Home

Pages Classes Methods

Search

Show/hide navigation

Parent

Numeric

Methods

#*
#**
#+
#-
#-@
#/
#<=>
#==
#abs
#ceil
#denominator
#fdiv
#floor
#hash
#inspect
#magnitude
#negative?
#numerator
#positive?
#quo
#rationalize
#round
#to_f
#to_i
#to_r
#to_s
#truncate

class Rational

A rational number can be represented as a pair of integer numbers: a/b (b>0), where a is the numerator and b is the denominator. `Integer` a equals rational a/1 mathematically.

You can create a Rational object explicitly with:

- A rational literal.

You can convert certain objects to Rationals with:

- Method `Rational`.

Examples

```
Rational(1)      #=> (1/1)
Rational(2, 3)   #=> (2/3)
Rational(4, -6)  #=> (-2/3) # Reduced.
3.to_r           #=> (3/1)
2/3r             #=> (2/3)
```

You can also create rational objects from floating-point numbers or strings.

```
Rational(0.3)    #=> (5404319552844595/18014398509481984)
Rational('0.3')  #=> (3/10)
Rational('2/3')  #=> (2/3)

0.3.to_r         #=> (5404319552844595/18014398509481984)
'0.3'.to_r       #=> (3/10)
'2/3'.to_r       #=> (2/3)
0.3.rationalize  #=> (3/10)
```

A rational object is an exact number, which helps you to write programs without any rounding errors.

```
10.times.inject(0) {|t| t + 0.1 }              #=> 0.9999999999999999
10.times.inject(0) {|t| t + Rational('0.1') }  #=> (1/1)
```

However, when an expression includes an inexact component (numerical value or operation), it will produce an inexact result.

```
Rational(10) / 3   #=> (10/3)
Rational(10) / 3.0 #=> 3.3333333333333335

Rational(-8) ** Rational(1, 3)
                   #=> (1.0000000000000002+1.7320508075688772i)
```

Public Instance Methods

rat * numeric → numeric

Performs multiplication.

```
Rational(2, 3)  * Rational(2, 3)   #=> (4/9)
Rational(900)   * Rational(1)      #=> (900/1)
Rational(-2, 9) * Rational(-9, 2)  #=> (1/1)
Rational(9, 8)  * 4                #=> (9/2)
Rational(20, 9) * 9.8              #=> 21.77777777777778
```

rat ** numeric → numeric

Performs exponentiation.

```
Rational(2)    ** Rational(3)     #=> (8/1)
Rational(10)   ** -2              #=> (1/100)
Rational(10)   ** -2.0            #=> 0.01
Rational(-4)   ** Rational(1, 2)  #=> (0.0+2.0i)
Rational(1, 2) ** 0               #=> (1/1)
Rational(1, 2) ** 0.0             #=> 1.0
```

rat + numeric → numeric

Performs addition.

```
Rational(2, 3)  + Rational(2, 3)   #=> (4/3)
Rational(900)   + Rational(1)      #=> (901/1)
Rational(-2, 9) + Rational(-9, 2)  #=> (-85/18)
Rational(9, 8)  + 4                #=> (41/8)
Rational(20, 9) + 9.8              #=> 12.022222222222222
```

rat - numeric → numeric

Performs subtraction.

```
Rational(2, 3)  - Rational(2, 3)   #=> (0/1)
Rational(900)   - Rational(1)      #=> (899/1)
Rational(-2, 9) - Rational(-9, 2)  #=> (77/18)
Rational(9, 8)  - 4                #=> (-23/8)
Rational(20, 9) - 9.8              #=> -7.577777777777778
```

-rat → rational

Negates `rat`.

rat / numeric → numeric

Performs division.

```
Rational(2, 3)  / Rational(2, 3)   #=> (1/1)
Rational(900)   / Rational(1)      #=> (900/1)
Rational(-2, 9) / Rational(-9, 2)  #=> (4/81)
Rational(9, 8)  / 4                #=> (9/32)
Rational(20, 9) / 9.8              #=> 0.22675736961451246
```

Also aliased as: quo

rational <=> numeric → -1, 0, +1, or nil

Returns -1, 0, or +1 depending on whether `rational` is less than, equal to, or greater than `numeric`.

