# How to convert floats to human-readable fractions?

Asked 16 years, 3 months ago Modified 2 years, 11 months ago

Viewed 66k times

117

Let's say we have `0.33`, we need to output `1/3`.
If we have `0.4`, we need to output `2/5`.

The idea is to make it human-readable to make the user understand "**x parts out of y**" as a better way of understanding data.

I know that percentages is a good substitute but I was wondering if there was a simple way to do this?

algorithm language-agnostic numbers

Share

Improve this question

Follow

edited Feb 14, 2020 at 15:10

Penny Liu

17.2k ● 5 ● 86 ● 108

asked Sep 18, 2008 at 19:00

Swaroop C H

17k ● 10 ● 45 ● 51

3 The `.33` => `"1/3"` example concerns me; I would expect `.33` => `"33/100"`. I assume you meant `.33...` of

course, but it exposes an issue with the question - before we can settle on an algorithm we need to decide on expected behavior. @Debilski's Python answer uses `.limit_denominator()` which defaults to a max denominator of 10^7; probably a good default in practice, but this can still introduce bugs if you're not careful, and *does* return `"33/100"` in the `.33` case. – dimo414 Apr 22, 2015 at 17:44

With whatever language-*specifc* features are available. Unclear what you're asking, if indeed it isn't a mere contradiction in terms. – user207421 Feb 13, 2017 at 9:05

Convert a float to a rational number that is guaranteed to convert back to the original float – phuclv Jan 3, 2022 at 0:30

## 26 Answers

Sorted by: Highest score (default)

77

I have found David Eppstein's find rational approximation to given real number C code to be exactly what you are asking for. Its based on the theory of continued fractions and very fast and fairly compact.

I have used versions of this customized for specific numerator and denominator limits.

```
/*
** find rational approximation to given real number
** David Eppstein / UC Irvine / 8 Aug 1993
**
** With corrections from Arno Formella, May 2008
**
** usage: a.out r d
**   r is real number to approx
**   d is the maximum denominator allowed
**
```

```
** based on the theory of continued fractions
** if x = a1 + 1/(a2 + 1/(a3 + 1/(a4 + ...)))
** then best approximation is found by truncating this
** (with some adjustments in the last term).
**
** Note the fraction can be recovered as the first col
**  ( a1 1 ) ( a2 1 ) ( a3 1 ) ...
**  ( 1  0 ) ( 1  0 ) ( 1  0 )
** Instead of keeping the sequence of continued fracti
** we just keep the last partial product of these matr
*/

#include <stdio.h>

main(ac, av)
int ac;
char ** av;
{
    double atof();
    int atoi();
    void exit();

    long m[2][2];
    double x, startx;
    long maxden;
    long ai;

    /* read command line arguments */
    if (ac != 3) {
        fprintf(stderr, "usage: %s r d\n",av[0]);  //
        exit(1);
    }
    startx = x = atof(av[1]);
    maxden = atoi(av[2]);

    /* initialize matrix */
    m[0][0] = m[1][1] = 1;
    m[0][1] = m[1][0] = 0;

    /* loop finding terms until denom gets too big */
    while (m[1][0] *  ( ai = (long)x ) + m[1][1] <= ma
        long t;
        t = m[0][0] * ai + m[0][1];
        m[0][1] = m[0][0];
        m[0][0] = t;
        t = m[1][0] * ai + m[1][1];
        m[1][1] = m[1][0];
        m[1][0] = t;
        if(x==(double)ai) break;     // AF: division b
        x = 1/(x - (double) ai);
        if(x>(double)0x7FFFFFFF) break;  // AF: repres
    }

    /* now remaining x is between 0 and 1/ai */
    /* approx as either 0 or 1/m where m is max that w
    /* first try zero */
    printf("%ld/%ld, error = %e\n", m[0][0], m[1][0],
           startx - ((double) m[0][0] / (double) m[1][

    /* now try other possibility */
    ai = (maxden - m[1][1]) / m[1][0];
    m[0][0] = m[0][0] * ai + m[0][1];
    m[1][0] = m[1][0] * ai + m[1][1];
    printf("%ld/%ld, error = %e\n", m[0][0], m[1][0],
           startx - ((double) m[0][0] / (double) m[1][
}
```

Share Improve this answer Follow

edited Jan 21, 2022 at 16:34
phuclv
41.6k ● 15 ● 177 ● 519

answered Sep 18, 2008 at 19:28
Epsilon
814 ● 6 ● 5

6 For those of you looking for a solution in Ruby, we're in luck! Christopher Lord has implemented the above algorithm in a Ruby gem. See christopher.lord.ac/fractions-in-ruby and rubygems.org/gems/fraction – shedd Jan 26, 2011 at 9:14

8 Be aware that there are some edge cases that this code does not handle very welll: when given -1.3333333 with a maximum denominator of 4 it returns 4/-3 with an error of

3.333333e-08 and -5/4 with an error = -8.333330e-02, which is correct. But when given -1.33333337 with the same maximum denominator, it turns 12121211/-9090908 with an error of error = 4.218847e-15 and -4/3 with an error of -3.666667e-08, which is not correct. This is an issue in particular when presenting the algorithm with computed floating point numbers such as -4/3, which yields incorrect results like these. – edsko Aug 1, 2011 at 8:26

33

From Python 2.6 on there is the `fractions` module.

