Ruby master - Bug #17017

Range#max & Range#minmax incorrectly use Float end as max

07/07/2020 08:16 PM - sambostock (Sam Bostock)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.8.0dev (2020-07-14T04:19:55Z master e60cd14d85) [x86_64-darwin17]	**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

While continuing to add edge cases to Range#minmax specs, I discovered the following bug:

```
(1..3.1).to_a         == [1, 2, 3] # As expected

(1..3.1).to_a.max     == 3         # As expected
(1..3.1).to_a.minmax  == [1, 3]    # As expected

(1..3.1).max     == 3.1            # Should be 3, as above
(1..3.1).minmax  == [1, 3.1]       # Should be [1, 3], as above
```

One way to detect this scenario might be to do (whatever the C equivalent is of)

```
range_end.is_a?(Numeric)                        // Is this a numeric range?
  && (range_end - range_begin).modulo(1) == 0 // Can we reach the range_end using the standard step size (1)
```

As for how to handle it, a couple options come to mind:

- We could error out and do something similar to what we do for exclusive ranges

```
raise TypeError, 'cannot exclude non Integer end value'
```

- We might be able to calculate the range end by doing something like

```
num_steps = (range_end / range_beg).to_i - 1 # one fewer steps than would exceed the range_end
max = range_beg + num_steps                  # take that many steps all at once
```

- We could delegate to super and enumerate the range to find the max

```
super
```

- We could update the documentation to define the max for this case as the range_end, similarly to how the documentation for include? says it behaves like cover? for numeric ranges.

Associated revisions

Revision 8900a255 - 07/13/2020 05:09 PM - jeremyevans (Jeremy Evans)

Fix Range#{max,minmax} for range with integer beginning and non-integer end

Previously, for inclusive ranges, the max would show up as the
end of the range, even though the end was not an integer and would
not be the maximum value. For exclusive ranges, max/minmax would
previously raise a TypeError, even though it is possible to get the
correct maximum.

This change to max/minmax also uncovered a similar error in cover?,
which calls max in certain cases, so adjust the code there so that
cover? still works as expected.

Fixes [Bug #17017]

Revision 05bf811c - 07/19/2020 02:25 PM - jeremyevans (Jeremy Evans)

Special case Range#max for integer beginning and Float::Infinity end

Popular Ruby libraries such as Rails and Rubocop relying on the
previous behavior, even though it is technically a bug. The
correct behavior is probably raising RangeError, since that is what
an endless range raises.

Related to [Bug #17017]

History

#1 - 07/07/2020 08:41 PM - sambostock (Sam Bostock)

- Description updated

Add .to_a examples to clarify unexpected behaviour.

#2 - 07/07/2020 08:43 PM - sambostock (Sam Bostock)

- Description updated

Add full array example

#3 - 07/07/2020 08:56 PM - sambostock (Sam Bostock)

- Description updated

Add documentation change as an option

#4 - 07/10/2020 04:29 PM - jeremyevans0 (Jeremy Evans)

I've added a pull request that fixes this issue: https://github.com/ruby/ruby/pull/3306

#5 - 07/13/2020 05:09 PM - jeremyevans (Jeremy Evans)

- Status changed from Open to Closed

Applied in changeset git|8900a25581822759daca528d46a75e0b743fc22e.

Fix Range#{max,minmax} for range with integer beginning and non-integer end

Previously, for inclusive ranges, the max would show up as the
end of the range, even though the end was not an integer and would
not be the maximum value. For exclusive ranges, max/minmax would
previously raise a TypeError, even though it is possible to get the
correct maximum.

This change to max/minmax also uncovered a similar error in cover?,
which calls max in certain cases, so adjust the code there so that
cover? still works as expected.

Fixes [Bug #17017]

#6 - 07/15/2020 12:47 AM - koic (Koichi ITO)

- ruby -v changed from 2.7.1 to ruby 2.8.0dev (2020-07-14T04:19:55Z master e60cd14d85) [x86_64-darwin17]

I encountered a breaking change in RuboCop repository when using ruby 2.8.0dev.
https://github.com/rubocop-hq/rubocop/blob/v0.88.0/lib/rubocop/comment_config.rb#L110

So, I have a question. Is this an expected behaviour?

