Ruby master - Bug #1336

Change in string representation of Floats

03/31/2009 02:21 PM - brixen (Brian Shirai)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:	2.0.0		
ruby -v:	ruby 1.9.2dev (2009-03-30 trunk 23098) [i386-darwin9.6.0]	**Backport:**	

Description

=begin
Did the displayed formats of Floats change in 1.9?

There was a thread [ruby-core:22595] on the ML about the format of Floats and marshaling, but there does not appear to be any indication that the normal display format would change. Also, I did not see any mention of a change of format in the Changelog.

```
gauss:mspec brian$ ~/devel/ruby19/install/bin/ruby19 -v -e 'p 5.51'
ruby 1.9.2dev (2009-02-12 trunk 22247) [i386-darwin9.6.0]
5.51

gauss:mspec brian$ ruby1.9 -v -e 'p 5.51'
ruby 1.9.2dev (2009-03-30 trunk 23098) [i386-darwin9.6.0]
5.5099999999999998

gauss:mspec brian$ ruby1.9 -v -e 'puts 5.51'
ruby 1.9.2dev (2009-03-30 trunk 23098) [i386-darwin9.6.0]
5.5099999999999998
```
=end

History

#1 - 03/31/2009 02:54 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Tue, 31 Mar 2009 14:21:15 +0900,
Brian Ford wrote in [ruby-core:23075]:

> Did the displayed formats of Floats change in 1.9?

Yes.

> There was a thread [ruby-core:22595] on the ML about the
> format of Floats and marshaling, but there does not appear to
> be any indication that the normal display format would
> change. Also, I did not see any mention of a change of format
> in the Changelog.

It's not about marshaling.

--
Nobu Nakada

=end

#2 - 03/31/2009 02:55 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

=begin

=end

#3 - 03/31/2009 03:58 PM - brixen (Brian Shirai)

=begin
Hi,

On Mon, Mar 30, 2009 at 10:54 PM, Nobuyoshi Nakada nobu@ruby-lang.org wrote:

> Hi,
>
> At Tue, 31 Mar 2009 14:21:15 +0900,
> Brian Ford wrote in [ruby-core:23075]:
>
>> Did the displayed formats of Floats change in 1.9?
>
> Yes.

Where was this discussed and what was the reason for changing it?

>> There was a thread [ruby-core:22595] on the ML about the
>> format of Floats and marshaling, but there does not appear to
>> be any indication that the normal display format would
>> change. Also, I did not see any mention of a change of format
>> in the Changelog.
>
> It's not about marshaling.

Thanks for clarifying this part.

Brian

> --
> Nobu Nakada

=end

#4 - 03/31/2009 05:33 PM - zenspider (Ryan Davis)

=begin

On Mar 30, 2009, at 23:58 , brian ford wrote:

> Where was this discussed and what was the reason for changing it?

I can answer the latter:

> r22783 | nobu | 2009-03-05 01:36:39 -0800 (Thu, 05 Mar 2009) | 3 lines
>
> - numeric.c (flo_to_s): keeps enough precision for round trip. [ruby-core:22325]

=end

#5 - 04/01/2009 02:08 AM - brixen (Brian Shirai)

=begin
On Tue, Mar 31, 2009 at 1:32 AM, Ryan Davis ryand-ruby@zenspider.com wrote:

> On Mar 30, 2009, at 23:58 , brian ford wrote:
>
>> Where was this discussed and what was the reason for changing it?
>
> I can answer the latter:
>
>> r22783 | nobu | 2009-03-05 01:36:39 -0800 (Thu, 05 Mar 2009) | 3 lines
>>
>> - numeric.c (flo_to_s): keeps enough precision for round trip. [ruby-core:22325]

This is more "what it does" than why it was changed. Why was it needed?

Ruby cannot round-trip a lot of stuff via the object's to_s or inspect
string. That's what marshal is for.

Why was it so important to change Ruby's long standing friendly
floats? I personally see it having a lot of unintended consequences.

