Ruby - Bug #14588

math library functions should NOT raise exceptions

03/08/2018 05:24 AM - Anon92929 (Anon Ymous)

Status:	Rejected		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

BigDecimal, Integer, Float, none of these should raise exceptions, but they should all fail SILENTLY or return NaN during error cases.

I shouldn't have to go fix every possible way that BigDecimal might throw a breaking change in a thousand places in my codebase. I need math libraries that DON'T BREAK!!!

BUG RESUBMITTED.

The bug was rejected at github because I was referred to open a ticket at ruby-lang.org.

The bug was rejectet at ruby-lang.org because the ticket was closed at github.

Not cool, guys.

Related issues:

Is duplicate of Ruby - Bug #14587: math library functions should NOT raise ex...	**Rejected**

History

#1 - 03/08/2018 05:25 AM - Anon92929 (Anon Ymous)

This just represents another way that the mentality of the Ruby programmers seems "sticky", to me.

No implicit conversion of int to string!!! We REFUSE to attempt .to_s automatically, and we INSIST on wasting everyone's time with this silly bug, forever!!!

Oh, you need to cast to a long decimal format? NOPE SORRY your code is broken now because "." is not technically a number, go spend a day trying to figure it out sucker! Your github issue has been CLOSED!!!

Is this how the ruby devs and community intend to win over new young coders? By making a language where everything breaks, there's no logical remediations even attempted, and you can't even trust simple MATH LIBRARIES not to halt your codebase on every version update?

#2 - 03/08/2018 05:33 AM - Anon92929 (Anon Ymous)

I've been BANNED from the ruby github repo, for reporting this bug. Wow. You guys are truly insane.

#3 - 03/08/2018 05:42 AM - Anon92929 (Anon Ymous)

To me, passing a breaking error back up to the top level and saying "oops! we couldn't figure out what to do!" is a BUG in any program. You had one job. Cast to integer, literally, shouldn't be that hard to classify error cases here. The fact that this wasn't even recognized as a valid bug is crazy to me. I don't understand programmers anymore. Passing a breaking change up and out of scope is a BUG! What's wrong with you people!?

The fact that I was literally banned from a FOSS repo after my ticket being force-closed should be a warning bell to every programmer who contributes here.

#4 - 03/08/2018 05:46 AM - shan (Shannon Skipper)

What Github issue and bugs.ruby-lang.org issue are you referring to?

> We REFUSE to attempt .to_s automatically

It seems you're wanting Ruby to be weakly typed. Ruby is strongly typed and is likely to stay that way.

> "." is not technically a number

Did you think a period was a number? I'm confused what you mean by this. Maybe a link to the issues you're referring to will clear this up.

#5 - 03/08/2018 05:56 AM - Anon92929 (Anon Ymous)

> What Github issue and bugs.ruby-lang.org issue are you referring to?
>
> > We REFUSE to attempt .to_s automatically
>
> It seems you're wanting Ruby to be weakly typed. Ruby is strongly typed and is likely to stay that way.

Strong typing is not the issue, the issue is codebase FRAGILITY, and ruby is becoming increasingly fragile. Strong typing is fine, but the problem is that every time there's any sort of error, it throws an assert, and the number of possible errors is increasing over time. For example, what should it do when it fails, during a puts or print statement, to convert an int to a string, for printing? Should that cause an assert, or should it, intelligently, just print the output statement? How many people's hours have been wasted trying to make sure that print statements don't assert?

> "." is not technically a number
> Did you think a period was a number? I'm confused what you mean by this. Maybe a link to the issues you're referring to will clear this up.

To me, this should return 0, or NaN. What will happen in reality? It will break and then you will have to customize your codebase to handle that one dataset that had a stray ".", because everything halted and the program asserted instead of giving a warning. Now every time I update any math library I have to worry about whether there will be new types of unhandled, breaking changes to the way NaN is handled in three separate sub-libraries. Great work.

#6 - 03/08/2018 05:58 AM - Anon92929 (Anon Ymous)

Maybe the issue is too many asserts and not enough warnings. Maybe these stray NaN cases should be warnings instead of asserts. Maybe puts/print should know how to print integers.

#7 - 03/08/2018 06:09 AM - Anon92929 (Anon Ymous)

Codebases that become more fragile over time will lose old users and fail to attract new ones. Passing asserts up and out of scope in new and fantastic new ways should be considered a BUG. These math libraries need to stay quiet and just warn about NaN or whatever, when they fail.

#8 - 03/08/2018 06:13 AM - Anon92929 (Anon Ymous)

Sorry, sir, the spaceship crashed because BigDecimal asserted, when a "." was passed to the orbital processor, causing the entire Arithmetic Logic Unit to cease operating. If only it had WARNED us instead of refusing to calculate any further datasets!

