Rosetta Diamonds
Walter Pachl
RexxLA Symposium Barcelona, May 2026

This will be the next excursion to the world of rosetta code.
A Dutch colleague (Paul van den Eertwegh) retired after a long software career using multiple languages in 2023.
He pursued then programming as a hobby. He came across rosetta code and on his user page https://rosettacode.org/wiki/User:Zeddicus
you can see why Rexx was the language of choice. He was, however, annoyed by the need to copy/paste functions that are needed by programs to solve miscellaneous tasks.
So he invented a method to provide these functions in files to be included in the REXX (Regina) programs proper. Later I drew his attention to ooRexx and he developed a method to generate from his files a CLS package to be used by ooRexx programs.
I served as the devil's advocate by testing his work and making several suggestions for corrections and improving the usability of these diamonds found on rosettacode.org. I asked him whether he would present his work on this symposium. He did not want to do this so I suggested that I present it from a user's point of view.

First Impressions: Polynoms

```
/* Show use of polynom.inc */
u=3 4 1 2
v=1 2 3
Say 'u='poly2form(u)
Say 'v='poly2form(v)
Parse Value DivP(u,v) With q','r  -- get quotient and remainder
Say 'q='poly2form(q)
Say 'r='poly2form(r)
t=AddP(MulP(q,v),r)               -- t=q*v+r
Say 't='poly2form(t)
If eqp(u,t) Then Say 'Test is ok'
Say 'evalp(u,2)='evalp(u,2)       -- evaluate u(2)
Exit
include polynom
```

Output:

```
F:\_paul>rexx run pol
u=3x^3+4x^2+x+2
v=x^2+2x+3
q=3x-2
r=-4x+8
t=3x^3+4x^2+x+2
Test is ok
evalp(u,2)=44
```

with v=0 2 3 in the above program:

```
F:\_paul>rexx run pol0
u=3x^3+4x^2+x+2
v=+2x+3
   114 *-*     q=Word(rx,1)/Word(dx,1);
     6 *-* Parse Value DivP(u,v) With q','r  -- get quotient and remainder
Error 42 running F:\_paul\Runner.rex line 114:  Arithmetic
overflow/underflow.
Error 42.3:  Arithmetic overflow; divisor must not be zero.
```

So I suggested to provide and use a function to remove the leading zeros from the polynom.

Meanwhile implemented as SimpP (simplify polynom) and extended to others (e.g., SimpC – simplify complex)

Run.rex is a program that produces a file runner.rex by expanding the include files in the given source.

The suggested framework for a program using math services is

```
Include setting
Your program
Exit
Include math
```

When using ooRexx, it's a bit simpler:

```
Your program
::REQUIRES math
```

Setting.inc establishes abend as the target for all conditions (including Novalue, which is, of course, also enabled here) and initializes some global stems.

Math.inc is a file that conatains ALL the individual inc files in which functions are grouped by their purpose such as those In polynomial.inc:

```
Polynom.inc       1
AddP         -- Returns sum of two polynoms
CubeP        -- Returns cube of a polynom
DifP         -- Returns first derivative
DivP         -- Returns quotient and remainder of 2 polynoms
EqP          -- Are 2 polynoms equal?
EvalP        -- Returns the value for the given number
GetP         -- Gets the degree of the given polynom
MulP         -- Returns product of 2 polynoms
NeP          -- Are 2 polynoms not equal?
NegP         -- Returns negative
Poly2form    -- Converts polynom to a formula
Poly2stem    -- Converts polynom to stem
Polynomi     -- Is a polynom valid?
PowP         -- Returns integer exponentiation for given power
ReverseP     -- Reverts the coefficents in opposite order
ScaleP       -- Scales by the given number
SquareP      -- Returns square of a polynom
Stem2poly    -- Converts stem to polynom
SubP         -- Returns the difference between 2 polynoms
ZeroP        -- Is a polynom zero?
```

In polynom.inc (and math.inc) you can see a short description and the signature of each function, e.g.,

```
DivP:
-- Returns quotient and remainder of 2 polynoms
-- polynom,polynom -> polynom quotient,polynom remainder
```

The „types“ of the variables are listed at the beginning of math.inc:

```
...
-- number     any number
-- polynom    structure of n numbers coefs from high to low
-- quaternion structure of 4 numbers 'real i j k'
-- recipe     formula for continued fraction
-- sign       -1, 0 or 1
-- stem       compound variable
-- string     series of characters
-- structure  string of blank separated numbers
```

and many others.

Algebraic Equations

One use of polynoms ist the solving of algebraic equations.
Math solves them up to the 4th degree.

```
Cbeq    -- Returns real roots of ax^3+bx^2+cx+d=0
CbeqC   -- Returns complex roots of ax^3+bx^2+cx+d=0
Lieq    -- Returns real root of ax+b=0
Qteq    -- Returns real roots of ax^4+bx^3+cx^2+dx+e=0
QteqC   -- Returns complex roots of ax^4+bx^3+cx^2+dx+e=0
Sqeq    -- Returns real roots of ax^2+bx+c=0
SqeqC   -- Returns complex roots of ax^2+bx+c=0
```

The real (or also complex) solutions are returned in stem ROOT.i

```
/* eq.rex */
include setting
p='1 0 0 0 -16'
n=qteqc(p)
Say 'Solutions of' poly2form(p)'='0
Do i=0 to n
  Say 'x.'i'=' root.i
  End
Exit
include math
```