(Quoting from the docs.)

```
>>> from fractions import Fraction
>>> Fraction('3.1415926535897932').limit_denominator(1
Fraction(355, 113)

>>> from math import pi, cos
>>> Fraction.from_float(cos(pi/3))
Fraction(4503599627370497, 9007199254740992)
>>> Fraction.from_float(cos(pi/3)).limit_denominator()
Fraction(1, 2)
```

Share Improve this answer Follow

edited Jan 21, 2022 at 16:46
phuclv
41.6k ● 15 ● 177 ● 519

answered Jan 2, 2010 at 19:16
Debilski
67.7k ● 12 ● 114 ● 133

7 Implementation and algorithm notes at hg.python.org/cpython/file/822c7c0d27d1/Lib/fractions.py#l211 – piro Mar 28, 2011 at 10:44

5 @Debilski which of the OP's `language agnostic` and `algorithm` tags does your answer satisfy? – vladr Nov 6, 2015 at 14:43

3 @vladr Well, given that I wrote this answer almost 6 years ago (and more than one year after the question had been asked), I guess I don't know anymore what my reasoning was back then. Most probably I was referring to this comment: stackoverflow.com/questions/95727/… OTOH It could also be that this answer has been merged from another question. Who can tell after all those years… – Debilski Nov 8, 2015 at 13:39

You could add a few sentences about the algorithm used by the fractions module (and update your answer for Python3 perhaps). – einpoklum Mar 26, 2016 at 23:48

24

If the the output is to give a human reader a fast impression of the order of the result, it makes no sense return something like "113/211", so the output should limit itself to using one-digit numbers (and maybe 1/10 and 9/10). If so, you can observe that there are only 27 *different* fractions.

Since the underlying math for generating the output will never change, a solution could be to simply hard-code a binary search tree, so that the function would perform at most log(27) ~= 4 3/4 comparisons. Here is a tested C version of the code

```
char *userTextForDouble(double d, char *rval)
{
    if (d == 0.0)
        return "0";
    // TODO: negative numbers:if (d < 0.0)...
    if (d >= 1.0)
        sprintf(rval, "%.0f ", floor(d));
    d = d-floor(d); // now only the fractional part is

    if (d == 0.0)
        return rval;

    if( d < 0.47 )
    {
        if( d < 0.25 )
        {
            if( d < 0.16 )
            {
                if( d < 0.12 ) // Note: fixed from .13
                {
                    if( d < 0.11 )
                        strcat(rval, "1/10"); // .1
                    else
                        strcat(rval, "1/9"); // .1111.
                }
                else // d >= .12
                {
                    if( d < 0.14 )
                        strcat(rval, "1/8"); // .125
                    else
                        strcat(rval, "1/7"); // .1428.
                }
            }
            else // d >= .16
            {
                if( d < 0.19 )
                {
                    strcat(rval, "1/6"); // .1666...
                }
                else // d > .19
                {
                    if( d < 0.22 )
                        strcat(rval, "1/5"); // .2
                    else
                        strcat(rval, "2/9"); // .2222.
                }
            }
        }
        else // d >= .25
        {
            if( d < 0.37 ) // Note: fixed from .38
            {
                if( d < 0.28 ) // Note: fixed from .29
                {
                    strcat(rval, "1/4"); // .25
                }
                else // d >=.28
                {
                    if( d < 0.31 )
                        strcat(rval, "2/7"); // .2857.
                    else
                        strcat(rval, "1/3"); // .3333.
                }
            }
            else // d >= .37
            {
                if( d < 0.42 ) // Note: fixed from .43
                {
                    if( d < 0.40 )
                        strcat(rval, "3/8"); // .375
                    else
                        strcat(rval, "2/5"); // .4
                }
                else // d >= .42
                {
                    if( d < 0.44 )
                        strcat(rval, "3/7"); // .4285.
                    else
                        strcat(rval, "4/9"); // .4444.
                }
            }
        }
    }
    else
    {
        if( d < 0.71 )
        {
            if( d < 0.60 )
            {
                if( d < 0.55 ) // Note: fixed from .56
                {
                    strcat(rval, "1/2"); // .5
                }
                else // d >= .55
                {
                    if( d < 0.57 )
                        strcat(rval, "5/9"); // .5555.
                    else
                        strcat(rval, "4/7"); // .5714
                }
            }
            else // d >= .6
            {
                if( d < 0.62 ) // Note: Fixed from .63
                {
                    strcat(rval, "3/5"); // .6
                }
                else // d >= .62
                {
                    if( d < 0.66 )
                        strcat(rval, "5/8"); // .625
                    else
                        strcat(rval, "2/3"); // .6666.
                }
            }
        }
        else
        {
            if( d < 0.80 )
            {
                if( d < 0.74 )
                {
                    strcat(rval, "5/7"); // .7142...
                }
                else // d >= .74
                {
                    if(d < 0.77 ) // Note: fixed from
                        strcat(rval, "3/4"); // .75
                    else
                        strcat(rval, "7/9"); // .7777.
                }
            }
            else // d >= .8
            {
                if( d < 0.85 ) // Note: fixed from .86
                {
                    if( d < 0.83 )
                        strcat(rval, "4/5"); // .8
                    else
                        strcat(rval, "5/6"); // .8333.
                }
                else // d >= .85
                {
                    if( d < 0.87 ) // Note: fixed from
                    {
                        strcat(rval, "6/7"); // .8571
                    }
                    else // d >= .87
                    {
                        if( d < 0.88 ) // Note: fixed
                        {
                            strcat(rval, "7/8"); // .8
                        }
                        else // d >= .88
                        {
                            if( d < 0.90 )
                                strcat(rval, "8/9"); /
                            else
                                strcat(rval, "9/10");
                        }
                    }
                }
            }
        }
    }

    return rval;
}
```