Brian

=end

#6 - 04/04/2009 03:49 PM - rogerdpack (Roger Pack)

=begin

- numeric.c (flo_to_s): keeps enough precision for round trip.

One possibility would be to allow Float#to_s to still be (depending on how you look at it) "friendly" or "imprecise."

And keep the precise version for Float#inspect.

The benefit of having them both verbose is that (tongue in cheek) it makes floats hideously ugly which might encourage people to avoid them :)

But having both available separately via #inspect and #to_s would be nice and I'd imagine a patch to that effect would be well received.

A discussion on it can be read at http://www.ruby-forum.com/topic/179361

Cheers.
-=r
=end

#7 - 04/05/2009 02:51 AM - brixen (Brian Shirai)

=begin
On Fri, Apr 3, 2009 at 11:49 PM, Roger Pack redmine@ruby-lang.org wrote:

> Issue #1336 has been updated by Roger Pack.
>
> - numeric.c (flo_to_s): keeps enough precision for round trip.
>
> One possibility would be to allow Float#to_s to still be (depending on how you look at it) "friendly" or "imprecise."
>
> And keep the precise version for Float#inspect.
>
> The benefit of having them both verbose is that (tongue in cheek) it makes floats hideously ugly which might encourage people to avoid them :)
>
> But having both available separately via #inspect and #to_s would be nice and I'd imagine a patch to that effect would be well received.
>
> A discussion on it can be read at http://www.ruby-forum.com/topic/179361

It's not an issue of float precision. It is an issue of representation
and there are many possibly representations of a float. The previous
representation was user-friendly. The change is not.

The only justification for the change that I see is this idea that
there is value to being able to round trip a float from #to_s through
eval. However, I think that is a poor reason to change because:

1. I can count the number Ruby classes that can be round-tripped this way on one hand.
2. There is a perfectly good mechanism for round-tripping any Ruby object.
3. If you don't want to marshal, you can use #sprintf when *you* want to round-trip a float via a string and eval.
4. The vast majority of times a float is represented as a string it is *not* to round-trip.

So, this decision takes a marginal case for which a perfectly good
mechanism already exists and promotes it to the common case. But
that's not all. The consequence for the common case is that 2.4 is
unnecessarily and uselessly echoed back to me as 2.3999999999999999.

It is very poor interface design to promote a marginal case above a

common case. There is nothing that this change in representation makes
better in the common case. It makes the common case hideous.

Floats are what they are. Use them as you will. Ruby used to have
nice, friendly representations of floats for humans. Nothing gained,
much lost. The decision should be reversed.

Brian

> Cheers.
>
> **-=r**
>
> http://redmine.ruby-lang.org/issues/show/1336
>
> ---
>
> http://redmine.ruby-lang.org

=end

#8 - 04/05/2009 04:57 PM - headius (Charles Nutter)

=begin
brian ford wrote:

> So, this decision takes a marginal case for which a perfectly good
> mechanism already exists and promotes it to the common case. But
> that's not all. The consequence for the common case is that 2.4 is
> unnecessarily and uselessly echoed back to me as 2.3999999999999999.
>
> It is very poor interface design to promote a marginal case above a
> common case. There is nothing that this change in representation makes
> better in the common case. It makes the common case hideous.
>
> Floats are what they are. Use them as you will. Ruby used to have
> nice, friendly representations of floats for humans. Nothing gained,
> much lost. The decision should be reversed.

Except that it was all a lie.

If a float can't be represented accurately, Ruby should not mask that,
because it further perpetuates the mistaken belief that floats are
accurate in Ruby. Treating 2.39999999999999 as 2.4 accomplishes exactly
one thing: it hides the true nature of floats.

I can appreciate the desire to have arbitrary-precision floating-point
math as the default in Ruby, but that's not the case right now. What we
have in Ruby 1.8 and 1.9 before this change is the horrible middle
ground of imprecise floats *pretending* to be precise. And we have run
into real-world bugs where JRuby's original lack of float-masking caused
application failures; people believed they could expect 2.4 in all cases
instead of 2.399999999999999. We should not have had to make our floats lie.

