent (p for positions)
The fifth argument (expt) determines when exponential notation is to be used. (t for trigger)

There is a discrepancy between before and expp. It’s the number of characters and the number of digits, respectively:

```
Say '                          1234.123456E+1234'
say 'format(-43.78e33,4,6,4) ->'format(-43.78e33,4,6,4)

                          1234.123456E+1234
format(-43.78e33,4,6,4) ->  -4.378000E+0034
```

Netrexx doesn‘t like the number as written above (-43.78E+33 is accepted), neither does it accept omission of expt. A modified program shows

```
                            1234.123456E+1234   scale
format(-43.78e+33,4,6,4,4)->  -4.378000E+0034
format(-43.78e+33,4,6,4,0)->  -4.378000E+0034
      (-43.78e+33)+0      ->-4.378E+34
format(-43.78e+33)        ->-43780000000000000000000000000000000
      (-43.78e-33)+0      ->-4.378E-32
format(-43.78e-33)        ->-0.00000000000000000000000000000004378
```

And exptt=0 should force exponential notation. Well, there is an exception When Numeric Form Engineering is in effect:

```
Numeric Form Engineering
Do e=-1 To 3
  x=3.456'E'||e
  Say format(x,,,,0)
  End

345.6E-3
3.456
34.56
345.6
3.456E+3
```

The good news: Regina and ooRexx give the same results.

I wrote a program to translate the code shown in the standard to an executable REXX program. In order to use that as an external function I added two parameters (d and f) denoting the Numeric Digits and Numeric Format settings to be used – the defaults being 9 and SCIENTIFIC)-

Parse Arg Number,Before,After,Expp,Expt,d,f

The first test program using this function (ansi0.rex) shows

```
     X                        ->     ansi0(X,4,9,2,0)    format(X,4,9,2,0)
   1 0.000000000123456789    ->     0.000000000           1.234567890E-10
   2 0.00000000123456789     ->     0.000000001           1.234567890E-09
   3 0.0000000123456789      ->     0.000000012           1.234567890E-08
   4 0.000000123456789       ->     0.000000123           1.234567890E-07
   5 0.00000123456789        ->     1.235                 1.234567890E-06
   6 0.0000123456789         ->     1.2346                1.234567890E-05
   7 0.000123456789          ->     1.23457               1.234567890E-04
   8 0.00123456789           ->     1.234568              1.234567890E-03
   9 0.0123456789            ->     1.2345679             1.234567890E-02
  10 0.123456789             ->     0.123456789           1.234567890E-01
  11 1.23456789              ->     1.234567890           1.234567890
  12 12.3456789              ->     1.234567890E+01       1.234567890E+01
```

Note the incorrect results beginning with line 5.

The reason ist this piece of code

Integer = left(Integer, Point + After)
if r >= '5' then Integer = Integer + 1

Where the leading zeroes of the integer are lost.

Adding them with this added line as follows

integer=right(integer,Point + After,0)   /* fix1 correct the rounding bug */

fixes this problem:

```
     X                        ->     ansi1(X,4,9,2,0)    format(X,4,9,2,0)
   1 0.000000000123456789    ->     0.000000000           1.234567890E-10
   2 0.00000000123456789     ->     0.000000001           1.234567890E-09
   3 0.0000000123456789      ->     0.000000012           1.234567890E-08
   4 0.000000123456789       ->     0.000000123           1.234567890E-07
   5 0.00000123456789        ->     0.000001235           1.234567890E-06
   6 0.0000123456789         ->     0.000012346           1.234567890E-05
   7 0.000123456789          ->     0.000123457           1.234567890E-04
   8 0.00123456789           ->     0.001234568           1.234567890E-03
   9 0.0123456789            ->     0.012345679           1.234567890E-02
  10 0.123456789             ->     0.123456789           1.234567890E-01
```

```
11 1.23456789             ->    1.234567890        1.234567890
12 12.3456789             ->    1.234567890E+01    1.234567890E+01
```

Note that the result shows the exponential notation for a positive exponent But fails to do so for negative exponents.

