

A Module for Rounding Values

Tomáš Hála, Tamara Kocurová, Adriana Kašparová, Hans Hagen

The article describes the module `util-rnd.lua`, which was created as an extension of a module for drawing statistical charts. The described module implements a total of nine rounding methods, which are presented as source code snippets with the corresponding functions. The results of function calls are clearly compared in the table.

Introduction

Rounding is a fundamental numerical operation that finds widespread application in many fields of science, engineering and computing. Although the principle of rounding is simple, its correct implementation is crucial for ensuring the accuracy and consistency of results in numerical calculations. This article focuses on a new module for rounding calculations, developed in the Lua programming language and then integrated into the ConTeXt typesetting system.

About the module `util-rnd.lua`

This module was created as a by-product while creating the statistical charts module (see Kocurová, Kašparová & Hála, 2020), when the need for a tool for correct rounding of values using various methods became apparent, not only by functions `math.floor` and `math.ceil` available in the programming language Lua.

Overview of implemented rounding functions

The following nine methods have been implemented:

no – no rounding

```
no = function ( num )  
    return num  
end
```

up – ceiling rounding

```
up = function ( num, coef )  
    coef = coef and pow(10,coef) or 1  
    return ceil(num * coef) / coef  
end
```

down – floor rounding

```
down = function ( num, coef )  
    coef = coef and pow(10,coef) or 1  
    return floor(num * coef) / coef  
end
```

halfup – rounds decimal numbers as usual, including those with 0.5, up as usual (e.g. number -0.5 will be rounded to 0)

```
halfup = function ( num, coef )  
    coef = coef and pow(10,coef) or 1  
    return floor(num * coef + 0.5) /  
        coef  
end
```

halfdown – rounds decimal numbers as usual, numbers with 0.5 down, too (e.g. number 0.5 will be rounded to 0)

```
halfdown = function ( num, coef )
  coef = coef and pow(10,coef) or 1
  return ceil(num * coef -0.5) / coef
end
```

halfabsup – rounds decimal numbers as usual, numbers with 0.5 not to zero, e.g. numbers -0.5 and 0.5 will be rounded to -1 and 1, respectively

```
halfabsup = function ( num, coef )
  coef = coef and pow(10,coef) or 1
  return (num >= 0
    and floor(num * coef + 0.5)
    or ceil(num * coef - 0.5)) /
    coef
end
```

halfabsdown – rounds decimal numbers as usual, numbers with 0.5 towards zero, e.g. numbers -0.5 and 0.5 will be both rounded to 0

```
halfabsdown = function ( num, coef )
  coef = coef and pow(10,coef) or 1
  return (num < 0
    and floor(num * coef + 0.5)
    or ceil(num * coef - 0.5)) /
    coef
end
```

halfeven – rounds decimal numbers as usual, numbers with 0.5 to the nearest even number, e.g. numbers 1.5 and 2.5 will be rounded both to 2

```
halfeven = function ( num, coef )
  coef = coef and pow(10,coef) or 1
  num = num * coef
  return floor ( num +
    ((num - floor(num)) ~= 0.5 and 0.5
```

```
)
  or ((floor(num)%2 == 1)
    and 1) or 0)) / coef
end
```

halfodd – rounds decimal numbers as usual, numbers with 0.5 to the nearest odd number, e.g. numbers 1.5 and 2.5 will be rounded to 1 and 3, respectively

```
halfodd = function ( num, coef )
  coef = coef and pow(10,coef) or 1
  num = num * coef
  return floor( num +
    ((num - floor(num)) ~= 0.5 and 0.5
    )
    or ((floor(num) % 2 == 1)
    and 0) or 1)) / coef
end
```

The functions `floor`, `ceil`, `pow`, `gsub` and `lower`, mentioned in the code snippets above, correspond to the functions of the same name from the `math` and `string` libraries.

The main function

To simplify the work, all rounding requests are covered by one common function:

```
rounding.round =
  function ( num, dec, mode )
    if type(dec) == "string" then
      mode = dec
      dec = 1
    end
    return ( mode
      and methods[mode]
      or defaultmethod)(num,dec)
end
```

where `num` is the number to be rounded, `dec` stands for the number of decimal places, and `mode` is the name (as a string) of a particular

contextgroup > context meeting 2024

rounding method (see above). The number of decimal places can be omitted, in which case the default value 1 will be used.

Examples of use

The typical way of joining the module followed by creation of a local variable, e.g. `round`, can be used:

```
local rounding = require "util-rnd"
local round    = rounding.round
```

However, one can also take it directly from the `number` structure:

```
require "util-rnd"
local round    = number.rounding.round
```

because then internal structure `rounding` has been assigned in the module as a key of the global structure `number`.

And then the rounding function can be tested

```
context(round( 6.50, 1, "up" ), " ",
        round( 6.50, 0, "up" ), " ",
        round( 6.5 , 1, "down"), " ",
        round( 6.55, 0, "down" ))
```

with expected output values 6.5, 7.0, 6.5, and 6.0, respectively.

Conclusion

The module `util-rnd.lua` supports nine rounding methods, compared in Table 1, but this set of methods can be expanded at any time. A big advantage is that the rounding tools are available in the structure `number`.

The module has been introduced at 16th ConT_EXt Meeting in Dreifelden, Germany, in September 2022.

After Hans modified this module to be consistent with the other modules, the module has been included in the official distribution of ConT_EXt and T_EXlive since spring 2023.

References

KOCUROVÁ, T., KAŠPAROVÁ, A., HÁLA, T.
t-statistical-charts [on-line]. Ver. 0.42.
 c2020–2024. [cit. 2025-07-14].
 Available at www.thala.cz/statcharts/.

no	6.49	6.5	6.51	6.54	6.55	6.56	6.65	-6.49	-6.5	-6.51	-6.54	-6.55	-6.56	-6.65
up	6.5	6.5	6.6	6.6	6.6	6.6	6.7	-6.4	-6.5	-6.5	-6.5	-6.5	-6.5	-6.6
down	6.4	6.5	6.5	6.5	6.5	6.5	6.6	-6.5	-6.5	-6.6	-6.6	-6.6	-6.6	-6.7
halfup	6.5	6.5	6.5	6.5	6.6	6.6	6.7	-6.5	-6.5	-6.5	-6.5	-6.5	-6.6	-6.6
halfdown	6.5	6.5	6.5	6.5	6.5	6.6	6.6	-6.5	-6.5	-6.5	-6.5	-6.6	-6.6	-6.7
halfabsup	6.5	6.5	6.5	6.5	6.6	6.6	6.7	-6.5	-6.5	-6.5	-6.5	-6.6	-6.6	-6.7
halfabsdown	6.5	6.5	6.5	6.5	6.5	6.6	6.6	-6.5	-6.5	-6.5	-6.5	-6.5	-6.6	-6.6
halfeven	6.5	6.5	6.5	6.5	6.6	6.6	6.6	-6.5	-6.5	-6.5	-6.5	-6.6	-6.6	-6.6
halfodd	6.5	6.5	6.5	6.5	6.5	6.6	6.7	-6.5	-6.5	-6.5	-6.5	-6.5	-6.6	-6.7

Table 1. comparison of rounding methods