Ruby 2.7.1:

```
% ruby -ve 'p (42..Float::INFINITY).max'
ruby 2.7.1p83 (2020-03-31 revision a0c7c23c9c) [x86_64-darwin17]
Infinity
```

Ruby 2.8.0-dev:

```
% ruby -ve 'p (42..Float::INFINITY).max'
ruby 2.8.0dev (2020-07-14T04:19:55Z master e60cd14d85) [x86_64-darwin17]
```

```
-e:1:in `floor': Infinity (FloatDomainError)
        from -e:1:in `max'
        from -e:1:in `<main>'
```

So, it expects that FloatDomainError will not occur even if the end of range is Float::INFINITY.

Thank you.

#7 - 07/15/2020 05:24 AM - jeremyevans0 (Jeremy Evans)

koic (Koichi ITO) wrote in #note-6:

> I encountered a breaking change in RuboCop repository when using ruby 2.8.0dev.
> https://github.com/rubocop-hq/rubocop/blob/v0.88.0/lib/rubocop/comment_config.rb#L110
>
> So, I have a question. Is this an expected behaviour?

Yes, it is expected behavior, at least to me. I believe the Ruby 2.7 behavior is wrong, because a range that starts with an integer will never have a non-integer maximum value, since the increment is an integer. Range#max specifies the maximum value in the range (should be the same as range.to_a.max), not the end of the range (that is Range#end). However, my expectation may be different from matz's expectation.

#8 - 07/15/2020 07:19 AM - koic (Koichi ITO)

Thank you for your reply. For now I've learned that updating to (42.to_f..Float::INFINITY) will prevent the error.

#9 - 07/15/2020 03:37 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Closed to Open

jeremyevans0 (Jeremy Evans) wrote in #note-7:

> koic (Koichi ITO) wrote in #note-6:
>
> > I encountered a breaking change in RuboCop repository when using ruby 2.8.0dev.
> > https://github.com/rubocop-hq/rubocop/blob/v0.88.0/lib/rubocop/comment_config.rb#L110
> >
> > So, I have a question. Is this an expected behaviour?
>
> Yes, it is expected behavior, at least to me. I believe the Ruby 2.7 behavior is wrong, because a range that starts with an integer will never have a non-integer maximum value, since the increment is an integer.

I should have been more clear when I commented to take care of Infinity, but I believe that previous behavior should remain. Since there is no Integer::INFINITY, returning Float::INFINITY is quite descriptive and useful. Raising a RangeError doesn't seem helpful. (I wrote RangeError since raising FloatDomainError as currently is definitely a mistake).

#10 - 07/15/2020 03:58 PM - jeremyevans0 (Jeremy Evans)

marcandre (Marc-Andre Lafortune) wrote in #note-9:

> jeremyevans0 (Jeremy Evans) wrote in #note-7:
>
> > koic (Koichi ITO) wrote in #note-6:
> >
> > > I encountered a breaking change in RuboCop repository when using ruby 2.8.0dev.
> > > https://github.com/rubocop-hq/rubocop/blob/v0.88.0/lib/rubocop/comment_config.rb#L110
> > >
> > > So, I have a question. Is this an expected behaviour?
> >
> > Yes, it is expected behavior, at least to me. I believe the Ruby 2.7 behavior is wrong, because a range that starts with an integer will never have a non-integer maximum value, since the increment is an integer.
>
> I should have been more clear when I commented to take care of Infinity, but I believe that previous behavior should remain. Since there is no Integer::INFINITY, returning Float::INFINITY is quite descriptive and useful. Raising a RangeError doesn't seem helpful. (I wrote RangeError since raising FloatDomainError as currently is definitely a mistake).

With current Ruby, you should use an endless range instead of range with an infinite end.

The previous behavior of returning Float::INFINITY seems wrong to me. I suppose the current behavior of raising FloatDomainError may be unexpected, and RangeError could be returned instead. Note that if you use the exclusive range (42...Float::INFINITY), you would get a TypeError in

2.7.1, so it's not like we exclusively used RangeError for issues like these. Mathematically, 42..Float::INFINITY and 42...Float::INFINITY represent the same range, so you should get the same behavior for both.