Investigations led me to this piece of code

```
pd=verify(Integer,0)
expo=pd-1
ilen=length(integer)
If expt='' Then expt=digits()
if pd>=expt*2+2 Then Do      /* fix2: 20250313 neg. exponent */
  number=substr(integer,pd,1)'.'||,
            substr(integer,pd+1)'E-'expo
  Return mkres(number,expo)
  End
```

where I implemented the exponent rule described above.
The function mkres takes care of Numeric Form Engineering, if applicable and the value of expp

```
mkres:
  Parse Arg number
  parse Var number number 'E' exponent
  shift=0
  if form() == 'ENGINEERING' then Do
    shift=0
    do while Exponent//3<>0
      shift=shift+1
      Exponent = Exponent-1
      end
    end
  Parse Var number int '.' dec
  lpart=int||left(dec,shift)
  If sign Then lpart='-'lpart
  rpart=substr(dec,shift+1)
  if before<>'' Then
    lpart=right(lpart,before)
  If after<>'' Then
    rpart=left(rpart,after,0)
  If sign(exponent)>0 Then
    expsign='+'
  Else
    expsign='-'
  If expp<>'' Then
    Exponent=right(abs(exponent),expp,0)
  Else
    Exponent=abs(exponent)
  result=lpart'.'rpart'E'expsign||exponent
  Return Result
```

With these additions we come to this result which looks really good.

```
   X                      ->    ansi2(X,4,9,2,0)   format(X,4,9,2,0)
 1 0.000000000123456789   ->    1.234567890E-10      1.234567890E-10
 2 0.00000000123456789    ->    1.234567890E-09      1.234567890E-09
 3 0.0000000123456789    ->    1.234567890E-08      1.234567890E-08
 4 0.000000123456789     ->    1.234567890E-07      1.234567890E-07
 5 0.00000123456789      ->    1.234567890E-06      1.234567890E-06
 6 0.0000123456789       ->    1.234567890E-05      1.234567890E-05
 7 0.000123456789        ->    1.234567890E-04      1.234567890E-04
 8 0.00123456789         ->    1.234567890E-03      1.234567890E-03
 9 0.0123456789          ->    1.234567890E-02      1.234567890E-02
10 0.123456789           ->    1.234567890E-01      1.234567890E-01
11 1.23456789            ->    1.234567890          1.234567890
12 12.3456789            ->    1.234567890E+01      1.234567890E+01
```

Next I found by accident a problem with the exponent

```
F:\_0505>rexx test3.rex
n=0000011123456789.0123456789
format(n) =1.11234568E+10
format(n,,,,3) =1.11234568E+10

 ansi0(n,,,,,9,s)=1.11234568E++10
 ansi0(n,,,,,9,e)=11.1234568E++9

 ansi3(n,,,,,9,s)=1.11234568E+10
 ansi3(n,,,,,9,e)=11.1234568E+9

 ansi4(n,,,,,9,s)=1.11234568E+10
 ansi4(n,,,,,9,e)=1.11234568E+9E+1

 ansi5(n,,,,,9,s)=1.11234568E+10
 ansi5(n,,,,,9,e)=11.1234568E+9
```

Note the ++ with ansi0 and the crazy regression with ansi4.
Things get, however, worse – this time with Regina:

```
F:\_0505>regina test3
n=0000011123456789.0123456789
format(n) =1.11234568E+10
format(n,,,,3) =1.1123456789012345678 9E+10
```

