Description
Rational() parses a float, an integer divided by an integer, and a float divided by an integer.

Rational("3.1") #=> (31/10)
Rational("3/2") #=> (3/2)
Rational("3.1/2") #=> (31/20)

But a float is not allowed as a denominator.

Rational("3.1/2.0") #=> ArgumentError

I'd expect the last also passes and results in (31/20), or the third also raises an ArgumentError

A patch to let all pass.
https://github.com/ruby/ruby/compare/trunk...nobu:parse_rat

Associated revisions
Revision a2ac0982 - 03/16/2017 03:35 AM - nobu (Nobuyoshi Nakada)
  rational.c: float denom
    • rational.c (parse_rat): allow float as a denominator as well as a numerator. [ruby-core:79104] [Bug #13134]
  git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57990 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 57990 - 03/16/2017 03:35 AM - nobu (Nobuyoshi Nakada)
  rational.c: float denom
    • rational.c (parse_rat): allow float as a denominator as well as a numerator. [ruby-core:79104] [Bug #13134]

Revision 57990 - 03/16/2017 03:35 AM - nobu (Nobuyoshi Nakada)
  rational.c: float denom
    • rational.c (parse_rat): allow float as a denominator as well as a numerator. [ruby-core:79104] [Bug #13134]

Revision 57990 - 03/16/2017 03:35 AM - nobu (Nobuyoshi Nakada)
  rational.c: float denom
    • rational.c (parse_rat): allow float as a denominator as well as a numerator. [ruby-core:79104] [Bug #13134]

History
#1 - 01/17/2017 08:59 AM - mrkn (Kenta Murata)
- Assignee set to mrkn (Kenta Murata)
- Status changed from Open to Assigned

#2 - 01/17/2017 09:33 PM - snood1205 (Eli Sadoff)
I propose leaving the behavior the way it is. A float denominator is filled with possible unexpected behavior. For example, one would hypothesize that (a/b).to_r == Rational("#a/#[b]") yet (3.1/2.0).to_r != Rational("3.1/2.0'"). The imprecision that can come with float denominators is why this behavior should not be allowed. Either we would have to allow for (a/b).to_r == Rational("#a/#[b]") to not always be true, or we would turn Rational("3.1/2.0") into (6980579422424269/4503599627370496) which defeats the purpose of the rational class.
#3 - 01/18/2017 12:48 AM - shyouhei (Shyouhei Urabe)

I don't think Rational("3.1/2.0") should introduce imprecision because the argument string has nothing to do with inexact numbers (at least seems to me).

So Eli's argument is ultimately "should we hold (a/b).to_r == Rational("#{a}/#{b}")?" and I think this is a valid argument. No opinion on this point though.

#4 - 01/18/2017 05:24 AM - stomar (Marcus Stollsteimer)

The imprecision that can come with float denominators is why this behavior should not be allowed.

Note there already is this kind of "unexpected" behavior:

```
2.4.0 :001 > (3.1/2).to_r == Rational("3.1/2")  # => false
```

or even

```
2.4.0 :002 > 1.2.to_r == Rational("1.2")  # => false
```

It's not caused by float denominators at all. So I don't see why strings with float denominator should raise an exception.

#5 - 01/18/2017 10:42 AM - Eregon (Benoit Daloze)

Floating-point numbers in String are exact, Float literal are inexact by nature.
So I would not expect any relation with to_r, even more so when float division is involved.
A Floating-point denominator in String sounds fine to me.

The equation that should hold is

Rational("#{a}/#{b}") == Rational(a) / Rational(b)

with a and b floating-point numbers as Strings.

Note that Float#rationalize produces a nicer-looking Rational, but there is no guarantee to how much precision might be lost on floating-point operations (e.g. 1.1-1.0), so it is moot to expect a relation between those.

#6 - 03/16/2017 03:35 AM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

Applied in changeset r57990.

---

rational.c: float denom
- rational.c (parse_rat): allow float as a denominator as well as a numerator. [ruby-core:79104] [Bug #13134]