I would say either floats should always be arbitrary precision, or they
should be honest about their imprecision. Anything else is doing the
developer a disservice.

=end

#9 - 04/06/2009 03:24 AM - brent (Brent Roman)

=begin

I have to agree with Brian on this.

A quick, informal survey of various other languages shows that, by default,
most
will faithfully convert floating point 2.4 from a string to float and back
again.

'C', 'C++', Lua, Haskell, Matlab, BASIC, Pascal, etc.
all output "2.4" when their best floating point approximation of 2.4 is
converted to a string

with default options.

Python, however, does output 2.3999...
While someone can probably find another commonly used language that behaves
as Python does in this regard, I'm sure such languages are firmly in the
minority.

A good compromise solution, in Ruby's case, would be to redefine
Float#inspect such that it preserves the value's full precision without
rounding (for round-tripping, ultimate "truth", or whatever), while leaving
Float#to_s as it was.

This is consistent with other uses of the inspect method. Consider, for
example, that String and Symbol#to_s will return unquoted, ambiguous output
strings, while the corresponding #inspect methods return fully quoted,
unambiguous results.

It might also be a good idea to include a paragraph or two about the
behavior of limited precision floating point numbers in Ruby Doc for the
Float class. Include {puts 2.4} verses {p 2.4} as an example. After all,
the crux of this issue is education about the behavior of Floats that every
programmer using them really needs to understand, regardless of how they are
output.

Having said all this, I'm not going to get too worked up about it. I will
just sigh and add:

class Float
def to_s
"%g" % self
end
end

to the home grown Ruby version compatibility library that I already require
in most of my apps.

- brent

brixen wrote:

> On Fri, Apr 3, 2009 at 11:49 PM, Roger Pack redmine@ruby-lang.org wrote:
>
>> Issue #1336 has been updated by Roger Pack.
>>
>> - numeric.c (flo_to_s): keeps enough precision for round trip.
>>
>> One possibility would be to allow Float#to_s to still be (depending on
>> how you look at it) "friendly" or "imprecise."
>>
>> And keep the precise version for Float#inspect.
>>
>> The benefit of having them both verbose is that (tongue in cheek) it
>> makes floats hideously ugly which might encourage people to avoid them :)
>>
>> But having both available separately via #inspect and #to_s would be nice
>> and I'd imagine a patch to that effect would be well received.
>>
>> A discussion on it can be read at http://www.ruby-forum.com/topic/179361
>
> It's not an issue of float precision. It is an issue of representation
> and there are many possibly representations of a float. The previous
> representation was user-friendly. The change is not.
>
> The only justification for the change that I see is this idea that
> there is value to being able to round trip a float from #to_s through
> eval. However, I think that is a poor reason to change because:
>
> 1. I can count the number Ruby classes that can be round-tripped this way on one hand.
> 2. There is a perfectly good mechanism for round-tripping any Ruby object.
> 3. If you don't want to marshal, you can use #sprintf when *you* want to round-trip a float via a string and eval.
> 4. The vast majority of times a float is represented as a string it is *not* to round-trip.

> So, this decision takes a marginal case for which a perfectly good
> mechanism already exists and promotes it to the common case. But
> that's not all. The consequence for the common case is that 2.4 is
> unnecessarily and uselessly echoed back to me as 2.3999999999999999.
>
> It is very poor interface design to promote a marginal case above a
> common case. There is nothing that this change in representation makes
> better in the common case. It makes the common case hideous.
>
> Floats are what they are. Use them as you will. Ruby used to have
> nice, friendly representations of floats for humans. Nothing gained,
> much lost. The decision should be reversed.
>
> Brian
>
>> Cheers.
>>
>> **-=r**
>>
>> http://redmine.ruby-lang.org/issues/show/1336
>>
>> ---
>>
>> http://redmine.ruby-lang.org

--
View this message in context:
http://www.nabble.com/-ruby-core%3A23075---Bug--1336--Change-in-string-representation-of-Floats-tp22798535p22896937.html
Sent from the ruby-core mailing list archive at Nabble.com.