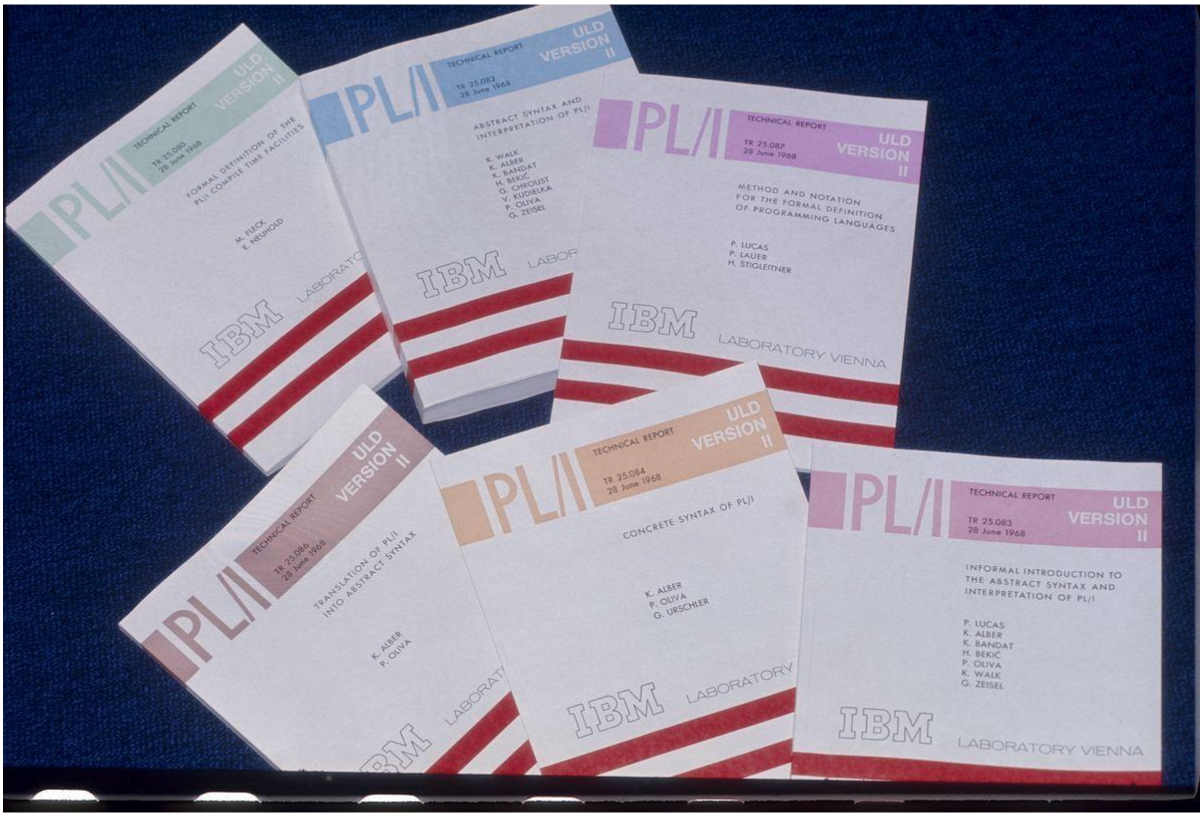
Next I tried variations of expp and expt;

```
F:\_0505>rexx test4
ansi3(-123e-33,,,,0,9,S) -> -1.23E-31
ansi3(-123e-33,,,1,8,9,S)-> -1.23E-1      <<<< WRONG
ansi3(-123e-33,,,3,0,9,S)-> -1.23E-031
ansi4(-123e-33,,,,0,9,S) -> -1.23E-31
ansi4(-123e-33,,,1,8,9,S)-> argument expp=1 is not large enough to format -31
ansi4(-123e-33,,,2,8,9,S)-> -1.23E-31
ansi4(-123e-33,,,3,0,9,S)-> -1.23E-031
format(-123e-33,,,4,0)   -> -1.23E-0031
format(-123e-33,,,3,1)   -> -1.23E-031
format(-123e-33,,,2,1)   -> -1.23E-31
format(-123e-33,,,1,1)   -> Error 93.941:  Exponent of "-1.23" is too large for 1 spaces.
    Regina -> Error 40.38: FORMAT argument 4 is not large enough to format "-1.23E-31"
format(-123e-33,,,0,0)   -> -0.0000000000000000000000000000001230000000000000█áú,»00└0000000R
```

Note the improvement from ANSI3 to ANSI4
And the massive problem with ooRexx!!