Regina lets you access the exposed stem ROOT. that is used in the function qteqc. The variables ROOT.x that are used in the method qteqc must, however, be retrieved by the function getst as shown below.

```
/* eqo.rex */
p='1 0 0 0 -16'
n=qteqc(p)
Say 'Solutions of' poly2form(p)'='0
Do i=0 to n
  Say 'x.'i'=' getst('ROOT.',i)
  End
::REQUIRES math

rexx run eq and rexx eqo show this result
Solutions of x^4-16=0
x.0= 4
x.1= 0 -2
x.2= 0 2
x.3= -2
x.4= 2
```

Lists

A list is a string of items, separated by semicolons. List.inc provides 21 functions. Three of them are shown here.

```
/* show use of list.inc */
include setting
Say 'Show minimum and maximum elements of the given list'
list='8e-3;1;1e5;33a'
say list2form(list) minl(list) '-' maxl(list)
list='8e-3;1;1e5;33'
say list2form(list) minl(list) '-' maxl(list)
Exit
include math

F:\_paul>rexx run maxl
Show minimum and maximum elements of the given list
{8E-3;1;1E5;33A} 1 - 8e-3
{8E-3;1;1E5;33} 8e-3 - 1e5
```

Note that the result depends on the fact whether all elements of the list are numbers or not!

When testing these functions I discovered a problem with Regina:

```
/* bug.rex */
Parse Version v
Say v
u='00'x
v=' '
Select
  When u<v Then op='<'
  When u>v Then op='>'
  Otherwise     op='='
  End
Say c2x(u) op c2x(v)

F:\_paul>rexx bug
REXX-ooRexx_5.2.0(MT)_64-bit 6.05 21 Nov 2025
00 < 20

F:\_paul>regina bug
REXX-Regina_3.9.6(MT) 5.00 29 Apr 2024
00 > 20
```

***After some mails, Mark Hessling conceded that this is a* bug.**

How do you get this Package?

On this page of rosettacode.org https://rosettacode.org/wiki/Mathematics.rex you'll find the information how to install math in a directory of your choice:

You may copy-paste-save-run below Extractor code, but that's somewhat unwieldy (it's 20k lines, the Wiki editor becomes slower for such large files). Better is to copy-paste-save below Installer. It's 'copy once, run often' and automates the manual steps: scrapes Extractor.rex from the web, saves it and runs it. No need to visit RC, but it requires installation of the well-known and famous freeware Wget program.

```
Z:\_paul>rexx installer ?
-- Module Installer.rex - 2 Mar 2026
-- REXX Mathematical Toolbox Installer

-- REXX
-- Regina (Classic mode) Version 3.9.0 (2014) or higher
-- ooRexx (Classic mode) Version 5.0.0 (2023) or higher
-- ooRexx (Object mode)  Version 5.2.0beta (2025) or higher

-- WGET
-- Install Wget (C)1996-2023 if not yet present:
-- MacOS   via MacPorts / Homebrew
-- Windows via dozens of websites (google wget.exe)
-- Unix    built-in

-- INSTALL
-- Run this program in your REXX folder, using Regina or ooRexx.
-- Assuming Wget, Regina and/or ooRexx are in your path.
-- Classic include files and an Object Math class are extracted.
-- If selected, extra logging/tests and/or reports are produced.

-- PARAMETERS FOR OUTPUT CONTROL
-- <empty>        Silent: quiet, no reports, no demos and no examples
-- C              Classic: extracts only Classic items
-- O              Object: extracts only Object items
-- V              Verbose: detailed progress report and stats (console)
-- S              Standard: user documentation (repository)
-- E              Extra: developer documentation (cross references)
-- X              Execute: all examples
-- ?              Help: this text
```

In this folder you will find two models, Model1.rex and Model2.rex that you can use to include your program and then run as classic REXX program using regina or (oo)rexx by issuing regina run mypgm or as ooRexx program by issuing rexx mypgm. Run.rex is the preprocessor that composes a program runner.rex made up of your code surrounded by the two include files Setting.inc and Math.inc.

Math.inc contains about 750 functions that are grouped into topic oriented files such as Polynom.inc, Equation.inc, and List.inc as shown above.

Paul wrote a converter that creates Math.cls from Math.inc.
Here is an example what gets created from Amean.rex:

```
Amean:
-- Returns arithmetic mean                -- Description
-- number,number -> number                -- Signature
procedure                                 -- Protection
arg xx,yy                                 -- Get the arguments
if \Datatype(xx,'N') then say number      -- Check if they are numbers
if \Datatype(yy,'N') then say number
numeric digits Digits()+1                 -- Increase the precision
return 0.5*(xx+yy)                        -- Return he arithmetic mean

::method Amean public
-- Returns arithmetic mean
-- number,number -> number
arg xx,yy
signal on novalue name Handler            -- Take care of Novalue
signal on syntax name Handler             -- Take care of Syntax error
if \Datatype(xx,'N') then say number
if \Datatype(yy,'N') then say number
numeric digits Digits()+1
return 0.5*(xx+yy)
Handler:                                  -- in case of a problem
signal off novalue                        -- diplay diagnostics
parse source . . prog
call Abend
prog,sigl,Sourceline(sigl),Condition('C'),Condition('D'),rc,xx,yy

::routine Amean public
arg xx,yy
return .Math~Amean(xx,yy)
```

The extra lines in the method provide debugging informatiomm in case of an error situation.

I will now show some examples for a number of functional modules.