=end

#10 - 04/06/2009 12:51 PM - mame (Yusuke Endoh)

=begin
I agree with Brian.

I think it is a spec for Ruby 1.8 and 1.9 to fail to round-trip,
even if it may be inconvenient a little. I cannot find a reason
why `round-trip' feature is more important than compatibility.

In addition, the change actually attacked my some scripts... :-(

`inspect' used to return user-friendly (and sometimes imprecise)
representation.
Why not create a new method Float#dump for round-trip and precise
representation (like String#dump) ?

If Float#inspect must be changed, please go over the shortest
representation algorithm which was mentioned in [ruby-core:22629].

Just my 2 cents,

2009/4/5 brian ford brixen@gmail.com:

> On Fri, Apr 3, 2009 at 11:49 PM, Roger Pack redmine@ruby-lang.org wrote:
>
>> Issue #1336 has been updated by Roger Pack.
>>
>>> - numeric.c (flo_to_s): keeps enough precision for round trip.
>>
>> One possibility would be to allow Float#to_s to still be (depending on how you look at it) "friendly" or "imprecise."
>>
>> And keep the precise version for Float#inspect.
>>
>> The benefit of having them both verbose is that (tongue in cheek) it makes floats hideously ugly which might encourage people to avoid them :)
>>
>> But having both available separately via #inspect and #to_s would be nice and I'd imagine a patch to that effect would be well received.
>>
>> A discussion on it can be read at http://www.ruby-forum.com/topic/179361

It's not an issue of float precision. It is an issue of representation
and there are many possibly representations of a float. The previous
representation was user-friendly. The change is not.

The only justification for the change that I see is this idea that
there is value to being able to round trip a float from #to_s through
eval. However, I think that is a poor reason to change because:

1. I can count the number Ruby classes that can be round-tripped this way on one hand.
2. There is a perfectly good mechanism for round-tripping any Ruby object.
3. If you don't want to marshal, you can use #sprintf when *you* want to round-trip a float via a string and eval.
4. The vast majority of times a float is represented as a string it is *not* to round-trip.

So, this decision takes a marginal case for which a perfectly good
mechanism already exists and promotes it to the common case. But
that's not all. The consequence for the common case is that 2.4 is
unnecessarily and uselessly echoed back to me as 2.3999999999999999.

It is very poor interface design to promote a marginal case above a
common case. There is nothing that this change in representation makes
better in the common case. It makes the common case hideous.

Floats are what they are. Use them as you will. Ruby used to have
nice, friendly representations of floats for humans. Nothing gained,
much lost. The decision should be reversed.

Brian

Cheers.

-=r

http://redmine.ruby-lang.org/issues/show/1336

---

http://redmine.ruby-lang.org

--
Yusuke ENDOH mame@tsg.ne.jp

=end

#11 - 04/11/2009 03:17 PM - rogerdpack (Roger Pack)

=begin
I like the recent change. Now it acts like this:

```
0.9
=> 0.9
2.1 - 3.0
=> -0.8999999999999999
(2.1-3.0) == -0.9
=> false
```

Is this satisfactory, or would anyone still prefer Float#to_s => rounded, Float#inspect => non rounded (i.e. 0.9.inspect => "0.8999999999")? Just wondering since I brought up this mess :)
I kind of like the current way. It hides some complexity while still explaining to end users why equality fails.
Thanks for the update.

