Ruby master - Bug #17098

Float#negative? reports negative zero as not negative

08/01/2020 01:05 AM - chrisseaton (Chris Seaton)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

Is this intended behaviour?

```
irb(main):001:0> neg_zero = -0.0
=> -0.0
irb(main):002:0> neg_zero.negative?
=> false
irb(main):003:0> neg_zero < 0
=> false
```

It happens because Numeric#negative? uses < 0. My understanding of IEEE floating point is that negative zero is not less than zero, but I think it should still report as negative.

History

#1 - 08/01/2020 02:18 AM - jeremyevans0 (Jeremy Evans)

I disagree. By that logic, positive float zero should be positive (it isn't), and since -0.0 == 0.0 both should report as negative and positive, which is a contradiction.

#2 - 08/01/2020 12:18 PM - chrisseaton (Chris Seaton)

It's already the case that neg_zero == pos_zero but not all methods on the two returns the same result - for example neg_zero.inspect != pos_zero.inspect.

#3 - 08/01/2020 12:38 PM - sawa (Tsuyoshi Sawada)

chrisseaton (Chris Seaton) wrote in #note-2:

> not all methods on the two [return] the same result - for example neg_zero.inspect != pos_zero.inspect.

What is your suggestion? To dispense with -0.0?

#4 - 08/01/2020 12:50 PM - chrisseaton (Chris Seaton)

My suggestion is that Float#negative?(neg_zero) == true and Float#positive?(pos_zero) == false.

We can't get rid of negative zero - it's a standard part of floating point numbers as implemented by the hardware. It's also useful in some applications.

#5 - 08/01/2020 04:00 PM - chrisseaton (Chris Seaton)

If people aren't keen on changing Float#negative? since it has existing semantics, then another option could be to add a new predicate Float#negative_zero? - that would allow people to differentiate as needed.

#6 - 08/01/2020 06:02 PM - Eregon (Benoit Daloze)

chrisseaton (Chris Seaton) wrote in #note-5:

> then another option could be to add a new predicate Float#negative_zero? - that would allow people to differentiate as needed.

Interesting, I thought equal? would work for that (it does on TruffleRuby) but it does not on CRuby 2.6.6:

```
[19] pry(main)> 0.0.equal?(0.0)
=> true
[20] pry(main)> -0.0.equal?(-0.0)
```

```
=> false
```

I guess it's a result of implementing Float as tagged (flonum) but only for a subset of Float:

```
[25] pry(main)> a=-0.0
=> -0.0
[26] pry(main)> a.equal?(a)
=> true
```

eql? and == do not differentiate 0.0 and -0.0.

One way to test for -0.0 seems (-f).equal?(0.0) but that's kind of brittle as it relies on 0.0 being a flonum on CRuby.

I tend to agree that 0.0.positive? == -0.0.negative? and since 0.positive? => false then both as false seems to make sense (essentially they are the same as > 0 and < 0).

#7 - 08/01/2020 07:41 PM - marcandre (Marc-Andre Lafortune)

AFAIK, the only way to check for -0.0 is 1.0 / var == -Float::INFINITY

I find it difficult to discuss what need to be done about any of this as I do not know of the use cases for -0.0; without them it seems very theoretical.

#8 - 08/02/2020 12:21 PM - sawa (Tsuyoshi Sawada)

Ruby or IEEE 754 seem to regard -0.0 and 0.0 not as 0 in the mathematical sense, but as (something like) negative and positive infinitesimal. Taking this into account, I started to think -0.0.negative? and 0.0.positive? should both be true.

Regarding the fact that -0.0 == 0.0 is true, I think it should be understood as '-0.0 approaches 0.0' rather than '-0.0 is 0.0'. Then it would not contradict with the above. Floating point numbers are approximated numbers to begin with, so it does not make much sense to talk about their exact identity. Hence it makes sense to regard Float#== to mean 'close enough' rather than 'exactly the same'.