Randoms

**This module provides functions to generate (pseudo-)random numbers.
randu returns integer numbers from the given range up to 12 digits
observing a uniform distribution.
rand5 and rand12 return numbers with 5 or 12 digits within the
given range.
randn returns 12 digit random numbers following a normal distribution
with the specified mean and standard deviation.
A function rand lets you set a seed to get reproducable results for
different runs of the same program**

```
parse Arg seed',n
If seed='' Then seed=17
If n='' Then n=10000
Numeric Digits 20
Say 'Run' time()
sss=rand(seed); Say 'sss=rand('seed')   sss='sss
sun=''; do 6; sun=sun randu(-300000,300000); End; Say 'sun='sun
s12=''; do 4; s12=s12 rand12(-90,90); End; Say 's12='s12
s5=''; do 10; s5=s5 rand5(9000,-9000); End; Say 's5='s5
sd=''; Do 6; sd=sd randn(3,1); End; Say 'sd='sd
Call time 'R'
Say 'n='n
sum=0
Do i=1 To n; r.i=randn(-5,10); sum+=r.i; End
mean=sum/n
sum=0
do i=1 To n; d2=(r.i-mean)**2; sum+=d2; End
sd=sqrt(sum/n)
Say 'mean='mean 'standard deviation='sd
Say time('E') 'seconds'
```

Output:

```
F:\_paul>rexx brandoms 33,30000
Run 16:23:20
sss=rand(33)   sss=094087320639
sun= 69729 -258490 -31803 196 -58345 218694
s12= 49.3310270255 -32.3874784836 -5.27985071802 -16.2318935351
s5= 8065.3 4440.1 1331.3 8750.9 -6157.1 7168.7 7665.8 -7401.4 5980.1 8229.6
sd= 1.85008560022 2.67083436614 2.49810351124 4.21836928914 3.85419600188
3.19531519964
n=30000
mean=-5.0516639593846526667 standard deviation=9.931694808106677669670
9.513000 seconds

F:\_paul>rexx brandoms 33,30000
Run 16:23:33
sss=rand(33)   sss=094087320639
sun= 69729 -258490 -31803 196 -58345 218694
s12= 49.3310270255 -32.3874784836 -5.27985071802 -16.2318935351
s5= -487.98 3818.5 -7448.2 4110.3 8243.3 8379.4 4146.5 -8276.8 -4234.1
6705.9
sd= 1.85008560022 2.67083436614 2.49810351124 4.21836928914 3.85419600188
3.19531519964
n=30000
mean=-5.0516639593846526667 standard deviation=9.931694808106677669670
9.814000 seconds
```

Note that the results of s5 are different. The reason is that rand5 uses REXX‘ RANDOM function for performance reasons.. Changing Random(to Randu(in math.cls and/or math.inc fixes that difference.

Another memory oft he past:

When I saw this a couple of months ago:

```
if Memo.rand='' then do
    d=Date('s')
    a=Substr(d,3,2); b=Substr(d,5,2); c=Substr(d,7,2)
    t=Time('l')
    d=Substr(t,1,2); e=Substr(t,4,2); f=Substr(t,7,2)
    g=Substr(t,10,1); h=Substr(t,11,1); i=Substr(t,12,1)
    Memo.rand=h||f||d||b||a||c||e||g||i
  End
```

Memo.rand is, of course , the seed used and can be set by dmy=rand(integer)

I suggested to replace this code by

```
if Memo.rand='' then
  Memo.rand=Translate('SOPIJEFCDGHLMRT',Date('S')Time('L'),,
                                        'ABCDEFGHIJKLMNOPQRSTUVW')
```

Recently this method came up on the rexxla-members list.
I suggested ,my‘ improvement which was the topped by another one:

```
date=20260425
HDGdate=substr(date,1,4)'-'substr(date,5,2)'-'substr(date,7,2)
dt1=translate('abcd-ef-gh',date,'abcdefgh')
dt2=date('i',date,'S')
say HDGdate dt1 dt2 -> 2026-04-25 2026-04-25 2026-04-25
```

Sequence

This module provides functions that return or compute sequences of series defined in the literature. Some prominent ones are prime numbers and factors of a given integer.
Another is the Fibonacci series.
And did you ever hear of Tribonacci or Tetranacci?
In total there are 58 functions in this module that return series or series related constants. Most constants return a hard coded value up to a Numeric Digits setting of 110 and are computed if a higher precision is needed.

```
include setting
Numeric Digits 40
Parse Arg z
If z='' Then
  z=100000
n1=primes(z) -- get all prime numbers less than or equal to z
say n1 'primes. The largest one is' prim.n1
n2=primes(-z) -- get the first z prime numbers
say n2 'primes. The largest one is' prim.n2
nf=factors(12345678912345678912345678912345678 9)
zz=showst('FACT.','factors',5)
nfe=efactors(12345678912345678912345678912345678 9)
Do i=1 To nfe
  Say i efac.factor.i efac.exponent.i
  End
nfu=ufactors(12345678912345678912345678912345678 9)
Do i=1 To nfu
  Say i ufac.i
  End
Call trib  50
Say 'tribonacci()/1='tribonacci()/1 '- adjusted to digits()='digits()
Call trib 2000
Exit
trib:
Parse Arg d
Numeric Digits d
Say 'tribonacci('d')='tribonacci(d)
Return
include math
```