Share Improve this answer Follow

edited Jan 21, 2022 at 16:35

phuclv
41.6k ● 15 ● 177 ● 519

answered Sep 18, 2008 at 21:46

jpsecher
4,871 ● 2 ● 36 ● 43

3 This is the kind of lateral thinking we need more of! Excellent suggestion. – edsko Aug 1, 2011 at 8:40

1 Its a bit ugly but very fast and practical way – Bosak Nov 11, 2012 at 17:10

1 This is an interesting approach that's wonderfully simple. To save space you could instead binary search an array, or create a binary tree, but your approach is probably a little faster (you could save space by using a single call to strcat before return and assign a var where it's now called) . Also I would have included 3/10 and 7/10, but maybe that's just me. – jimhark Jan 20, 2013 at 5:43

1 Inspired by this solution, I've created a short (but totally unoptimized) code. It can easily be extended to cover a larger range of fractions. jsfiddle.net/PdL23/1 – Deepak Joy Cheenath Dec 9, 2013 at 10:41

2 Note that `1/1000` is also very humanly readable, but the above algorithm would only produce a very coarse `1/10` approximation; I believe that improvements can be made in terms of which humanly readable denominators one can pick from, and/or the addition of `<`, `>`, `<<`, `>>` prefixes to give an idea of the coarseness of the approximation. – vladr Nov 6, 2015 at 14:52

17

Here's a link explaining the math behind converting a decimal to a fraction:

http://www.webmath.com/dec2fract.html

And here's an example function for how to actually do it using VB (from www.freevbcode.com/ShowCode.asp?ID=582):

```
Public Function Dec2Frac(ByVal f As Double) As String

   Dim df As Double
   Dim lUpperPart As Long
   Dim lLowerPart As Long

   lUpperPart = 1
   lLowerPart = 1

   df = lUpperPart / lLowerPart
   While (df <> f)
      If (df < f) Then
         lUpperPart = lUpperPart + 1
      Else
         lLowerPart = lLowerPart + 1
         lUpperPart = f * lLowerPart
      End If
      df = lUpperPart / lLowerPart
   Wend
Dec2Frac = CStr(lUpperPart) & "/" & CStr(lLowerPart)
End Function
```

(From google searches: convert decimal to fraction, convert decimal to fraction code)

Share Improve this answer Follow

edited Jan 21, 2022 at 16:35

phuclv

41.6k ● 15 ● 177 ● 519

answered Sep 18, 2008 at 19:06

devinmoore
2,756 ● 3 ● 19 ● 14

5 Note this algorithm takes Ω(m) time when f = n/m . And that could be a lot, even if you didn't intend it to be (consider 0.66666666667). – einpoklum Mar 26, 2016 at 23:53

12

You might want to read What Every Computer Scientist Should Know about Floating Point Arithmetic.

You'll have to specify some precision by multiplying by a large number:

```
3.141592 * 1000000 = 3141592
```

then you can make a fraction:

```
3 + (141592 / 1000000)
```

and reduce via GCD...

```
3 + (17699 / 125000)
```

but there is no way to get the *intended* fraction out. You might want to *always* use fractions throughout your code instead --just remember to reduce fractions when you can to avoid overflow!

Share Improve this answer

Follow

edited Jul 26, 2021 at 18:42

Ghassan Maslamani
801 ● 8 ● 13

answered Sep 18, 2008 at 19:05

nlucaroni
47.9k ● 6 ● 66 ● 86

---

9

A C# implementation

```
/// <summary>
/// Represents a rational number
/// </summary>
public struct Fraction
{
    public int Numerator;
    public int Denominator;

    /// <summary>
    /// Constructor
    /// </summary>
    public Fraction(int numerator, int denominator)
    {
        this.Numerator = numerator;
        this.Denominator = denominator;
    }

    /// <summary>
    /// Approximates a fraction from the provided doub
    /// </summary>
    public static Fraction Parse(double d)
    {
        return ApproximateFraction(d);
    }

    /// <summary>
    /// Returns this fraction expressed as a double, r
number of decimal places.
    /// Returns double.NaN if denominator is zero
    /// </summary>
    public double ToDouble(int decimalPlaces)
    {
        if (this.Denominator == 0)
            return double.NaN;

        return System.Math.Round(
            Numerator / (double)Denominator,
            decimalPlaces
        );
    }


    /// <summary>
    /// Approximates the provided value to a fraction.
    /// http://stackoverflow.com/questions/95727/how-t
human-readable-fractions
    /// </summary>
    private static Fraction ApproximateFraction(double
    {
        const double EPSILON = .000001d;

        int n = 1;  // numerator
        int d = 1;  // denominator
        double fraction = n / d;

        while (System.Math.Abs(fraction - value) > EPS
        {
            if (fraction < value)
            {
                n++;
            }
            else
            {
                d++;
                n = (int)System.Math.Round(value * d);
            }

            fraction = n / (double)d;
        }

        return new Fraction(n, d);
    }
}
```

Share Improve this answer

Follow

edited Sep 24, 2012 at 12:26

Ivan Kochurkin
4,481 ● 8 ● 47 ● 84

answered Dec 21, 2011 at 2:07

Tom
2,036 ● 1 ● 22 ● 22

9

Here are Perl and Javascript versions of the VB code suggested by devinmoore:

Perl:

```
sub dec2frac {
    my $d = shift;

    my $df  = 1;
    my $top = 1;
    my $bot = 1;

    while ($df != $d) {
      if ($df < $d) {
        $top += 1;
      }
      else {
         $bot += 1;
         $top = int($d * $bot);
      }
      $df = $top / $bot;
   }
   return "$top/$bot";
}
```

