Ruby master - Bug #12958
Breaking change in how `#round` works
11/18/2016 08:32 PM - rafaelfranca (Rafael França)

Status: Closed
Priority: Normal
Assignee: mrkn (Kenta Murata)
Target version: ruby -v:
Backport: 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description
We noticed in the Rails test suite that there is a breaking change in how #round works between 2.3 and 2.4

https://github.com/rails/rails/pull/27091

Is that desirable?
I think it is may cause a lot of problem if the behavior of #round without any arguments changes between a minor version.

Related issues:
Related to Ruby master - Bug #12548: Rounding modes inconsistency between rou...
Closed

Associated revisions
Revision f72be814 - 12/10/2016 02:36 AM - mrkn (Kenta Murata)
internal.h: change the default rounding mode to half-up

- internal.h (ROUND_DEFAULT): changed to RUBY_NUM_ROUND_HALF_UP.
  [Bug #12958] [ruby-core:78204]
- test/ruby/test_integer.rb: fix assertions for the above change.
- test/ruby/test_rational.rb: ditto.
- test/test_mathn.rb: ditto.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57038 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 57038 - 12/10/2016 02:36 AM - mrkn (Kenta Murata)
internal.h: change the default rounding mode to half-up

- internal.h (ROUND_DEFAULT): changed to RUBY_NUM_ROUND_HALF_UP.
  [Bug #12958] [ruby-core:78204]
- test/ruby/test_integer.rb: fix assertions for the above change.
- test/ruby/test_rational.rb: ditto.
- test/test_mathn.rb: ditto.

Revision 57038 - 12/10/2016 02:36 AM - mrkn (Kenta Murata)
internal.h: change the default rounding mode to half-up

- internal.h (ROUND_DEFAULT): changed to RUBY_NUM_ROUND_HALF_UP.
  [Bug #12958] [ruby-core:78204]
- test/ruby/test_integer.rb: fix assertions for the above change.
- test/ruby/test_rational.rb: ditto.
- test/test_mathn.rb: ditto.
internal.h: change the default rounding mode to half-up

- internal.h (ROUND_DEFAULT): changed to RUBY_NUM_ROUND_HALF_UP.
  [Bug #12958] [ruby-core:78204]
- test/ruby/test_integer.rb: fix assertions for the above change.
- test/ruby/test_rational.rb: ditto.
- test/test_mathn.rb: ditto.

Revision 57038 - 12/10/2016 02:36 AM - mrkn (Kenta Murata)

internal.h: change the default rounding mode to half-up

- internal.h (ROUND_DEFAULT): changed to RUBY_NUM_ROUND_HALF_UP.
  [Bug #12958] [ruby-core:78204]
- test/ruby/test_integer.rb: fix assertions for the above change.
- test/ruby/test_rational.rb: ditto.
- test/test_mathn.rb: ditto.

History

#1 - 11/18/2016 08:40 PM - Eregon (Benoit Daloze)
- Related to Bug #12548: Rounding modes inconsistency between round versus sprintf added

#2 - 11/18/2016 08:43 PM - Eregon (Benoit Daloze)
This is an intended change, see #12548 and
https://github.com/ruby/ruby/blob/8d7c380216809ba5bd4a3eeec41d7dda81825ffa/NEWS#L73-L74

#3 - 11/18/2016 09:53 PM - rafaelfranca (Rafael França)
Thank you for the links. While I agree with the reasoning in #12438, I believe changing the default behavior is dangerous in a minor version.
The downside of changing the default behavior is that now applications using round without argument will have their behavior changes silently.

#4 - 11/19/2016 05:46 PM - shevegen (Robert A. Heiler)
Also see the discussion in https://bugs.ruby-lang.org/issues/12548 - ruby version changed from 2.3.x to 2.4.x, so I think the change is fine. Default similar behaviour can be expected e.g. from 2.3.1 to 2.3.9 and such. Otherwise you'd have to release ruby 3.x when wanting to do any change that alters behaviour - and this will be years in the future. I personally rather like to have a great xmas ruby but I also understand that it adds burden to adapt tests.

Would perhaps be useful if there could be some additional classifications or filtering to have a look what behaviour was changed, so that others can anticipate in time. Kinda a bit like how github uses preset default tags and templates for "bug" or "enhancement" and such classifications. Then one could have a classification tag in addition to this stating something like "API behaviour change" or something like that, and people could scan for these between different ruby releases. This could make it a bit easier to write tests and anticipate what is coming next.