Output:

```
9592 primes. The largest one is 99991
100000 primes. The largest one is 1299709
FACT. 12 items factors
3 3 7 11 13
19 101 3607 3803 9901
52579 999999000001
1 3 2
2 7 1
3 11 1
4 13 1
5 19 1
6 101 1
7 3607 1
8 3803 1
9 9901 1
10 52579 1
11 999999000001 1
1 3
2 7
3 11
4 13
5 19
6 101
7 3607
8 3803
9 9901
10 52579
11 999999000001
tribonacci(50)=1.839286755214161132551852564653286600424178746097592246778758639404203222081966425738435419428307014141979826859240974164178450746507436943831545820499513796249655539644613666121540277972678118941041211609223282155956071816712182365986652273378537815696989252117395791413228721061878984085254956931145349134985345957617503596522132381424727272241735818770006979055102549044965710742526547722811006598937555636309333052826235753851971994299145300825466397747290058700597448139193167282584883962633297070068723683112783775025055712227515325957894656057068642228391865969829469135623922044319247614706881145172676671274396414621257184334266234039021835249459103322723106151328699703080803630222332499710524310747235423139974438182656560735194035787491176268052453707922111084971080687641005015654147566223500888566594971582183418486871480290125543699348051367916502585305387827666612622431776635820094298550538732599165178773018447238860426222324857820792721049160181783725613203439814302274533997621231550403386742932921162146187100543139497172087706678362853584316186872007627055606628661587258360489454871597451795731999462327709788239443131599315695499190298299048981191916257722782191174180199045440499947044043229285981508734918293245147828123401148003007073574598656098833438493566301129173893234063267481191764950157371974015880477404776348279399016395649947064551517494209599046758752434909014193948612979107608322724628671474351459189589696163947387982031102155041182486503156933830487420319810880465853331830851430366302449460735931295692058541968461824397581165038824739722367017295475978351918193473639497513647291668756644361268315558783190692655437890313648997152468605416411326029641077969966180265090775829186196438197376704935716357355144041786187768279063024644643664468198481021668435486716127419079338927184690612269028988954193748138004210823645826476913545130631559858288157914526792814736123596443096200148219371037625824498050020645570497054879533510694440165170667117893299783956
```

Trigonometric

This module provides 22 functions including the standard functions sin, cos, tan, cotan, sec, as well as their inverse functions.
Long time ago I implemented a package rxm.cls that provides the same functions as ooRexx' RxMath library with arbitrary (high) precision. RxMath is limited to 16 digits.
See below where the last digits of the three implementations of sine(x) differ.
The current math package works only with radians whereas RxMath and rxm.cls allow for using degrees for the given or returned angles.
math.inc (and math.cls) provides conversion routines between radians and degrees (rad and deg).

Conveniently you can get the constants halfpi, pi, and twopi (or alias tau).

Did you ever hear about constants named Dottie (the solution of x=cos(x)) or Kepler-Bouwkamp? I haven't!

All these functions are also provided for complex arguments in module Complex.

```
Numeric Digits 20
Say 'A short trip:'
x=1             ; Say 'x=1          ' x
s=sin(x)        ; Say 's=sin(x)     ' s
c=cos(x)        ; Say 'c=cos(x)     ' c
t=tan(x)        ; Say 't=tan(x)     ' t
u=s/c           ; Say 'u=s/c        ' u
it=arctan(t)    ; Say 'xx=arctan(t) ' it
iu=arctan(u)    ; Say 'xx=arctan(u) ' iu
sc=sec(x)       ; Say 'sc=sec(x)    ' sc
is=arcsec(sc)   ; Say 'is=arcsec(sc)' is
say
say 'Comparing the results'' precisions'
Numeric Digits 90
Parse Arg x
If x='' Then x=1.51
Say 'x='||x
say 'sin(x)             ='ht(sin(x),20,70)
say 'rxmsin(x,95,''R'')   ='ht(rxmsin(x,95,'R'),20,70)
say 'rxCalcsin(x,16,''R'')='rxCalcsin(x,16,'R')
say
Say 'and a glimpse at the complex world:'
Numeric Digits 20
sinc=sinc('1 -1')  ; Say 'sinc=sinc(''1 -1'')' sinc
asi=arcsinc(sinc)  ; Say 'asi=arcsinc(sinc)' asi
Exit
ht: Procedure
Parse Arg s,p1,p2; Return left(s,p1)'...'substr(s,p2)
::REQUIRES rxmath Library
::REQUIRES rxm
::REQUIRES math
```

Output:

```
F:\_paul>rexx btrig
A short trip:
x=1           1
s=sin(x)      0.8414709848078965066523
c=cos(x)      0.540302305868139717 4009
t=tan(x)      1.5574077246549022305 1
u=s/c         1.557407724654902230 5
xx=arctan(t)  1.000000000000000000 09
xx=arctan(u)  0.999999999999999999 98
sc=sec(x)     1.85081571768092561791
is=arcsec(sc) 1.00000000000000000000

Comparing the results' precisions
x=1.51
sin(x)              =0.998152472497548119...839714838333644293380 1917
rxmsin(x,95,'R')    =0.998152472497548119...839714838333644293380 1924567
rxCalcsin(x,16,'R')=0.9981524724975481

and a glimpse at the complex world:
sinc=sinc('1 -1') 1.29845758141597729483 -0.634963914784736108254
asi=arcsinc(sinc) 0.999999999999999999 94 -1.000000000000000000 01
```

Exponential

This module provídes 19 functions supporting calculation of logarithms with base e, 10, or any other base and the corresponting exponentiation functions exp, alog10, and power.