And the almost identical javascript:

```javascript
function dec2frac(d) {

    var df = 1;
    var top = 1;
    var bot = 1;

    while (df != d) {
        if (df < d) {
            top += 1;
        }
        else {
            bot += 1;
            top = parseInt(d * bot);
        }
        df = top / bot;
    }
    return top + '/' + bot;
}

//Put in your test number here:
var floatNumber = 2.56;
alert(floatNumber + " = " + dec2frac(floatNumber));
```

Run code snippet   Expand snippet

Share  Improve this answer

Follow

edited Jan 21, 2022 at 16:50

phuclv

41.6k ● 15 ● 177 ● 519

answered Mar 25, 2009 at 13:17

mivk

14.7k ● 5 ● 83 ● 74

8

The Stern-Brocot Tree induces a fairly natural way to approximate real numbers by fractions with simple denominators.

Share Improve this answer Follow

answered Sep 19, 2008 at 2:27

Doug McClean
14.5k ● 6 ● 51 ● 70

---

6

Part of the problem is that so many fractions aren't actually easily construed as fractions. E.g. 0.33 isn't 1/3, it's 33/100. But if you remember your elementary school training, then there is a process of converting decimal values into fractions, however it's unlikely to give you what you want since most of the time decimal numbers aren't stored at 0.33, but 0.329999999999998 or some such.

Do yourself a favor and don't bother with this, but if you need to then you can do the following:

Multiply the original value by 10 until you remove the fractional part. Keep that number, and use it as the divisor. Then do a series of simplifications by looking for common denominators.

So 0.4 would be 4/10. You would then look for common divisors starting with low values, probably prime numbers. Starting with 2, you would see if 2 divides both the numerator and denominator evenly by checking if the floor of division is the same as the division itself.

```
floor(5/2) = 2
5/2 = 2.5
```

So 5 does not divide 2 evenly. So then you check the next number, say 3. You do this until you hit at or above the square root of the smaller number.

After you do that then you need

Share Improve this answer Follow

edited Sep 18, 2008 at 19:14

answered Sep 18, 2008 at 19:06

Orion Adrian
**19.5k** ● 13 ● 54 ● 67

1 I'd suggest using the euclidean algorithm for that last step – Graphics Noob Aug 25, 2009 at 22:52

6

This algorithm by Ian Richards / John Kennedy not only returns nice fractions, it also performs very well in terms of speed. This is C# code as taken from this answer by me.

It can handle all `double` values except special values like NaN and +/- infinity, which you'll have to add if needed.

It returns a `new Fraction(numerator, denominator)`. Replace by your own type.

**For more example values and a comparison with other algorithms, go here**

```
public Fraction RealToFraction(double value, double ac
{
    if (accuracy <= 0.0 || accuracy >= 1.0)
    {
        throw new ArgumentOutOfRangeException("accurac
1.");
    }

    int sign = Math.Sign(value);

    if (sign == -1)
    {
        value = Math.Abs(value);
    }

    // Accuracy is the maximum relative error; convert
    double maxError = sign == 0 ? accuracy : value * a

    int n = (int) Math.Floor(value);
    value -= n;

    if (value < maxError)
    {
        return new Fraction(sign * n, 1);
    }

    if (1 - maxError < value)
    {
        return new Fraction(sign * (n + 1), 1);
    }

    double z = value;
    int previousDenominator = 0;
    int denominator = 1;
    int numerator;

    do
    {
        z = 1.0 / (z - (int) z);
        int temp = denominator;
        denominator = denominator * (int) z + previous
        previousDenominator = temp;
        numerator = Convert.ToInt32(value * denominato
    }
    while (Math.Abs(value - (double) numerator / denom
!= (int) z);

    return new Fraction((n * denominator + numerator)
}
```

Example values returned by this algorithm:

```
Accuracy: 1.0E-3      | Richards
Input                 | Result           Error
======================|
=============================
   3                  |      3/1           0
   0.999999           |      1/1          1.0E-6
   1.000001           |      1/1         -1.0E-6
   0.50 (1/2)         |      1/2           0
   0.33... (1/3)      |      1/3           0
   0.67... (2/3)      |      2/3           0
   0.25 (1/4)         |      1/4           0
   0.11... (1/9)      |      1/9           0
   0.09... (1/11)     |      1/11          0
   0.62... (307/499)  |      8/13         2.5E-4
```

```
    0.14... (33/229)   |      16/111        2.7E-4
    0.05... (33/683)   |      10/207       -1.5E-4
    0.18... (100/541)  |      17/92        -3.3E-4
    0.06... (33/541)   |       5/82        -3.7E-4
    0.1                |       1/10          0
    0.2                |       1/5           0
    0.3                |       3/10          0
    0.4                |       2/5           0
    0.5                |       1/2           0
    0.6                |       3/5           0
    0.7                |       7/10          0
    0.8                |       4/5           0
    0.9                |       9/10          0
    0.01               |       1/100         0
    0.001              |       1/1000        0
    0.0001             |       1/10000       0
    0.33333333333      |       1/3          1.0E-11
    0.333              |     333/1000        0
    0.7777             |       7/9          1.0E-4
    0.11               |      10/91        -1.0E-3
    0.1111             |       1/9          1.0E-4
    3.14               |      22/7          9.1E-4
    3.14... (pi)       |      22/7          4.0E-4
```

Share Improve this answer

Follow

edited Jan 21, 2022 at 16:44

phuclv
41.6k ● 15 ● 177 ● 519

answered Feb 13, 2017 at 6:08

Kay Zed
1,376 ● 3 ● 21 ● 31

---

5

This is not an "algorithm", just a Python solution: http://docs.python.org/library/fractions.html

```
>>> from fractions import Fraction
>>> Fraction('3.141592653589793').limit_denominator(1
Fraction(355, 113)
```

Share Improve this answer

Follow

edited Jan 21, 2022 at 16:57

phuclv

41.6k ● 15 ● 177 ● 519

answered Aug 25, 2009 at 22:44

eldad

51 ● 1 ● 1

---

4

"Let's say we have 0.33, we need to output "1/3". "

What precision do you expect the "solution" to have? 0.33 is not equal to 1/3. How do you recognize a "good" (easy to read) answer?