#9 - 03/08/2018 06:29 AM - duerst (Martin Dürst)

Two points of advice:

1. I was not able to find the issue you mention on github. Please provide a pointer.
2. If you think others have been rude to you, that's not a reason to shout at everybody here (most of us have heard about this issue for the first time here).

#10 - 03/08/2018 06:36 AM - Anon92929 (Anon Ymous)

https://github.com/ruby/bigdecimal/issues/95

I request to be unbanned from the FOSS repo. I won't bug your BigDecimal devs anymore, but seriously, I shouldn't have to worry about new and exotic NAN exceptions that ASSERT any time I update a math library.

#11 - 03/08/2018 06:39 AM - shan (Shannon Skipper)

Kernel methods like Float(), Integer(), and BigDecimal() are all explicitly for the purpose of ensure strict conformance with numeric representation. Each of these Kernel methods raise a TypeError when the argument does not strictly conform. That's how they work. That's how they're meant to work. If you're using them when you don't want strict conformance, you're doing it wrong. As pointed out in the Github issue, the proper method to use for your purpose is #to_d.

#12 - 03/08/2018 06:44 AM - Anon92929 (Anon Ymous)

My points all stand. Please unban me.

#13 - 03/08/2018 06:45 AM - Anon92929 (Anon Ymous)

Or is this like a personal vendetta thing for you now?

#14 - 03/08/2018 06:48 AM - Anon92929 (Anon Ymous)

I'm so sorry, that I reported, as a bug, the fact that my code started breaking because I updated BigDecimal and it choked on a random NaN.

#15 - 03/08/2018 06:49 AM - Anon92929 (Anon Ymous)

The whole point was meant to be about code fragility and backwards compatility. You guys missed it.

#16 - 03/08/2018 07:01 AM - shan (Shannon Skipper)

Your behavior isn't acceptable. Do not open new issues when your issue is closed and locked. It is very bad manners.

> I'm so sorry, that I reported, as a bug, the fact that my code started breaking because I updated BigDecimal and it choked on a random NaN.

```
BigDecimal::VERSION #=> "1.3.4"
BigDecimal(Float::NAN) #=> NaN
```

If you'd like to learn Ruby, and understand why it is how it is, we'd be happy to help in the #ruby channel on freenode.net. Or you could ask on the mailing list. What you're doing here isn't a good way to learn. Harassing maintainers will get you banned from Github entirely.

This issue should be closed as a duplicate of #14587.

#17 - 03/08/2018 07:09 AM - Anon92929 (Anon Ymous)

My points all stand. Please unban me.

#18 - 03/08/2018 07:10 AM - Anon92929 (Anon Ymous)

Or is this like a personal vendetta thing for you now?

#19 - 03/08/2018 07:13 AM - Anon92929 (Anon Ymous)

BigDecimal(".")
ArgumentError: invalid value for BigDecimal(): "."

#20 - 03/08/2018 07:19 AM - Anon92929 (Anon Ymous)

I suggest that those who think that passing new types of exceptions upstream as asserts is not a BUG are the ones who need to learn programming, not myself. Force-closing actual bug reports and banning users because you think your little math sub-library should feature that new breaking "assert" statement, which breaks peoples' codebases is a symptom of more than just one little "." that broke the camel's back.

#21 - 03/08/2018 07:36 AM - Anon92929 (Anon Ymous)

> Kernel methods like Float(), Integer(), and BigDecimal() are all explicitly for the purpose of ensure strict conformance with numeric representation.

Are you a robot? NaN should not be an exception. No form of NaN should an exception, it should just be NaN. Let NaN be NaN. I don't want to worry about all the new ways NaN could be NaN when I 'bundle update'.

#22 - 03/08/2018 08:03 AM - Anon92929 (Anon Ymous)

Your behavior isn't acceptable.

Going on personal vendettas and threatening to get users banned from github because you didn't like how they re-opened a ticket you closed is not acceptable.

#23 - 03/08/2018 10:09 AM - Anon92929 (Anon Ymous)

I don't even care anymore. I just hate the guts of the ruby devs now. I'll make sure to make all my bug reports to your repos as anonymous users from now on so you can't try to get me banned in revenge for pointing out your increasingly fragile codebase.

#24 - 03/31/2018 10:40 AM - nobu (Nobuyoshi Nakada)

- Is duplicate of Bug #14587: math library functions should NOT raise exceptions added

#25 - 03/31/2018 10:56 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

Anon92929 (Anon Ymous) wrote:

> NaN should not be an exception. No form of NaN should an exception, it should just be NaN.

Do you mean "quiet NaN"?

Ruby uses signaling NaN by "fail-earlier" design.
All calculations which NaN involves result in NaN, and it's hard to tell from where it came.
Maybe it could be enough to fallback into NaN in your particular use case, but it isn't in general.