`nil` is returned if the two values are incomparable.

```
Rational(2, 3) <=> Rational(2, 3)  #=> 0
Rational(5)    <=> 5               #=> 0
Rational(2, 3) <=> Rational(1, 3)  #=> 1
Rational(1, 3) <=> 1               #=> -1
Rational(1, 3) <=> 0.3             #=> 1

Rational(1, 3) <=> "0.3"           #=> nil
```

rat == object → true or false

Returns `true` if `rat` equals `object` numerically.

```
Rational(2, 3)  == Rational(2, 3)   #=> true
Rational(5)     == 5                #=> true
Rational(0)     == 0.0              #=> true
Rational('1/3') == 0.33             #=> false
Rational('1/2') == '1/2'            #=> false
```

abs → rational

Returns the absolute value of `rat`.

```
(1/2r).abs    #=> (1/2)
(-1/2r).abs   #=> (1/2)
```

Also aliased as: magnitude

ceil([ndigits]) → integer or rational

Returns the smallest number greater than or equal to `rat` with a precision of `ndigits` decimal digits (default: 0).

When the precision is negative, the returned value is an integer with at least `ndigits.abs` trailing zeros.

Returns a rational when `ndigits` is positive, otherwise returns an integer.

```
Rational(3).ceil      #=> 3
Rational(2, 3).ceil   #=> 1
Rational(-3, 2).ceil  #=> -1

  #    decimal      -  1  2  3 . 4  5  6
  #                   ^  ^  ^  ^   ^  ^
  #   precision      -3 -2 -1  0  +1 +2

Rational('-123.456').ceil(+1).to_f  #=> -123.4
Rational('-123.456').ceil(-1)       #=> -120
```

denominator → integer

Returns the denominator (always positive).

```
Rational(7).denominator             #=> 1
Rational(7, 1).denominator          #=> 1
Rational(9, -4).denominator         #=> 4
Rational(-2, -10).denominator       #=> 5
```

fdiv(numeric) → float

Performs division and returns the value as a `Float`.

```
Rational(2, 3).fdiv(1)       #=> 0.6666666666666666
Rational(2, 3).fdiv(0.5)     #=> 1.3333333333333333
Rational(2).fdiv(3)          #=> 0.6666666666666666
```

floor([ndigits]) → integer or rational

Returns the largest number less than or equal to `rat` with a precision of `ndigits` decimal digits (default: 0).

When the precision is negative, the returned value is an integer with at least `ndigits.abs` trailing zeros.

Returns a rational when `ndigits` is positive, otherwise returns an integer.

```
Rational(3).floor      #=> 3
Rational(2, 3).floor   #=> 0
Rational(-3, 2).floor  #=> -2

  #    decimal      -  1  2  3 . 4  5  6
  #                   ^  ^  ^  ^   ^  ^
  #   precision      -3 -2 -1  0  +1 +2

Rational('-123.456').floor(+1).to_f  #=> -123.5
Rational('-123.456').floor(-1)       #=> -130
```

hash()

inspect → string

Returns the value as a string for inspection.

```
Rational(2).inspect      #=> "(2/1)"
Rational(-8, 6).inspect  #=> "(-4/3)"
Rational('1/2').inspect  #=> "(1/2)"
```

magnitude → rational

Returns the absolute value of `rat`.

```
(1/2r).abs    #=> (1/2)
(-1/2r).abs   #=> (1/2)
```

Alias for: abs

negative? → true or false

Returns `true` if `rat` is less than 0.

numerator → integer

Returns the numerator.

```
Rational(7).numerator       #=> 7
Rational(7, 1).numerator    #=> 7
Rational(9, -4).numerator   #=> -9
Rational(-2, -10).numerator #=> 1
```

`positive? → true or false`

Returns `true` if `rat` is greater than 0.