No matter what, a possible algorithm could be:

If you expect to find a nearest fraction in a form X/Y where Y is less then 10, then you can loop though all 9 possible Ys, for each Y compute X, and then select the most accurate one.

Share Improve this answer

Follow

answered Sep 18, 2008 at 19:11

Suma

34.3k ● 18 ● 129 ● 199

3

You can do this in any programming language using the following steps:

1. Multiply and Divide by 10^x where x is the power of 10 required to make sure that the number has no decimal places remaining. Example: Multiply 0.33 by 10^2 = 100 to make it 33 and divide it by the same to get 33/100
2. Reduce the numerator and the denominator of the resulting fraction by factorization, till you can no longer obtain integers from the result.
3. The resulting reduced fraction should be your answer.

Example: 0.2 =0.2 x 10^1/10^1 =2/10 =1/5

So, that can be read as '1 part out of 5'

Share Improve this answer Follow

answered Sep 18, 2008 at 19:13

Pascal

4,127 ● 8 ● 34 ● 29

---

3

A built-in solution in R:

```
library(MASS)
fractions(0.666666666)
## [1] 2/3
```

This uses a continued fraction method and has optional `cycles` and `max.denominator` arguments for adjusting the precision.

Share Improve this answer Follow

edited Jan 21, 2022 at 16:58

phuclv
41.6k ● 15 ● 177 ● 519

answered Dec 29, 2012 at 20:09

Ben Bolker
225k ● 26 ● 396 ● 490

Also `library(numbers)` and `contFrac(0.6666)`; to get the string output as desired: `paste(contFrac(0.666, tol=1e-03)$rat, collapse="/")` – rbatt Jun 30, 2015 at 15:36

2

You'll have to figure out what level of error you're willing to accept. Not all decimal fractions will reduce to a simple fraction. I'd probably pick an easily-divisible number, like 60, and figure out how many 60ths is closest to the value, then simplify the fraction.

Share Improve this answer Follow

answered Sep 18, 2008 at 19:07

Mark Bessey
19.8k ● 4 ● 49 ● 69

2

One solution is to just store all numbers as rational numbers in the first place. There are libraries for rational number arithmetic (eg GMP). If using an OO language you may be able to just use a rational number class library to replace your number class.

Finance programs, among others, would use such a solution to be able to make exact calculations and preserve precision that may be lost using a plain float.

Of course it will be a lot slower so it may not be practical for you. Depends on how much calculations you need to do, and how important the precision is for you.

```
a = rational(1);
b = rational(3);
c = a / b;

print (c.asFraction)  --->  "1/3"
print (c.asFloat) ----> "0.333333"
```

Share Improve this answer Follow

answered Sep 18, 2008 at 22:16

robottobor
11.7k ● 12 ● 42 ● 38

---

2

> Let's say we have 0.33, we need to output "1/3".
> If we have "0.4", we need to output "2/5".

It's wrong in common case, because of 1/3 = 0.3333333 = 0.(3) Moreover, it's impossible to find out from suggested above solutions is decimal can be converted to fraction with defined precision, because output is always fraction.

BUT, i suggest my comprehensive function with many options based on idea of Infinite geometric series, specifically on formula:

$$\sum_{k=0}^{\infty} ar^k = \frac{a}{1-r} - 0 = \frac{a}{1-r}$$

At first this function is trying to find period of fraction in string representation. After that described above formula is applied.