#5 - 11/25/2016 07:41 AM - shyouhei (Shyouhei Urabe)
- Status changed from Open to Assigned
- Assignee set to mrkn (Kenta Murata)

#6 - 11/25/2016 09:42 AM - rafaelfranca (Rafael França)
Tests are not the only problem. I believe this change is dangerous because this code may change behavior in production without being easily noticed.

#7 - 11/25/2016 01:13 PM - shyouhei (Shyouhei Urabe)
Of course it changes behaviour. A change always is. The problem is how vital this (rounding an inexact number to a yet more inexact number) is.

As far as I can see the Rails breakage only happens in views, where 30secs is shown "less than a minute" instead of "1 minute". That sounds very trivial to me. And yet, did not break silently.

#8 - 11/29/2016 03:14 PM - jeromecornet (Jerome Cornet)
Shyouhei Urabe wrote:

As far as I can see the Rails breakage only happens in views, where 30secs is shown "less than a minute" instead of "1 minute". That sounds very trivial to me. And yet, did not break silently.

Actually, any money-related calculation (penny rounding) would now be silently changing behavior, leading to weird questions as to why the math doesn't add up to the penny.

While consistency is a good thing, I'm not sure why we can't make the default the old behavior, and change sprintf to explicitly call the new behavior.

#9 - 11/30/2016 12:53 AM - shyouhei (Shyouhei Urabe)
Jerome Cornet wrote:

Shyouhei Urabe wrote:

As far as I can see the Rails breakage only happens in views, where 30secs is shown "less than a minute" instead of "1 minute". That sounds very trivial to me. And yet, did not break silently.

Actually, any money-related calculation (penny rounding) would now be silently changing behavior, leading to weird questions as to why the math doesn't add up to the penny.

The new default is called the "banker's round" for reasons. Money related people should prefer this. And to say frankly, using Floats for money-related calculation is the source of weirdness.

While consistency is a good thing, I'm not sure why we can't make the default the old behavior, and change sprintf to explicitly call the new behavior.

sprintf has no option field to specify tie-breaking mode. You can't but change round instead.

#10 - 11/30/2016 01:17 AM - jeromecornet (Jerome Cornet)
Unfortunately there are many instances where money math requires floating point calculation, like taxes for example (discounts are also another source of float-related money math)


Banker's rounding is actually a poorly named rounding method, as it is only used outside the financial industry. It has the nice property for statistical analysis that it has no bias over the entire numbers space; it's better named as Gaussian rounding.

There are many companies in the financial industry that use ruby, making a change like this is actually a bigger deal than just breaking rails views.

#11 - 11/30/2016 01:53 AM - shyouhei (Shyouhei Urabe)
Jerome Cornet wrote:

Unfortunately there are many instances where money math requires floating point calculation, like taxes for example (discounts are also another source of float-related money math)

I would like to disagree.

It is true that money math tends to require decimal calculation. However I have never seen an example where "floating point" calculation is mandatory.

Am I making myself understood? I am distinguishing Floats and BigDecimals. We did not change how a BigDecimal rounds. Money related calculations should use that class instead. Not Float. Using Float for money is in fact toxic; for instance you can't express $0.1 using Float.

They are all examples of decimal calculations, not floating points.

Also, as I reported in Feature #12953, there do exist a case where round-half-down is requested by a government. That doesn't mean we don't need round-half-up mode but at least it shows that the mode is not the only thing we need.

Banker's rounding is actually a poorly named rounding method, as it is only used outside the financial industry. It has the nice property for statistical analysis that it has no bias over the entire numbers space; it's better named as Gaussian rounding.

(I have no strong opinion over how it should be called. I just pointed the status quo.)

There are many companies in the financial industry that use ruby, making a change like this is actually a bigger deal than just breaking rails views.

I would like to believe that no serious money calculation is done using Floats. We have provided BigDecimal for a long time for that purpose. Even when someone use Float for some reason, I can hardly believe they do so without careful tests.

#12 - 11/30/2016 02:43 AM - jeromecornet (Jerome Cornet)

Am I making myself understood? I am distinguishing Floats and BigDecimals. We did not change how a BigDecimal rounds. Money related calculations should use that class instead. Not Float. Using Float for money is in fact toxic; for instance you can't express $0.1 using Float.

Oh I'm sorry, my bad, I thought the change changed rounding for everything, not just Float (I misread the initial change, and agreed, floats are toxic for money)

But does this mean that Float rounds in a different way as BigDecimal by default then?
So the patch made Float rounding consistent with sprintf rouding, but by doing so it made default Float rounding inconsistent with default BigDecimal rounding.
Is my understanding correct?