It also provides 10 functions for miscellaneous constants:
Euler constant (e), natural log of 2, 4, and 10, second Dubois-Reymond, Erdos-Tenenbaum-Ford, Gelfond, Gelfond-Schneider, Ramanujan, Van der Pauw.

```
include setting
Parse Arg z
If z='' Then
  z=2
Numeric Digits 20
                          Say 'z                   ='z
ln2=ln(z)/1              ; Say 'ln2=ln(z)/1        ='ln2
log10=log10(z)           ; Say 'log10=log10(z)     ='log10
logz8=logxy(z,8)         ; Say 'logz8=logxy(z,8)   ='logz8
za=exp(ln2)              ; Say 'za=exp(ln2)        ='za
zb=alog10(log10)         ; Say 'zb=alog10(log10)   ='zb
zc=power(8,logz8)        ; Say 'zc=power(8,logz8)  ='zc
r5=nroot(z,5)            ; Say 'r5=nroot(z,5)      ='r5
z5=power(r5,5)           ; Say 'z5=power(r5,5)     ='z5
g1=gelfond1()            ; Say 'g1=gelfond1()      ='normal(g1)
gg=power(e(),pi())       ; Say 'gg=power(e(),pi()) ='gg
include math
```

Output:

```
z                  =2
ln2=ln(z)/1        =0.69314718055994530942
log10=log10(z)     =0.30102999566398119521
logz8=logxy(z,8)   =0.33333333333333333333
za=exp(ln2)        =2.00000000000000000054
zb=alog10(log10)   =2.00000000000000000000120173
zc=power(8,logz8)  =1.99999999999999999998533
r5=nroot(z,5)      =1.148698354997035006799
z5=power(r5,5)     =2
g1=gelfond1()      =23.140692632779269006
gg=power(e(),pi()) =23.1406926327792690057373 5
```

Base

This module deals with encoding and conversions. Three alphabets are provided for bitcoin (Base58), file transfer (Base64), and conversions to and from any base from 2 to 62 (Base62).
It provides conversions from and to binary, decimal, and any other base.
c2b(x) is the same as x2b(c2x(x)) and d2n(x,16) is the same as d2x(x).
I wonder why c2n and n2c are missing in this module!

Then we find three not well known constants: Champernowne, Komornik-Loret, and Prouhet-Thue-Morse.

base.rex in math's distribution shows examples for all that:

```
Alphabets...
Base58() = 123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
Base62() = 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
Base64() = ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

Encoding and decoding...
Encode58('02468ACE') = 4NfqX
Decode58('4NfqX') = 2468ACE
Encode64('Rosetta Code.') = Um9zZXR0YSBDb2RlLg==
Decode64('Um9zZXR0YSBDb2RlLg==') = Rosetta Code.

Conversions...
B2c('0100000101000010') = AB
B2d('10101') = 21
C2b('AB') = 0100000101000010
D2b(123) = 01111011
D2n(12.34,16) = C.570A3D70A3
N2d('ABC.DEF',16) = 2748.8708496094
N2m('ABC.DEF',16,62) = iK.rzXwCkKEgG

Constants...
Champernowne()/1 = 0.123456789
Komornik()/1 = 1.78723165
Prouhet()/1 = 0.412454034

My small example:

Numeric digits 1000
Say 'Refer to base.rex in the distribution!'
c2b=c2b('Walter')       ; Say 'c2b='c2b
yyy=x2b(c2x('Walter'))  ; Say 'yyy='yyy
d2n=d2n(4711,16)        ; Say 'd2n='d2n
d2x=d2x(4711)           ; Say 'd2x='d2x
::requires math
```

Output:

```
Refer to base.rex in the distribution!
c2b=010101110110000101101100011101000110010101110010
yyy=010101110110000101101100011101000110010101110010
d2n=1267
d2x=1267
```

Complex

This module includes 64 functions, 26 of which are trigonometric or hyperbolic. There are functions for arithmetic operations, computing the logarithm and exponentiation. Complex numbers are represented by number pairs or just one number if the imaginary part is zero.
A few examples are shown here. Complex.rex in the distribution shows much more.

As always, a few less known constants: Landau-Bloch, Salem, and i (!)

```
Parse Arg z
If z='' Then
  z='-3 3'
Say 'z='z
lz=logc(z);          Say 'lz=logc(z)            lz='Rec2formC(lz),

'expc(lz)='Rec2formC(expc(lz))
pz=rec2polc(lz);     Say 'pz=rec2polc(lz)       pz='pol2formC(pz)
az=anglec(z);        Say 'az=anglec(z)          az='az'rad
='deg(az)'degrees'
zi=invc(z);          Say 'zi='zi
pz=mulc(z,invc(z));  Say 'pz=mulc(z,invc(z))    pz='pz
zc=conjc(z);         Say 'zc='zc
sz=addc(z,conjc(z)); Say 'sz=addc(z,conjc(z))   sz='sz '=>' Rec2formC(sz)
dz=subc(z,conjc(z)); Say 'dz=subc(z,conjc(z))   dz='dz '=>' Rec2formC(dz)
z3=cubec(z);         Say 'z3=cubec(z)           z3='z3 '=>' Rec2formC(z3),

'nrootc(z3,3)='Rec2formC(nrootc(z3,3))
r3=CbrtC(z3);        Say 'r3=CbrtC(z3)          r3='r3
zz3=cubec(r3);       Say 'zz3=cubec(r3)        zz3='zz3
Say
Say 'Constants:'
I=i();               Say 'I=i()                  I='i
Salem=salem();       Say 'Salem=salem()      salem='Normal(salem)
Landau2=Landau2();   Say 'Landau2=Landau2() Landau2='Normal(Landau2)
::requires math
```

Output:

```
z=-3 3
lz=logc(z)            lz=(1.44518587895+2.35619449019i) expc(lz)=(-
3.000000000+3.000000000i)
pz=rec2polc(lz)       pz=(2.76409382981r,1.0206141217557th)
az=anglec(z)          az=2.3561944902rad =135degrees
zi=-0.1666666667 -0.1666666667
pz=mulc(z,invc(z))    pz=1.00000000
zc=-3 -3
sz=addc(z,conjc(z))   sz=-6 => (-6)
dz=subc(z,conjc(z))   dz=0 6 => (6i)
z3=cubec(z)           z3=54 54 => (54+54i)
nrootc(z3,3)=(4.0980762113+1.0980762114i)
r3=CbrtC(z3)          r3=4.0980762113 1.0980762114
zz3=cubec(r3)        zz3=53.99999999 53.99999999

Constants:
I=i()                  I=0 1
Salem=salem()      salem=1.17628082
Landau2=Landau2() Landau2=0.543258965
```