Rational numbers code is borrowed from Stephen M. McKamey rational numbers implementation in C#. I hope there is not very hard to port my code on other languages.

```
/// <summary>
/// Convert decimal to fraction
/// </summary>
/// <param name="value">decimal value to convert</para
/// <param name="result">result fraction if conversati
/// <param name="decimalPlaces">precision of considere
value</param>
/// <param name="trimZeroes">trim zeroes on the right
not</param>
/// <param name="minPeriodRepeat">minimum period repea
/// <param name="digitsForReal">precision for determin
period has not been founded</param>
/// <returns></returns>
public static bool FromDecimal(decimal value, out Rati
    int decimalPlaces = 28, bool trimZeroes = false, d
2, int digitsForReal = 9)
```

```
{
    var valueStr = value.ToString("0.000000000000000000
CultureInfo.InvariantCulture);
    var strs = valueStr.Split('.');

    long intPart = long.Parse(strs[0]);
    string fracPartTrimEnd = strs[1].TrimEnd(new char[
    string fracPart;

    if (trimZeroes)
    {
        fracPart = fracPartTrimEnd;
        decimalPlaces = Math.Min(decimalPlaces, fracPa
    }
    else
        fracPart = strs[1];

    result = new Rational<T>();
    try
    {
        string periodPart;
        bool periodFound = false;

        int i;
        for (i = 0; i < fracPart.Length; i++)
        {
            if (fracPart[i] == '0' && i != 0)
                continue;

            for (int j = i + 1; j < fracPart.Length; j
            {
                periodPart = fracPart.Substring(i, j -
                periodFound = true;
                decimal periodRepeat = 1;
                decimal periodStep = 1.0m / periodPart
                var upperBound = Math.Min(fracPart.Len
                int k;
                for (k = i + periodPart.Length; k < up
                {
                    if (periodPart[(k - i) % periodPar
                    {
                        periodFound = false;
                        break;
                    }
                    periodRepeat += periodStep;
                }

                if (!periodFound && upperBound - k <=
periodPart[(upperBound - i) % periodPart.Length] > '5'
                {
                    var ind = (k - i) % periodPart.Len
                    var regroupedPeriod = (periodPart.
periodPart.Remove(ind)).Substring(0, upperBound - k);
                    ulong periodTailPlusOne = ulong.Pa
                    ulong fracTail = ulong.Parse(fracP
regroupedPeriod.Length));
                    if (periodTailPlusOne == fracTail)
                        periodFound = true;
                }

                if (periodFound && periodRepeat >= min
                {
                    result = FromDecimal(strs[0], frac
periodPart);
                    break;
                }
                else
                    periodFound = false;
            }

            if (periodFound)
                break;
        }

        if (!periodFound)
        {
            if (fracPartTrimEnd.Length >= digitsForRea
                return false;
            else
            {
                result = new Rational<T>(long.Parse(st
                if (fracPartTrimEnd.Length != 0)
                    result = new Rational<T>(ulong.Par
TenInPower(fracPartTrimEnd.Length));
                return true;
            }
        }
        return true;
    }
    catch
    {
        return false;
    }
}

public static Rational<T> FromDecimal(string intPart,
periodPart)
{
    Rational<T> firstFracPart;
    if (fracPart != null && fracPart.Length != 0)
    {
        ulong denominator = TenInPower(fracPart.Length
        firstFracPart = new Rational<T>(ulong.Parse(fr
    }
    else
        firstFracPart = new Rational<T>(0, 1, false);

    Rational<T> secondFracPart;
    if (periodPart != null && periodPart.Length != 0)
        secondFracPart =
            new Rational<T>(ulong.Parse(periodPart),
TenInPower(fracPart.Length)) *
            new Rational<T>(1, Nines((ulong)periodPart
    else
        secondFracPart = new Rational<T>(0, 1, false);

    var result = firstFracPart + secondFracPart;
    if (intPart != null && intPart.Length != 0)
    {
        long intPartLong = long.Parse(intPart);
        result = new Rational<T>(intPartLong, 1, false
: Math.Sign(intPartLong)) * result;
    }

    return result;
}

private static ulong TenInPower(int power)
{
    ulong result = 1;
    for (int l = 0; l < power; l++)
```

```
            result *= 10;
        return result;
    }

    private static decimal TenInNegPower(int power)
    {
        decimal result = 1;
        for (int l = 0; l > power; l--)
            result /= 10.0m;
        return result;
    }

    private static ulong Nines(ulong power)
    {
        ulong result = 9;
        if (power >= 0)
            for (ulong l = 0; l < power - 1; l++)
                result = result * 10 + 9;
        return result;
    }
```

There are some examples of usings:

```
Rational<long>.FromDecimal(0.33333333m, out r, 8, fals
// then r == 1 / 3;

Rational<long>.FromDecimal(0.33333333m, out r, 9, fals
// then r == 33333333 / 100000000;
```

Your case with right part zero part trimming:

```
Rational<long>.FromDecimal(0.33m, out r, 28, true);
// then r == 1 / 3;

Rational<long>.FromDecimal(0.33m, out r, 28, true);
// then r == 33 / 100;
```

Min period demostration:

```
Rational<long>.FromDecimal(0.123412m, out r, 28, true,
// then r == 1234 / 9999;
Rational<long>.FromDecimal(0.123412m, out r, 28, true,
// then r == 123412 / 1000000; because of minimu repea
0.1234123 in this case.
```

Rounding at the end:

```
Rational<long>.FromDecimal(0.88888888888888888888888888
// then r == 8 == 9;
```

The most interesting case:

```
Rational<long>.FromDecimal(0.12345678m, out r, 28, tru
// then r == 12345678 / 100000000;

Rational<long>.FromDecimal(0.12345678m, out r, 28, tru
// Conversation failed, because of period has not been
too many digits in fraction part of input value.

Rational<long>.FromDecimal(0.12121212121212121m, out r
// then r == 4 / 33; Despite of too many digits in inp
founded. Thus it's possible to convert value to fracti
```

Other tests and code everyone can find in my MathFunctions library on github.

Share Improve this answer Follow

edited Sep 24, 2012 at 12:24

answered Sep 24, 2012 at 12:17

Ivan Kochurkin
4,481 ● 8 ● 47 ● 84

2

I think the best way to do this is to first convert your float value to an ascii representation. In C++ you could use `ostringstream` or in C, you could use `sprintf`. Here's how it would look in C++:

```
ostringstream oss;
float num;
cin >> num;
oss << num;
string numStr = oss.str();
int i = numStr.length(), pow_ten = 0;
while (i > 0) {
    if (numStr[i] == '.')
        break;
    pow_ten++;
    i--;
}
for (int j = 1; j < pow_ten; j++) {
    num *= 10.0;
}
cout << static_cast<int>(num) << "/" << pow(10, pow_te
```

A similar approach could be taken in straight C.

Afterwards you would need to check that the fraction is in lowest terms. This algorithm will give a precise answer, i.e. 0.33 would output "33/100", not "1/3." However, 0.4 would give "4/10," which when reduced to lowest terms would be "2/5." This may not be as powerful as EppStein's solution, but I believe this is more straightforward.