#13 - 11/30/2016 03:16 AM - shyouhei (Shyouhei Urabe)

Jerome Cornet wrote:

But does this mean that Float rounds in a different way as BigDecimal by default then?
So the patch made Float rounding consistent with sprintf rouding, but by doing so it made default Float rounding inconsistent with default BigDecimal rounding.
Is my understanding correct?

Correct. It is true that we fixed one inconsistency to introduce another. This might perhaps be a problem. Not sure how big the problem would be though.

There is no BigDecimal-versus-sprintf problem because you cannot pass a BigDecimal to sprintf. So I believe the disturbance (if any) would be faint. But yes, I admit I might be wrong here.

#14 - 11/30/2016 04:26 AM - duerst (Martin Dürst)

Shyouhei Urabe wrote:

The new default is called the "banker's round" for reasons. Money related people should prefer this.

We don't know to what extent such an argument applies around the world.

And to say frankly, using Floats for money-related calculation is the source of weirdness.

Definitely. But while we can tell everybody to use BigDecimal, we don't know how many people are using Float, and maybe use some special tricks to make sure they get the rounding they need (which may be confused by our planned changes).

And money is one of the areas (another is security) where any kind of unexpected changes are a bad thing.

#15 - 12/08/2016 02:14 PM - jduff (John Duff)
It is true that we fixed one inconsistency to introduce another.

The new inconsistency seems worse to me. Having Numeric classes like BigDecimal and Float being consistent is more important in my opinion. People will tend to use these in similar places and seeing inconsistencies will be confusing.

I also agree that Float shouldn't be used for Money, but I am sure people are using it. This change could have some pretty terrible real world consequences while not really making anything that much better (trading one inconsistency for another, worse inconsistency imo).

---

#16 - 12/09/2016 03:31 PM - meta (mathew murphy)

For the record, I think that the new choice of default is poor.

People have a normal everyday expectation for how numbers are rounded. I claim that the expectation is that $\geq .5$ rounds up, $<.5$ rounds down.

I cite as evidence:

- It's the rule required by the US Internal Revenue Service.
- It's the behavior of Math.round() in JavaScript.
- It's the behavior of the C standard library round() function as required by POSIX.
- It's the behavior of Math.round in Java.
- It's the default behavior of PHP.
- It's the default for Perl.
- It's the behavior for Swift.
- It's what MATLAB does.
- It's what SQL does.
- It's what COBOL does.
- It's what Rust does.

It's also what I was taught throughout school, until I studied computer science and bookkeeping and learned about banker's rounding and statistical bias and other advanced topics.

So in keeping with the principle of least surprise, I think the default for #round should be to do what the man on the street would expect, and what programmers used to almost every other programming language would expect.

In checking all the most popular programming languages, I can find only three examples which do round-to-even by default: Python, R, and Visual BASIC. VB notes that "Rounding away from zero is the most widely known form of rounding". R is infamously user-hostile.

Fun sidenote: When Go discussed adding a round method, the comment was made "It's not obvious to me that TiesToEven is useful in the context of integral rounding." I agree with this. I have literally never needed round-to-even for floating point. Yes, it eliminates consistent bias in financial rounding, but nobody should be using floats for financial data to start with.

My feeling is that the rounding behavior of sprintf is unexpected and should be documented; and given that it's documented, it's unnecessary and unexpected for the rest of Ruby to conform to sprintf's behavior and not to POSIX.

---

#17 - 12/09/2016 07:25 PM - chrisccerami (Chris Cerami)

I would echo 100% of what mathew murphy wrote, and also add that this negatively affects ergonomics for the vast majority of cases where one wants to not use banker's rounding. I now need to know that Ruby rounds Floats differently than almost every other context, and then need to use

```
f.round(half: :up)
```

to get the expected behavior rather than simply f.round. It makes far more sense to make banker's rounding require the args and maintain backwards compatibility with the default behavior.

Not to mention the fact that this is a breaking change introduced in a minor version.

---

#18 - 12/10/2016 02:20 AM - matz (Yukihiro Matsumoto)

OK, the default behavior should be kept unchanged.

Matz.

---

#19 - 12/10/2016 02:36 AM - mrkn (Kenta Murata)

- Status changed from Assigned to Closed

Applied in changeset r57038.

---

internal.h: change the default rounding mode to half-up

- internal.h (ROUND_DEFAULT): changed to RUBY_NUM_ROUND_HALF_UP.
  [Bug #12958] [ruby-core:78204]
• test/ruby/test_integer.rb: fix assertions for the above change.
• test/ruby/test_rational.rb: ditto.
• test/test_mathn.rb: ditto.