Vectors

**This module provides 36 functions for vectors of different kinds:
2-dimensional vector rectangular (x,y) or polar (r phi)
3-dimensional vector rectangular (x,y,z) or cylindrical (rho phi z)
or spherical (r theta phi). All angles are given in radians.**

The trivial functions provided are for adding and subtracting vectors as well as computing their dot and cross products. For all these functions vectors must be rectangular or must be converted to their rectangular format.

**You may never have heard of triple and quadruple products (scalar and vector) of three or four dimensional vectors. I haven't either.
Anyway: ScaltripV, VecttripV, ScalquadV, VectquadV provide these products.**

```
include setting
v2r='1 1'
Say 'v2r='v2r   'formatted as' vect2form(v2r)
v22='2 2'
Say 'v22='v22   'formatted as' vect2form(v22)
Say 'addv(v2r,v22)='addv(v2r,v22)
Say 'lenv(v2r)='Normal(lenv(v2r))
Say 'angv(v2r)='Normal(angv(v2r))
v3r=1 1 1
Say 'lenv(v3r)='Normal(lenv(v3r))
v3c=Normal(Rec2cylV(v3r))
v3s=Normal(Rec2sphV(v3r))
Say 'v3r          ='v3r
Say 'Rec2cylV(v3r)='v3c '->' Cyl2RecV(v3c)
Say 'Rec2sphV(v3r)='v3s '->' Sph2RecV(v3s)
include math
```

Output:

```
v2r=1 1 formatted as [1,1]
v22=2 2 formatted as [2,2]
addv(v2r,v22)=3 3
lenv(v2r)=1.414213562373095048801688724209698078569671875376948073176679737
990732478462107038850387534327641572735013846 2
angv(v2r)=0.78539816339745
lenv(v3r)=1.732050807568877293527446341505872366942805253810380628055806979
451933016908800037081146186757248575675626141 4
v3r          =1 1 1
Rec2cylV(v3r)=1.414213562373095048801688724209698078569671875376948073176677
973799073247846210703885038753432764157273501384 62 0.78539816339745 1 ->
1.000000000 0.9999999999 1
Rec2sphV(v3r)=1.732050807568877293527446341505872366942805253810380628055806
697945193301690880003708114618675724857567562614 14 0.9553166181
0.7853981633974 -> 0.9999999997 0.9999999997 1.000000000
```

What ist the distance from Vienna to New York City?

I asked chatGPD what the distance between Vienna and NYC is.

Back came this answer:

The geographic coordinates of Vienna are approximately:
Latitude: 48.2082° N
Longitude: 16.3738° E

The geographic coordinates of New York City are approximately:
Latitude: 40.7128° N
Longitude: 74.0060° W

The average radius of the Earth is: 6,371 km

The straight-line (air) distance between Vienna and New York City is about: 6,800 km

```
include setting
r=6370
thv=rad(90-48.2)    ; Say left('thv='thv,40) 'theta for Vienna'
phv=rad(-16.36)     ; Say left('phv='phv,40) 'phi   for Vienna'
thn=rad(90-40.7)    ; Say left('thn='thn,40) 'theta for NYC'
phn=rad(74)         ; Say left('phn='phn,40) 'phi   for NYC'
vvs=r thv phv       ; Say left('vvs='vvs,40) 'spherical vector of Vienna'
vns=r thn phn       ; Say left('vns='vns,40) 'spherical vector of NYC'
vv=sph2recv(vvs)    ; Say left('vv= 'vv ,40) 'convert vvs to rectangular'
vn=sph2recv(vns)    ; Say left('vn= 'vn ,40) 'convert vns to rectangular'
ang=AngleV(vv,vn)   ; Say left('ang='ang,40) 'compute the angle between
them'
d=r*ang             ; Say copies(' ',40) 'd=r*ang compute the arc between
the vectors'
Say left('d='d'=6800?',40) 'Verify the result'
include math
```

Output:

```
thv=0.7295476272                          theta for Vienna
phv=-0.2855358656                         phi   for Vienna
thn=0.8604473211                          theta for NYC
phn=1.291543647                           phi   for NYC
vvs=6370 0.7295476272 -0.2855358656       spherical vector of Vienna
vns=6370 0.8604473211 1.291543647         spherical vector of NYC
vv= 4073.902532 -1195.924870 4748.682118 convert vvs to rectangular
vn= 1331.139813 4642.236221 4153.866833  convert vns to rectangular
ang=1.0667772785                          compute the angle between them
d=r*ang                                  compute the arc between the vectors
d=6795.37126=6800?                        verify the result
```

The net distance (moving through the earth's crust) would be

```
d1=subv(vn,vv)
Say lenv(d1)   → 6477.70618970km
```

Calculus

This module provides functions to evaluate an expression f(x), to compute the first and second differential quotient of that function for a specified x and to compute the definite integral of the function using various methods.