Up to now I have shown four problems with the documentation of FORMAT in the ANSI Standard.
AND the first bug probably tob e reported for the brand new ooRexx 5.1

This reminds me of my early days wit IBM when the Vienna Lab worked on a „Formal Definition of PL/I“ called ULD (Universal Language Definition)



I wonder how many errors were and still are in these thousand or so pages.

Now a few comparisons of the Rexx Interpreters I have access to.

```
/* REXX    taf.rex   */
expt=0
X=0.0000000001234567890123456789
SAY '   X                     ->    format(X,4,9,2,'exPT')'
DO I=1 TO 12
  SAY right(I,2) LEFT(X,20)'  ->' LEFT(format(X,4,9,2,exPT),20)
  X=X*10
  END
```

```
                               ansi4
                               Regina
                               ooRexx              Netrexx
                               VM                  CRX
   X                     ->    FORMAT(X,4,9,2,0)   FORMAT(X,4,9,2,0)
 1 0.000000000123456789  ->    1.234567890E-10     1.234567890E-10
 2 0.00000000123456789   ->    1.234567890E-09     1.234567890E-09
 3 0.0000000123456789    ->    1.234567890E-08     1.234567890E-08
 4 0.000000123456789     ->    1.234567890E-07     1.234567890E-07
 5 0.00000123456789      ->    1.234567890E-06     0.000001235
 6 0.0000123456789       ->    1.234567890E-05     0.000012346
 7 0.000123456789        ->    1.234567890E-04     0.000123457
 8 0.00123456789         ->    1.234567890E-03     0.001234568
 9 0.0123456789          ->    1.234567890E-02     0.012345679
10 0.123456789           ->    1.234567890E-01     0.123456789
11 1.23456789            ->    1.234567890         1.234567890
12 12.3456789            ->    1.234567890E+01     1.234567890E+01
```

```
                                                  Netrexx
                               ooRexx             Regina
exec taf3                      VM                 CRX
   X                     ->    FORMAT(X,4,9,2,3)  FORMAT(X,4,9,2,3)  ansi4(X,4,9,2,3)
 1 0.000000000123456789  ->    1.234567890E-10     1.234567890E-10   1.234567890E-10
 2 0.00000000123456789   ->    1.234567890E-09     1.234567890E-09   1.234567890E-09
 3 0.0000000123456789    ->    1.234567890E-08     1.234567890E-08   1.234567890E-08
 4 0.000000123456789     ->    1.234567890E-07     1.234567890E-07   1.234567890E-07
 5 0.00000123456789      ->    1.234567890E-06     0.000001235       1.234567890E-06
 6 0.0000123456789       ->    1.234567890E-05     0.000012346       1.234567890E-05
 7 0.000123456789        ->    1.234567890E-04     0.000123457       0.000123457
 8 0.00123456789         ->    1.234567890E-03     0.001234568       0.001234568
 9 0.0123456789          ->    1.234567890E-02     0.012345679       0.012345679
10 0.123456789           ->    1.234567890E-01     0.123456789       0.123456789
11 1.23456789            ->    1.234567890         1.234567890       1.234567890
12 12.3456789            ->    1.234567890E+01    12.345678900       1.234567890E+01
```

ooRexx and VM don’t look correct: they ignore expt=3. Why ANSI4 differs from Regina et al remains to be analyzed but now it’s May 5, 2025 and I have spent too much time with FORMAT already ☹️

