Ruby master - Feature #3289
Division of negative numbers
05/14/2010 09:37 AM - patrick_thomson (Patrick Thomson)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>matz (Yukihiro Matsumoto)</td>
</tr>
<tr>
<td>Target version:</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Description

=end
The documentation for Numeric (http://www.ruby-doc.org/core/classes/Numeric.html#M000179) states that integer divmod() (and, by extension, /) rounds the quotient towards negative infinity. Python and Tcl behave similarly, while C, Java, bc, and gdb round the quotient towards zero, as is taught in standard arithmetic courses.

Is this a quirk of MRI's implementation, or is it desired Ruby behavior? If so, why?
It's counterintuitive that (-x/y) ≠ -(x/y), and even moreso when (-x/y) = -(x/y) if x or y is a non-integer.
=end

History

#1 - 05/14/2010 11:39 AM - shyouhei (Shyouhei Urabe)
=end
Japanese elementary schools tend to teach their children that "x / y is z remainder w" means

x == y * z + w, where 0 <= w < y.

So we follow it.
=end

#2 - 05/14/2010 09:50 PM - mame (Yusuke Endoh)
- Assignee set to matz (Yukihiro Matsumoto)
- Priority changed from Normal to 3
=end

2010/5/14 Patrick Thomson redmine@ruby-lang.org:

The documentation for Numeric (http://www.ruby-doc.org/core/classes/Numeric.html#M000179) states that integer divmod() (and, by extension, /) rounds the quotient towards negative infinity. Python and Tcl behave similarly, while C, Java, bc, and gdb round the quotient towards zero, as is taught in standard arithmetic courses.

It is very various. According to:

"Integer modulo operators in various programming languages"
http://en.wikipedia.org/wiki/Modulo_operation

J, Lua, Mathematica, Excel, Oberon, Perl, PL/I, Python, R, Ruby, Smalltalk, Tcl and Turing seem to behave the same.

I bet Ruby's division is taken after Perl.
Anyway, it is too late to change.

--
Yusuke Endoh mame@tsg.ne.jp
=end

#3 - 09/14/2010 04:00 PM - shyouhei (Shyouhei Urabe)
- Status changed from Open to Assigned
=end
#4 - 06/26/2011 01:47 PM - akr (Akira Tanaka)
- Project changed from Ruby to Ruby master
- Category changed from core to core

#5 - 10/30/2011 07:39 AM - alexeymuranov (Alexey Muranov)
I think that division with always rounding towards 0 is not logical. Rounding towards the closest integer, going towards 0 if there are 2 integers at equal distance, sounds better to me. This would be symmetric with respect to exchanging x and -x. The current behavior/definition of '/' is also an important operation. Maybe it can be renamed to 'div' and return both the quotient and the remainder in the "integral" sense (quotient being rounded towards '-linfty')?

Update: Sorry, i have not read the documentation! So divmod does exactly what one would expect it to do. The division '/' for integers is indeed questionable. I would suggest that it raise an exception or return some special value if two integers do not divide evenly. Apart from being nice in mathematical sense, this can help to avoid unexpected bugs.

-Alexey.

#6 - 11/20/2012 08:54 PM - mame (Yusuke Endoh)
- Description updated
- Target version set to 3.0

#7 - 04/26/2013 07:26 PM - Anonymous
Given the context, Ruby choice seems right to me. But the real core of the problem is the minus sign. My father says, that minus simply represents an unfinished operation, intention to subtract. Citing anonymous Wikipedia contributor:

"Negative numbers appeared for the first time in history in the Nine Chapters on the Mathematical Art, which in its present form dates from the period of the Chinese Han Dynasty (202 BC. - AD 220), but may well contain much older material. Indian mathematicians developed consistent and correct rules on the use of negative numbers, which later spread to the Middle East and then into Europe. Prior to the concept of negative numbers, negative solutions to problems were considered "false" and equations requiring negative solutions were described as absurd (Diophantus's Arithmetica cited)."

My father thinks, that Greek adherence to natural numbers, inherited by medieval Europe, was not due to their backwardness (they eg. knew that π was irrational), but because complicated objects catering to daily needs of bookkeepers were bad elementary blocks for general-purpose applied mathematics.

I would like to see basic math rebuilt without reaching for minus (and zero) too early. Relevant Ruby proposal would be: have true natural numbers (class Natural), have morally sound rationals (natural/ natural), and positive-only floats. Will I try to prototype this myself one day? Regarding zero, which has similar issues as minus and is absent from true natural numbers. I wonder how far would one get using nil (or null?) in its stead, like ancient Romans did...

#8 - 11/27/2017 07:41 AM - mame (Yusuke Endoh)
- Status changed from Assigned to Rejected

I believe this ticket is hopeless. Closing.