There are also functions related to the gamma function and integrating first order differential equation using Euler or Runge-Kutta. See calculus.rex in the distribution,

```
say 'Evaluation...'
call Example "Eval('x**3',5)"
Say
say 'Differentiation...'
call Example "Dif('x**3',2)"
call Example "Dif('x**3',2,2)"
Say
say 'Integration...'
call Example "Int('x**3',0,2)"
call Example "Boole('x**3',0,2)"
call Example "Gaussian('x**3',0,2)"
call Example "Midpoint('x**3',0,2)"
call Example "Romberg('x**3',0,2)"
call Example "Simpson('x**3',0,2)"
call Example "Tanhsinh('x**3',0,2)"
call Example "Trapezoid('x**3',0,2)"
exit
Example:
-- Shows example
-- -> console
parse arg xx
interpret "Say xx '=' Normal("xx")"
return
::requires math
```

Output:

```
F:\_paul>rexx bcalculus
Evaluation...
Eval('x**3',5) = 125

Differentiation...
Dif('x**3',2) = 12
Dif('x**3',2,2) = 12

Integration...
Int('x**3',0,2) = 4
Boole('x**3',0,2) = 4
Gaussian('x**3',0,2) = 4
Midpoint('x**3',0,2) = 4
Romberg('x**3',0,2) = 4
Simpson('x**3',0,2) = 4
Tanhsinh('x**3',0,2) = 4
Trapezoid('x**3',0,2) = 4
```

Geometric

This module provides one function used in geometry but apart from that a large number of constants starting with the well known golden ratio. Other such ratios are called super golden, silver, bronze, metallic, and plastic ratio.
phi and psi are aliases for the golden and super golden ratios, respectively. Other constants you can find here are named Devicci, Goldenangle, Hermite, Lemniscate, Lieb, Magicangle, Paperfolding, Parabolic, and Robbins.

Ahh: The only geometric function in this module is Hypot

```
include setting
Parse Arg a b
If a='' Then
  Parse Value 3 4 with a b
Say 'a='a
Say 'a='||b
c=hypot(a,b); Say 'c=hypot(a,b)' c
Numeric Digits 20
Say
Say 'and now a few ratios'
Golden=Golden()          ; Say 'Golden()   ='Normal(Golden)
Psi=Psi()                ; Say 'Psi()      ='Normal(Psi)  'Super Golden'
Silver=Silver()          ; Say 'Silver()   ='Normal(Silver)
Bronze=Bronze()          ; Say 'Bronze()   ='Normal(Bronze)
Metallic=Metallic(2)     ; Say 'Metallic(2)='Normal(Metallic)
Plastic=Plastic()        ; Say 'Plastic()  ='Normal(Plastic)
include math
```

Output:

```
a=3
a=4
c=hypot(a,b) 5.000

and now a few ratios
Golden()   =1.6180339887498948482
Psi()      =1.4655712318767680267 Super Golden
Silver()   =2.4142135623730950488
Bronze()   =3.3027756377319946466
Metallic(2)=2.4142135623730950488
Plastic()  =1.324717957244746026
```

Formats

This module provides functions that extend the possibilities offered by REXX' FORMAT function.
Commatize makes huge numbers readable (and does not round to numeric digits. See, however, below.) It inserts commas for any three digits before the decimal point. The example below shows that leading zeros and trailing zeros after the decimal point should be removed :-)
Std shows the number rounded to digit() without exponent, adding trailing zeros when needed.
Sci and Eng show the rounded number with one or up to two (for Eng) digits before the 'decimal point'.
Digit rounds to the specified number of digits.
Round rounds to the specified decimals.
Fuzzy removes the specified number of decimals.

```
Numeric digits 9
z=00123456789012345.67890
zs=z/1e14
Say 'z='z
Say 'zs='zs
Say 'Commatize(z) ='Commatize(z)
Say 'Commatize(00123.45678901234567890)
='Commatize(00123.45678901234567890)
Say 'Std(z)       ='Std(z)
Say 'Sci(z)       ='Sci(z)
Say 'Eng(z)       ='Eng(z)
Say 'Normal(z)    ='Normal(z)
Say 'Digit(z,5)   ='Digit(z,5)
Say 'Round(zs,6)  ='Round(zs,6)
Say 'Fuzzy(z,6)   ='Fuzzy(z,6)
::REQUIRES math
```

Output:

```
z=00123456789012345.67890
zs=1.23456789
Commatize(z) =00,123,456,789,012,345.67890
Commatize(00123.45678901234567890) =00,123.45678901234567890
Std(z)       =123456789000000
Sci(z)       =1.23456789E14
Eng(z)       =123.456789E12
Normal(z)    =1.23456789E+14
Digit(z,5)   =1.2346E+14
Round(zs,6)  =1.234568
Fuzzy(z,6)   =1.23E+14
```

Paul fixed the problem with the zeros in the meanwhile and the result of Commatize(00123.456789012345.67890) is now

```
123,456,789,000,000
```

because commatize in math now rounds the input first according to digits().

To get all digits one can set Numerig Digits 30 or whatever:

```
123,456,789,012,345.67890
```