`quo(numeric) → numeric`

Performs division.

```
Rational(2, 3)  / Rational(2, 3)   #=> (1/1)
Rational(900)   / Rational(1)      #=> (900/1)
Rational(-2, 9) / Rational(-9, 2)  #=> (4/81)
Rational(9, 8)  / 4                #=> (9/32)
Rational(20, 9) / 9.8              #=> 0.22675736961451246
```

Alias for: /

`rationalize → self`
`rationalize(eps) → rational`

Returns a simpler approximation of the value if the optional argument `eps` is given (rat-|eps| <= result <= rat+|eps|), self otherwise.

```
r = Rational(5033165, 16777216)
r.rationalize                    #=> (5033165/16777216)
r.rationalize(Rational('0.01'))  #=> (3/10)
r.rationalize(Rational('0.1'))   #=> (1/3)
```

`round([ndigits] [, half: mode]) → integer or rational`

Returns `rat` rounded to the nearest value with a precision of `ndigits` decimal digits (default: 0).

When the precision is negative, the returned value is an integer with at least `ndigits.abs` trailing zeros.

Returns a rational when `ndigits` is positive, otherwise returns an integer.

```
Rational(3).round      #=> 3
Rational(2, 3).round   #=> 1
Rational(-3, 2).round  #=> -2

  #    decimal      -  1  2  3 . 4  5  6
  #                   ^  ^  ^  ^   ^  ^
  #   precision      -3 -2 -1  0  +1 +2

Rational('-123.456').round(+1).to_f  #=> -123.5
Rational('-123.456').round(-1)       #=> -120
```

The optional `half` keyword argument is available similar to `Float#round`.

```
Rational(25, 100).round(1, half: :up)    #=> (3/10)
Rational(25, 100).round(1, half: :down)  #=> (1/5)
Rational(25, 100).round(1, half: :even)  #=> (1/5)
Rational(35, 100).round(1, half: :up)    #=> (2/5)
Rational(35, 100).round(1, half: :down)  #=> (3/10)
Rational(35, 100).round(1, half: :even)  #=> (2/5)
Rational(-25, 100).round(1, half: :up)   #=> (-3/10)
Rational(-25, 100).round(1, half: :down) #=> (-1/5)
Rational(-25, 100).round(1, half: :even) #=> (-1/5)
```

to_f → float

Returns the value as a `Float`.

```
Rational(2).to_f      #=> 2.0
Rational(9, 4).to_f   #=> 2.25
Rational(-3, 4).to_f  #=> -0.75
Rational(20, 3).to_f  #=> 6.666666666666667
```

to_i → integer

Returns the truncated value as an integer.

Equivalent to `Rational#truncate`.

```
Rational(2, 3).to_i    #=> 0
Rational(3).to_i       #=> 3
Rational(300.6).to_i   #=> 300
Rational(98, 71).to_i  #=> 1
Rational(-31, 2).to_i  #=> -15
```

to_r → self

Returns self.

```
Rational(2).to_r      #=> (2/1)
Rational(-8, 6).to_r  #=> (-4/3)
```

to_s → string

Returns the value as a string.

```
Rational(2).to_s      #=> "2/1"
Rational(-8, 6).to_s  #=> "-4/3"
Rational('1/2').to_s  #=> "1/2"
```

truncate([ndigits]) → integer or rational

Returns `rat` truncated (toward zero) to a precision of `ndigits` decimal digits (default: 0).

When the precision is negative, the returned value is an integer with at least `ndigits.abs` trailing zeros.

Returns a rational when `ndigits` is positive, otherwise returns an integer.

```
Rational(3).truncate      #=> 3
Rational(2, 3).truncate   #=> 0
Rational(-3, 2).truncate  #=> -1

  #    decimal      -  1  2  3 . 4  5  6
  #                   ^  ^  ^  ^   ^  ^
  #   precision      -3 -2 -1  0  +1 +2

Rational('-123.456').truncate(+1).to_f  #=> -123.4
Rational('-123.456').truncate(-1)       #=> -120
```

Validate
Generated by RDoc 6.4.0.
Based on Darkfish by Michael Granger.
Ruby-doc.org is provided by James Britt and Neurogami.
Maximum R+D.