-=r
=end

#12 - 04/11/2009 04:03 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Sat, 11 Apr 2009 15:17:24 +0900,
Roger Pack wrote in [ruby-core:23179]:

> Is this satisfactory, or would anyone still prefer Float#to_s
> => rounded, Float#inspect => non rounded (i.e. 0.9.inspect =>
> "0.8999999999")? Just wondering since I brought up this mess
> :)

I had another thought, Float#to_s could take an optional
argument to specify the precision or the format.

e.g.:

```
0.9.to_s(3)   #=> "0.900"
0.9.to_s("e") #=> "9.000000e-01"
0.9.to_s(prec: 4, width: 6) #=> " 9.000"
```

--
Nobu Nakada

=end

#13 - 04/12/2009 03:53 AM - brent (Brent Roman)

=begin

I'd prefer that Float#to_s => rounded, Float#inspect => non rounded
(i.e. 0.9.inspect => "0.8999999999")

- brent

Sakuro OZAWA wrote:

> Issue #1336 has been updated by Roger Pack.
>
> I like the recent change. Now it acts like this:
>
> ```
> 0.9
> => 0.9
> 2.1 - 3.0
> => -0.8999999999999999
> (2.1-3.0) == -0.9
> => false
> ```
>
> Is this satisfactory, or would anyone still prefer Float#to_s => rounded,
> Float#inspect => non rounded (i.e. 0.9.inspect => "0.8999999999")? Just
> wondering since I brought up this mess :)
> I kind of like the current way. It hides some complexity while still
> explaining to end users why equality fails.
> Thanks for the update.
>
> ## -=r
>
> http://redmine.ruby-lang.org/issues/show/1336
>
> ---
>
> http://redmine.ruby-lang.org

--
View this message in context:
http://www.nabble.com/-ruby-core%3A23075---Bug--1336--Change-in-string-representation-of-Floats-tp22798535p23004293.html
Sent from the ruby-core mailing list archive at Nabble.com.

=end

#14 - 04/12/2009 03:59 AM - brent (Brent Roman)

=begin

Your latter example seems quite nice.
Especially if it would accept unabbreviated forms such as this:

```
0.9.to_s(precision: 4, width: 6) #=> " 9.000"
```

- brent

Nobuyoshi Nakada-2 wrote:

> Hi,
>
> At Sat, 11 Apr 2009 15:17:24 +0900,
> Roger Pack wrote in [ruby-core:23179]:
>
>> Is this satisfactory, or would anyone still prefer Float#to_s
>> => rounded, Float#inspect => non rounded (i.e. 0.9.inspect =>
>> "0.8999999999")? Just wondering since I brought up this mess
>> :)
>
> I had another thought, Float#to_s could take an optional
> argument to specify the precision or the format.
>
> e.g.:
>
> 0.9.to_s(3)   #=> "0.900"
> 0.9.to_s("e") #=> "9.000000e-01"
> 0.9.to_s(prec: 4, width: 6) #=> " 9.000"
>
> --
> Nobu Nakada

--
View this message in context:
http://www.nabble.com/-ruby-core%3A23075---Bug--1336--Change-in-string-representation-of-Floats-tp22798535p23004340.html
Sent from the ruby-core mailing list archive at Nabble.com.

=end

#15 - 04/13/2009 07:28 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:23181] Re: [Bug #1336] Change in string representation of Floats"
on Sun, 12 Apr 2009 03:53:12 +0900, Brent Roman brent@mbari.org writes:

|I'd prefer that Float#to_s => rounded, Float#inspect => non rounded
|(i.e. 0.9.inspect => "0.8999999999")

I see several people who prefer that way. Could you elaborate the
reason behind the preference? Considering being to_s for mere string
representation, and inspect being human readable string
representation, I thought other way around.

```
                            matz.
```

=end

#16 - 04/13/2009 10:55 AM - brent (Brent Roman)

=begin

Perhaps I'm not understanding the question.
Consider:

```
s = "string"
s.inspect   ==>  "string"   #this will round-trip
s.to_s      ==>   string    #this lacks the required delimiters

s = :symbol
s.inspect  ==>  :symbol   #similarly...
s.to_s     ==>  symbol    #but this lacks delimiters and is abiguous

a = [1,2,3]                #again...
a.inspect  ==>  [1, 2, 3]
a.to_s     ==>  123        #this is pretty useless

h = [:foo=>:bar, :bar=>:foo]   #and, finally...
```

a.inspect ==> {:foo=>:bar, :bar=>:foo}
a.to_s   ==> foobarbarfoo   #as is this.