Statistic

This module deals with some statistical properties.
You can compute miscellaneous means of two numbers or of a list of numbers. It also supports computing the the cumulative density for some distributions (binomial, normal, and Poisson).
statl(list) returns arithmetic mean, standard deviation, and variance of a list of numbers.
Running statistic.rex from the distribution shaws examples for all these functions and the provided constants: Bernstein, Chaitin, Feigenbaum1, Feigenbaum2, Gauss, Lebesgue. and Zscore.

```
include setting
Parse Arg a b
If a='' Then
  Parse Value 1 5 with a b
Say 'a='a
Say 'b='||b
Amean=Amean(a,b)  ;Say 'Amean=Amean(a,b)  ='Amean' arithmetic mean'
Agmean=Agmean(a,b);Say 'Agmean=Agmean(a,b)='Agmean' arithmetic-geom mean'
Gmean=Gmean(a,b)  ;Say 'Gmean=Gmean(a,b)  ='normal(Gmean)' geometric mean'
Cmean=Cmean(a,b)  ;Say 'Cmean=Cmean(a,b)  ='Cmean' circular mean'
Hmean=Hmean(a,b)  ;Say 'Hmean=Hmean(a,b)  ='Hmean' harmonic mean'
Qmean=Qmean(a,b)  ;Say 'Qmean=Qmean(a,b)  ='Qmean' quadratic mean'
nl='1;2;3;4;5;6;7;8;9'
Parse Value StatsL(nl) With mean dev var
Say 'nl='nl
Say 'Mean='mean
Say 'Standard deviation='dev
Say 'Variance='var
include math
```

Output:

```
a=1
b=5
Amean=Amean(a,b)  =3.0 arithmetic mean
Agmean=Agmean(a,b)=2.6040081906 arithmetic-geom mean
Gmean=Gmean(a,b)  =2.23606798 geometric mean
Cmean=Cmean(a,b)  =1.7123889805 circular mean
Hmean=Hmean(a,b)  =1.666666667 harmonic mean
Qmean=Qmean(a,b)  =3.60555127547 quadratic mean
nl=1;2;3;4;5;6;7;8;9
Mean=5
Standard deviation=2.58198889754
Variance=6.666666667
```

NTheory

**This module provides functions and constants from number theory.
There are 54 functions that return the xxth element of some sequence.
13 functions return a constant,
12 functions determine if a number has a given property,
and a few deal with the smallest prime factor of a given integer.
Here are a few examples. More can be seen by running Ntheory.rex of the distribution.**

```
Numeric Digits 100
parse Arg z
If z='' Then z=12345678999999999999999991
Say '            z='z
Say 'ConwayN(10)  ='ConwayN(10)
Say 'Artin()      ='Normal(Artin())  'length(artin())='length(artin())
Call time 'R'
Say 'Composite(12)='Composite(z)  time('E'); Call time 'R'
Say
Say 'Factorization:'
Say 'Ffactor(z)   ='Ffactor(z)     time('E'); Call time 'R'
Say 'Pollardrho(z)='Pollardrho(z)  time('E'); Call time 'R'
Say 'Squfof(z)    ='Squfof(z)      time('E'); Call time 'R'
Say
Say 'Home Primes'
hp=home(1)
Do n=2 To 10
  hp=hp',' home(n)
  End
Say 'The first 10 home primes are' strip(hp)
::REQUIRES math
```

Output:

```
            z=12345678999999999999999991
ConwayN(10)  =111312211331121321132122 21
Artin()
=0.37395581361920228805472805434641641511162924860615004209474280241735018 2
040028082344304317087250569 length(artin())=107
Composite(z)=1 292.618000

Factorization:
Ffactor(z)   =164326273 270.594000
Pollardrho(z)=164326273 0.484000
Squfof(z)    =164326273 243.172000

Home Primes
The first 10 home primes are 1, 2, 3, 211, 5, 23, 7, 3331113965338635107,
```

An example from rosettacode.org

https://rosettacode.org/wiki/Giuga_numbers

Giuga numbers: composite numbers n such that p divides n/p - 1 for every prime divisor p of n.

```
Numeric Digits 1000
Call time 'R'
list=''
max=100000
do n=1 To max
  If uf(n) Then
    list=list n
  End
Say 'Giuga numbers less than' max 'are:' list
Say 'This took' time('E') 'seconds'
Say 'Verifying some solutions found by C++ and others on rosettacode.org'
Call ufv(2214408306)
Call ufv(24423128562)
Call ufv(432749205173838)
Call ufv(14737133470010574)
Call ufv(550843391309130318)
Call ufv(244197000982499715087866346)
Say 'Module sequence returns all 12 numbers listed in the'
Say 'On-Line_Encyclopedia_of_Integer_Sequences:'
n=giugas(1e39)
Do i=1 To n
  Say getst('GIUG.',i)
  End
Exit
uf: Procedure Expose UFAC.
  Parse Arg n
  nn=ufactors(n)
  If nn>1 Then Do
    do i=1 To nn
      f=getst('UFAC.',i)
      If (n/f-1)//f<>0 Then leave
      End
    if i> nn Then
      Return 1
    End
  Return 0

ufv: Procedure
  Parse Arg n
  Call time 'R'
  If uf(n) Then
    Say n 'is a Giuga number' time('E') 'seconds'
  Return

::requires math
```

Output:

```
Giuga numbers less than 100000 are:  30 858 1722 66198
This took 44.066000 seconds
Verifying some solutions found by C++ and others on rosettacode.org
2214408306 is a Giuga number 0 seconds
24423128562 is a Giuga number 0.008000 seconds
432749205173838 is a Giuga number 0.001000 seconds
14737133470010574 is a Giuga number 0.033000 seconds
550843391309130318 is a Giuga number 0.082000 seconds
2441970009824997150878663 46 is a Giuga number 3.523000 seconds
Module sequence returns all 12 numbers listed in the
On-Line_Encyclopedia_of_Integer_Sequences:
30
858
1722
66198
2214408306
24423128562
432749205173838
14737133470010574
550843391309130318
2441970009824997150878663 46
554079914617070801288578559178
191066718142050798455575991633 8506
```

The On-Line_Encyclopedia_of_Integer_Sequences https://oeis.org/A007850 lists one more but it isn't known if this is the next one

```
420001794970774706203871150967065663240419575375163060922876441614\
25572115820984325451903234748 18
```