Share Improve this answer Follow

edited Jan 21, 2022 at 16:35

phuclv
41.6k ● 15 ● 177 ● 519

answered Sep 17, 2011 at 19:35

bpm
105 ● 4

8 years later I come accross your solution, I've tested and it is working perfectly so far, but you said it is not as powerful as EppStein's solution and I wonder why. Since your solution is far more simple shouldn't this be the solution of choice, aren't we meant to do the simplest code possible as long as it works and it is safe?? – HBatalha Dec 10, 2019 at 10:08

Using `pow` for integer powers like this is a bad idea: Why pow(10,5) = 9,999 in C++, Why does pow(5,2) become 24? – phuclv Jan 21, 2022 at 16:45

2

Answer in C++, assuming that you have a `BigInt` class, which can store unlimited-size integers.

You can use `unsigned long long` instead, but it will only work for certain values.

```
void GetRational(double val)
{
    if (val == val+1) // Inf
        throw "Infinite Value";
    if (val != val) // NaN
        throw "Undefined Value";

    bool sign = false;
    BigInt enumerator = 0;
    BigInt denominator = 1;

    if (val < 0)
    {
        val = -val;
```

```
            sign = true;
        }

        while (val > 0)
        {
            unsigned int intVal = (unsigned int)val;
            val -= intVal;
            enumerator += intVal;
            val *= 2;
            enumerator *= 2;
            denominator *= 2;
        }

        BigInt gcd = GCD(enumerator,denominator);
        enumerator /= gcd;
        denominator /= gcd;

        Print(sign? "-":"+");
        Print(enumerator);
        Print("/");
        Print(denominator);

        // Or simply return {sign,enumerator,denominator}
    }
```

BTW, GetRational(0.0) will return "+0/1", so you might wanna handle this case separately.

P.S.: I've been using this code in my own 'RationalNum' class for several years, and it's been tested thoroughly.

Share Improve this answer Follow

edited Jan 21, 2022 at 16:43

phuclv
41.6k ● 15 ● 177 ● 519

answered Dec 30, 2013 at 6:42

barak manos
30.1k ● 10 ● 65 ● 115

Your example seems to break down on values like 1.333333.. it goes into a very long loop trying to find the value and does not seem to work... does fine with other simple values such as 1.25 – Adamski May 31, 2014 at 23:01

@Adamski: Thanks. The "convergence" period of the `while` loop is bounded by the size of `double`, which is typically 64 bits. So it does not depend on the initial value of the input (`val`). The `GCD` function, however, does depend on this value, although it usually converges to a solution pretty quick. Is it possible that you did not implement this function properly? – barak manos Jun 1, 2014 at 7:20

@Adamski: In addition, as I mentioned at the beginning of the answer, if you're using `unsigned long long` instead of `BigInt`, then it will not necessarily yield the correct result for every input value... But even under that scenario, the code is not supposed to "go into a very long loop".
– barak manos Jun 1, 2014 at 7:24

Ah ok yes, that is totally possible, the GCD function I was using is part of the Juce library BigInteger class. Thanks for the information! – Adamski Jun 1, 2014 at 9:24

@Adamski: So it **doesn't** make sense that the `GCD` function is not implemented properly. Have you checked if the code runs for a long time during the `while` loop or after it? I will check the value of 1.33333, to see what's behind this. Thanks. – barak manos Jun 1, 2014 at 9:47

2

Ruby already has a built in solution:

```
0.33.rationalize.to_s # => "33/100"
0.4.rationalize.to_s # => "2/5"
```

In Rails, ActiveRecord numerical attributes can be converted too:

```
product.size = 0.33
product.size.to_r.to_s # => "33/100"
```

Share Improve this answer Follow

edited Jan 21, 2022 at 16:44
phuclv
41.6k ● 15 ● 177 ● 519

answered Dec 8, 2012 at 4:29
Josh W Lewis
1,319 ● 11 ● 19

1

You are going to have two basic problems that will make this hard:

1) Floating point isn't an exact representation which means that if you have a fraction of "x/y" which results in a value of "z", your fraction algorithm may return a result other than "x/y".

2) There are infinity many more irrational numbers than rational. A rational number is one that can be represented as a fraction. Irrational being ones that can not.

However, in a cheap sort of way, since floating point has limit accuracy, then you can always represent it as some form of faction. (I think...)

Share Improve this answer answered Sep 18, 2008 at 19:06

Follow

Torlack
4,475 ● 1 ● 25 ● 24

5 A float (or double) *is* a fraction. Its denominator is a power of 2. That's why they can't exactly represent some rational numbers. – erickson Sep 18, 2008 at 20:35