In all the above examples, the #inspect method outputs delimiters necessary for Ruby's parser to recreate the data object, while the #to_s method omits delimiters to produce terser, albeit sometimes ambiguous, output.

Admittedly, to_s doesn't work very well on collections, but, it will convert any basic type to a short, human readable string free of extraneous punctuation. Float#to_s should do the same for consistency' sake.

As a practical matter, changing Float#to_s at this point will cause many no longer maintained Ruby scripts to output confusing long trains of digits after the decimal. And, consider the bugs that will emerge when data base queries on ranges of values like 0.1 to 0.9, if the string representations actually passed (from Ruby via Float.to_s) into the data base engine (which for all we know is using decimal floats) are in fact 0.10000000000000001 to 0.90000000000000002

- brent

Yukihiro Matsumoto wrote:

> Hi,
>
> In message "Re: [ruby-core:23181] Re: [Bug #1336] Change in string representation of Floats"
> on Sun, 12 Apr 2009 03:53:12 +0900, Brent Roman brent@mbari.org
> writes:
>
> |I'd prefer that Float#to_s => rounded, Float#inspect => non rounded
> |(i.e. 0.9.inspect => "0.8999999999")
>
> I see several people who prefer that way. Could you elaborate the reason behind the preference? Considering being to_s for mere string representation, and inspect being human readable string representation, I thought other way around.
>
> ```
>                      matz.
> ```

--
View this message in context:
http://www.nabble.com/-ruby-core%3A23075---Bug--1336--Change-in-string-representation-of-Floats-tp22798535p23016776.html
Sent from the ruby-core mailing list archive at Nabble.com.

=end

#17 - 04/13/2009 11:17 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:23187] Re: [Bug #1336] Change in string representation of Floats"
on Mon, 13 Apr 2009 10:54:46 +0900, Brent Roman brent@mbari.org writes:

|Perhaps I'm not understanding the question.
|Consider:
|
|s = "string"
|s.inspect   ==> "string"   #this will round-trip
|s.to_s   ==>   string   #this lacks the required delimiters
|
|s = :symbol
|s.inspect ==> :symbol   #similarly...
|s.to_s   ==> symbol   #but this lacks delimiters and is abiguous
|
|a = [1,2,3]   #again...
|a.inspect ==> [1, 2, 3]
|a.to_s   ==> 123   #this is pretty useless
|
|h = [:foo=>:bar, :bar=>:foo]   #and, finally...
|a.inspect ==> {:foo=>:bar, :bar=>:foo}
|a.to_s   ==> foobarbarfoo   #as is this.
|

|In all the above examples, the #inspect method outputs delimiters necessary
|for Ruby's parser to recreate the data object, while the #to_s method omits
|delimiters to produce terser, albeit sometimes ambiguous, output.

to_s for arrays and hashes are changed in 1.9.

Perhaps we need 3 ways for string representation:

- mere string representation
- human readable representation
- regenerating representation (as in [ruby-core:23128])

|As a practical matter, changing Float#to_s at this point will cause many no
|longer maintained Ruby scripts to output confusing long trains of digits
|after the decimal. And, consider the bugs that will emerge when data base
|queries on ranges of values like 0.1 to 0.9, if the string representations
|actually passed (from Ruby via Float.to_s) into the data base engine (which
|for all we know is using decimal floats) are in fact 0.10000000000000001 to
|0.90000000000000002

I'm not sure what you mean by "at this point". The point we move 1.8
from 1.9 is the only point where we can make such changes. In
addition, the latest trunk gives you "0.1" for 1.0.inspect.

```
                              matz.
```

=end

#18 - 04/13/2009 01:12 PM - brent (Brent Roman)

=begin

I'm not sure I understand the difference between "mere string" and "human
readable" representations.
Could you give an example?