Some time ago I experimented with a FORMAT function that would produce results I am totally happy with. Here is what I came up to when using myformat(x,4,4)
(It does NOT consider any of the other parameters)

```
F:\_0505>regina myf
REXX-Regina_3.9.6(MT) 5.00 29 Apr 2024
x                                           format(x,4,4)  ansi3(x,4,4)  myf(u,v,4,4)
0.000000000001234567890123456789000000        1.2346E-12    1.2346E-12    1.2345E-12
0.00000000001234567890123456789000000        12.3457E-12   12.3457E-12   12.3456E-12
0.0000000001234567890123456789000000        123.4568E-12  123.4568E-12  123.4567E-12
0.000000001234567890123456789000000           1.2346E-9     1.2346E-9     1.2345E-9
0.00000001234567890123456789000000           12.3457E-9    12.3457E-9    12.3456E-9
0.0000001234567890123456789000000           123.4568E-9   123.4568E-9   123.4567E-9
0.000001234567890123456789000000              0.0000        0.0000        1.2345E-6
0.00001234567890123456789000000               0.0000        0.0000       12.3456E-6
0.0001234567890123456789000000                0.0001        0.0001        0.0001
0.001234567890123456789000000                 0.0012        0.0012        0.0012
0.01234567890123456789000000                  0.0123        0.0123        0.0123
0.1234567890123456789000000                   0.1235        0.1235        0.1235
1.234567890123456789000000                    1.2346        1.2346        1.2346
12.34567890123456789000000                   12.3457       12.3457       12.3457
123.4567890123456789000000                  123.4568      123.4568      123.4568
1234.567890123456789000000                 1234.5679     1234.5679     1234.5679
12345.67890123456789000000                 -             -               12.3456E+3
123456.7890123456789000000                 -             -              123.4567E+3
1234567.890123456789000000                 -             -                1.2345E+6
12345678.90123456789000000                 -             -               12.3456E+6
123456789.0123456789000000                 -             -              123.4567E+6
1234567890.123456789000000                 -             -                1.2345E+9
12345678901.23456789000000                 -             -               12.3456E+9
123456789012.3456789000000                 -             -              123.4567E+9
1234567890123.456789000000                 -             -                1.2345E+12
```

The features are that the decimal part of the formatted number is never 0
And if the integer part is too large for „before“ positions I escape to exponential notation.

Coda: complex.cls

803 extensions to complex.cls 5.0.0 accepted Erich Walter 2022-06-02 2022-06-03 tests 0

```
A test case would be
Parse Arg r i
If r='' Then Parse Value '-3 1' With r i
z=.complex~new(r,i)
w2=z~sqrt
Say 'Z='z
Say 'sqrt(z)='w2
say 'w2**2='||(w2**2)
::REQUIRES complex0.cls
```

Originally I posted a file complex.cls showing the following result:

```
F:\complex>rexx tc0 -3 1
Z=-3+i
sqrt(z)=0.284848788+1.75531730i
w2**2=-2.99999999+1.00000001i
```

Later I posted an "improved" version and did not notice this problem

```
F:\complex>rexx tc1 -3 1
Z=-3+i
sqrt(z)=1.03977826+1.44261528i
w2**2=-1.00000002+3.00000002i
```

When Rony asked me to provide a test case, I was surprised to find a much simpler implementation of sqrt, probably provided by Erich Steinböck.

```
F:\complex>rexx tcoo
Z=-3+i
sqrt(z)=0.284848784+1.75531730i
w2**2=-2.99999999+i
```

```
::method sqrt                      /* complex square root            */
  expose real imaginary
  use strict arg

  sqrtx2y2 = sqrt(real ** 2 + imaginary ** 2)
  sign = (imaginary < 0)~?(-1, 1)

  return self~class~new( -
    sqrt((sqrtx2y2 + real) / 2), -
    sqrt((sqrtx2y2 - real) / 2) * sign)
```

Now I went back to my correct routine and extended it do compute any root
Here the seventeenth

```
F:\complex>rexx  tc3
Z=-3+i
sqrt(z)=0.284848788+1.75531730i
w2**2=-2.99999999+1.00000001i
z~root(7)=1.08441228+0.462109842i
w7**7=-3.00000003+0.99999995i
```

Just for fun I added a test using Paul van den Eertwegh's Mathematics
(he represents complex numbers as pairs of real numbers

```
Parse Arg r i
If r='' Then Parse Value '-3 1' With r i
Say r i
w2=csqrt(r i)
say w2
say cmul(w2,w2)
exit
include Math

D:\complex>rexx run tce.rex
-3 1
2.848487844458E-1 1.7553173018
-3.000000000 0.9999999994
```

Finally, to get compaxx.rex (my best program ever) go to

www.wpachl.at

and find it under Rexx Programs.