1

Completed the above code and converted it to as3

```
public static function toFrac(f:Number) : String
    {
        if (f>1)
        {
            var parte1:int;
            var parte2:Number;
            var resultado:String;
            var loc:int = String(f).indexOf(".");
            parte2 = Number(String(f).slice(loc, Strin
            parte1 = int(String(f).slice(0,loc));
            resultado = toFrac(parte2);
            parte1 *= int(resultado.slice(resultado.in
resultado.length)) + int(resultado.slice(0, resultado.
            resultado = String(parte1) +
resultado.slice(resultado.indexOf("/"), resultado.leng
            return resultado;
        }
        if( f < 0.47 )
            if( f < 0.25 )
                if( f < 0.16 )
                    if( f < 0.13 )
                        if( f < 0.11 )
                            return "1/10";
                        else
                            return "1/9";
                    else
                        if( f < 0.14 )
                            return "1/8";
                        else
                            return "1/7";
                else
                    if( f < 0.19 )
                        return "1/6";
                    else
                        if( f < 0.22 )
                            return "1/5";
                        else
                            return "2/9";
            else
                if( f < 0.38 )
                    if( f < 0.29 )
                        return "1/4";
                    else
                        if( f < 0.31 )
                            return "2/7";
                        else
                            return "1/3";
                else
                    if( f < 0.43 )
                        if( f < 0.40 )
                            return "3/8";
                        else
                            return "2/5";
                    else
                        if( f < 0.44 )
                            return "3/7";
                        else
                            return "4/9";
        else
            if( f < 0.71 )
                if( f < 0.60 )
                    if( f < 0.56 )
                        return "1/2";
                    else
                        if( f < 0.57 )
                            return "5/9";
                        else
                            return "4/7";
                else
                    if( f < 0.63 )
                        return "3/5";
                    else
                        if( f < 0.66 )
                            return "5/8";
                        else
                            return "2/3";
            else
                if( f < 0.80 )
                    if( f < 0.74 )
                        return "5/7";
                    else
                        if(f < 0.78 )
                            return "3/4";
                        else
                            return "7/9";
                else
                    if( f < 0.86 )
                        if( f < 0.83 )
                            return "4/5";
                        else
                            return "5/6";
                    else
                        if( f < 0.88 )
                            return "6/7";
                        else
                            if( f < 0.89 )
                                return "7/8";
                            else
                                if( f < 0.90 )
                                    return "8/9";
                                else
                                    return "9/10";
    }
```

Share Improve this answer

Follow

edited Jan 21, 2022 at 16:42

phuclv

**41.6k** ● 15 ● 177 ● 519

answered Jan 2, 2010 at 18:59

João Lopes

**11** ● 1

Thanks, I used this for Delphi, easier to port than all that curly stuff – Peter Turner Sep 26, 2012 at 17:02

1

Here is a quick and dirty implementation in javascript that uses a brute force approach. Not at all optimized, it works within a predefined range of fractions: http://jsfiddle.net/PdL23/1/

```
/* This should convert any decimals to a simplified
specified by the two for loops. Haven't done any tho
to work fine.

I have set the bounds for numerator and denominator
increase this if you want in the two for loops.

Disclaimer: Its not at all optimized. (Feel free to
version.)
*/

decimalToSimplifiedFraction = function(n) {

for(num = 1; num < 20; num++) {  // "num" is the pot
    for(den = 1; den < 20; den++) {  // "den" is the
        var multiplyByInverse = (n * den ) / num;

        var roundingError = Math.round(multiplyByInv

        // Checking if we have found the inverse of
        if((Math.round(multiplyByInverse) == 1) && (
0.01)) {
            return num + "/" + den;
        }
    }
}
};

//Put in your test number here.
```

```
var floatNumber = 2.56;

alert(floatNumber + " = " + decimalToSimplifiedFract
```

Run code snippet   Expand snippet

This is inspired by the approach used by JPS.

Share Improve this answer Follow

edited Jan 21, 2022 at 16:48
phuclv
41.6k ● 15 ● 177 ● 519

answered Dec 9, 2013 at 10:40
Deepak Joy Cheenath
5,986 ● 5 ● 28 ● 31

---

0

As many people have stated you really can't convert a floating point back to a fraction (unless its extremely exact like .25). Of course you could create some type of look up for a large array of fractions and use some sort of fuzzy logic to produce the result you are looking for. Again this wouldn't be exact though and you would need to define a lower bounds of how large your want the denominator to go.

.32 < x < .34 = 1/3 or something like that.

Share Improve this answer Follow

answered Sep 18, 2008 at 19:17
Tim
1,476 ● 1 ● 17 ● 20

0

Here is implementation for ruby
http://github.com/valodzka/frac

```
Math.frac(0.2, 100)  # => (1/5)
Math.frac(0.33, 10)  # => (1/3)
Math.frac(0.33, 100) # => (33/100)
```

Share Improve this answer Follow

edited Jan 21, 2022 at 16:37
phuclv
41.6k ● 15 ● 177 ● 519

answered May 26, 2010 at 12:05
valodzka
5,785 ● 4 ● 40 ● 52

---

0

I came across an especially elegant Haskell solution making use of an anamorphism. It depends on the recursion-schemes package.

```
{-# LANGUAGE AllowAmbiguousTypes #-}
{-# LANGUAGE FlexibleContexts    #-}

import           Control.Applicative   (liftA2)
import           Control.Monad         (ap)
import           Data.Functor.Foldable
import           Data.Ratio            (Ratio, (%))

isInteger :: (RealFrac a) => a -> Bool
isInteger = ((==) <*>) (realToFrac . floor)

continuedFraction :: (RealFrac a) => a -> [Int]
continuedFraction = liftA2 (:) floor (ana coalgebra)
    where coalgebra x
              | isInteger x = Nil
              | otherwise = Cons (floor alpha) alpha
```

```
                   where alpha = 1 / (x - realToFrac (f

collapseFraction :: (Integral a) => [Int] -> Ratio a
collapseFraction [x]    = fromIntegral x % 1
collapseFraction (x:xs) = (fromIntegral x % 1) + 1 / c

-- | Use the nth convergent to approximate x
approximate :: (RealFrac a, Integral b) => a -> Int ->
approximate x n = collapseFraction $ take n (continued
```

If you try this out in ghci, it really does work!

```
λ:> approximate pi 2
22 % 7
```

Share Improve this answer

Follow

edited Jan 21, 2022 at 16:38

phuclv

41.6k ● 15 ● 177 ● 519

answered Sep 9, 2017 at 6:06

user8174234