1.8 is similar enough to 1.9 that many straightforward scripts will run
without change.
However, Float#to_s is likely used in almost every script that uses Floats,
even those that would otherwise run unchanged under 1.9. I think changing
it introduces bugs in these for no great gain. Others on this thread have
already said the same.

I'm sure you meant that 0.1.inspect would produce "0.1" for the current
trunk.
I could see how careful string conversion could ensure that simple constant
literals would always
be preserved. The trouble really begins only after one starts doing
arithmetic with them.

What does the current trunk output for (2.1-3.0).to_s ?

Consider that (here at least):
2.1-3.0+0.9 == 1.110223024625156540e-16

I'd like to be proven wrong, but I currently believe that the one cannot
create a human friendly Float string conversion that is also a regenerating
representation.

- brent

Yukihiro Matsumoto wrote:

> to_s for arrays and hashes are changed in 1.9.
>
> Perhaps we need 3 ways for string representation:
>
> - mere string representation
> - human readable representation
> - regenerating representation (as in [ruby-core:23128])
>
> |As a practical matter, changing Float#to_s at this point will cause many
> no
> |longer maintained Ruby scripts to output confusing long trains of digits
> |after the decimal. And, consider the bugs that will emerge when data

> base
> |queries on ranges of values like 0.1 to 0.9, if the string
> representations
> |actually passed (from Ruby via Float.to_s) into the data base engine
> (which
> |for all we know is using decimal floats) are in fact 0.10000000000000001
> to
> |0.90000000000000002
>
> I'm not sure what you mean by "at this point". The point we move 1.8
> from 1.9 is the only point where we can make such changes. In
> addition, the latest trunk gives you "0.1" for 1.0.inspect.
>
> ```
>                         matz.
> ```

--
View this message in context:
http://www.nabble.com/-ruby-core%3A23075---Bug--1336--Change-in-string-representation-of-Floats-tp22798535p23017694.html
Sent from the ruby-core mailing list archive at Nabble.com.

=end

#19 - 04/13/2009 04:19 PM - rue (Eero Saynatkari)

=begin
Excerpts from Yukihiro Matsumoto's message of Mon Apr 13 05:17:28 +0300 2009:

> Perhaps we need 3 ways for string representation:
>
> - mere string representation
> - human readable representation
> - regenerating representation (as in [ruby-core:23128])

I think I am in agreement, but I would like to amend the
description of the three to confirm:

#to_s: String representing the object mainly for output
#inspect: Internal details, ivars etc., mainly for debug
#dump: Round-trip/evalable representation

The latter's usefulness is slightly questionable since, as
was pointed out, one may round-trip through e.g. Marshal,
but it could be assumed to be a 'lighter-weight' version
(surely a less intimidating one) of the same.

Since String#dump already exists, perhaps a different name
is in order for the latter. Call it #to_eval or whatever,
but I believe these three offer the necessary granularity
of semantics.

Is this an agreeable proposition?

--
Magic is insufficiently advanced technology.

=end

#20 - 04/14/2009 01:45 PM - rogerdpack (Roger Pack)

=begin
currently (for curiosity sake) trunk does:

```
    [1,2,3].to_s
    => "[1, 2, 3]"
    [1,2,3].inspect
    => "[1, 2, 3]"

    -0.9
    => -0.9
    2.1-3.0
    => -0.8999999999999999
    2.1-3.0+0.9
    => 1.1102230246251565e-16
```

> 1.9-0.9
> => 0.9999999999999999

I'd personally be all right with how it is now or with having .to_s round and .inspect either (not round or behave like .to_s does currently). As long as somehow or other users are alerted to that fact that "1.9-0.9 ain't one" (a fact slightly obscured previously).
Thanks.
-=r
=end

#21 - 04/14/2009 02:05 PM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:23189] Re: [Bug #1336] Change in string representation of Floats"
on Mon, 13 Apr 2009 13:12:13 +0900, Brent Roman brent@mbari.org writes:
|
|What does the current trunk output for (2.1-3.0).to_s ?

"-0.8999999999999999"

|Consider that (here at least):
|2.1-3.0+0.9 == 1.110223024625156540e-16
|
|I'd like to be proven wrong, but I currently believe that the one cannot
|create a human friendly Float string conversion that is also a regenerating
|representation.

I agree with it. My opinion is that IF we need to get regenerating
string representation, we need a method dedicated to the purpose
(e.g. dump). I am not yet convinced we need such method.

And the current situation is #to_s and #inspect for floats share the
implementation, so both methods give same representation.

The reason for the change (increased precision) was too many people do
not understand the nature of floating point values on computers, and
fall in error traps by illusion partly emphasized by dropping
precision in string representations.

```
                                        matz.
```

=end

#22 - 04/14/2009 02:22 PM - headius (Charles Nutter)

=begin
Yukihiro Matsumoto wrote:

> The reason for the change (increased precision) was too many people do
> not understand the nature of floating point values on computers, and
> fall in error traps by illusion partly emphasized by dropping
> precision in string representations.

Well put. This is what I tried to get across in my too-long response.
Don't hide the imprecision behind false precision, because it only leads
to bugs.

- Charlie

=end

#23 - 04/14/2009 03:06 PM - shyouhei (Shyouhei Urabe)

=begin
Gary Wright wrote:

> I don't think it is even possible unless you introduce binary (or maybe
> hex) floating point literals. That is the only way you are going to be
> able to safely round-trip between an internal IEEE floating point value
> and a string representation.

No that's not the only one. Take a look at [ruby-core:23145]. You can reach
to the same answer when you happen to think that hexadecimal numbers are all
representable in decimal. So are binaries. Not all decimal floats can be
represented in binary, but all binary floats must have a corresponding decimal
representation. That is sufficient for this situation.

=end

#24 - 04/14/2009 05:22 PM - zenspider (Ryan Davis)

=begin

On Apr 13, 2009, at 00:19 , Eero Saynatkari wrote:

> I think I am in agreement, but I would like to amend the
> description of the three to confirm:
>
> #to_s: String representing the object mainly for output
> #inspect: Internal details, ivars etc., mainly for debug
> #dump: Round-trip/evalable representation

I agree with this and thought/hoped that was the original intent of

inspect.

In smalltalk (a stream-based IO system) they use #printOn: like our

puts/to_s and #storeOn: for something like a meld of your #dump and

#inspect. storeOn: outputs the code that would eval to an object equal/
equivalent to the original (see below). Unfortunately, our #inspect

for non-core objects don't default to a ruby expression.

I think #to_ruby would be a better name than #dump. It doesn't clash

with Marshal/YAML and matches #to_s and friends.

```
storeOn: aStream
"Append to the argument aStream a sequence of characters that is an
expression whose evaluation creates an object similar to the

receiver."

aStream nextPut: $(.
self class isVariable
ifTrue: [aStream nextPutAll: '(', self class name, ' basicNew: ';
store: self basicSize;
nextPutAll: ') ']
ifFalse: [aStream nextPutAll: self class name, ' basicNew'].
1 to: self class instSize do:
[:i |
aStream nextPutAll: ' instVarAt: ';
store: i;
nextPutAll: ' put: ';
store: (self instVarAt: i);
nextPut: $;].
1 to: self basicSize do:
[:i |
aStream nextPutAll: ' basicAt: ';
store: i;
nextPutAll: ' put: ';
store: (self basicAt: i);
nextPut: $;].
aStream nextPutAll: ' yourself)'
```

=end

#25 - 04/17/2009 10:02 PM - rogerdpack (Roger Pack)

=begin

In addition, the change actually attacked my some scripts... :-(

Does it still break them? Can anyone give an example where the "more honest" #to_s broke their scripts (for my curiosity sake).
It seems the general thought currently is that to_s should be "dishonest" and inspect should be "honest"? This would work well, as at least in irb users would know more precisely what was going on, while still preserving pretty output and backwards compat.
Thanks.
-=r
=end