All that said, I'm not completely opposed to special casing Float::INFINITY in this case. I would like to hear opinions from other committers.

#11 - 07/15/2020 04:40 PM - marcandre (Marc-Andre Lafortune)

jeremyevans0 (Jeremy Evans) wrote in #note-10:

> With current Ruby, you should use an endless range instead of range with an infinite end.

Endless ranges are 2.6+. Ruby 2.5 is not yet EOL. RuboCop supports 2.4 and other gems support earlier Rubies.

More importantly, endless range max is currently not useful... Note even (4.2..).max and variations return an infinity. I should open an issue about this...

> Note that if you use the exclusive range (42...Float::INFINITY), you would get a TypeError in 2.7.1, so it's not like we exclusively used RangeError for issues like these.

I agree with you, TypeError is also wrong.

> Mathematically, 42..Float::INFINITY and 42...Float::INFINITY represent the same range

We'll have to disagree with this □□. I'm sure in some mathematical definition that it is the case, but in many others it is not. 42..Float::INFINITY is not defined in traditional Euclidian space. On a Riemann sphere it is not the same as 42...Float::INFINITY. If seen as sequences, you could see them as ω ("infinity") and ω - 1 which are distinct numbers (see surreal numbers), etc.

> All that said, I'm not completely opposed to special casing Float::INFINITY in this case.

Great. I hope we can make infinity handling useful.

#12 - 07/16/2020 09:31 AM - yahonda (Yasuo Honda)

Let me provide another incompatibility/breaking change since this commit.

One of the Rails validation, validates_length_of(:title, within: 5..Float::INFINITY) raises Infinity (FloatDomainError) with Ruby 2.8.0dev.
This behavior of validates_length_of implemented since Rails 4.0.0 by https://github.com/rails/rails/pull/4905

I think there is no "Integer::INFINITY" in Ruby, "..Float::INFINITY" may be used by other applications/frameworks.
It would be appreciated that this Rails behavior kept with Ruby 2.8.0-dev.

- Script to reproduce

```
# frozen_string_literal: true

require "bundler/inline"

gemfile(true) do
  source "https://rubygems.org"

  git_source(:github) { |repo| "https://github.com/#{repo}.git" }

  gem "rails", github: "rails/rails"
  gem "sqlite3"
end

require "active_record"
require "minitest/autorun"
require "logger"

# This connection will do for database-independent bug reports.
ActiveRecord::Base.establish_connection(adapter: "sqlite3", database: ":memory:")
ActiveRecord::Base.logger = Logger.new(STDOUT)

ActiveRecord::Schema.define do
  create_table :posts, force: true do |t|
    t.string :title
  end
end

class Post < ActiveRecord::Base
```

```
  validates_length_of(:title, within: 5..Float::INFINITY)
end

class ValidationTest < Minitest::Test
  def test_validates_length_of
    post = Post.create!(title: "first post")
    assert post.title, "first post"
  end
end
```

- Result with ruby 2.8.0dev raises "Infinity (FloatDomainError)"

```
% ruby -v ; ruby diag.rb
ruby 2.8.0dev (2020-07-16T02:49:09Z master 1fb4e28002) [x86_64-darwin20]
Fetching https://github.com/rails/rails.git
Fetching gem metadata from https://rubygems.org/......
Fetching gem metadata from https://rubygems.org/.............
Fetching gem metadata from https://rubygems.org/.............
Resolving dependencies...
Using rake 13.0.1
Using bundler 2.2.0.dev
Using method_source 1.0.0
Using minitest 5.14.1
Using zeitwerk 2.4.0
Using builder 3.2.4
Using erubi 1.9.0
Using mini_portile2 2.4.0
Using crass 1.0.6
Using rack 2.2.3
Using nio4r 2.5.2
Using websocket-extensions 0.1.5
Using mimemagic 0.3.5
Using mini_mime 1.0.2
Using marcel 0.3.3
Using sqlite3 1.4.2
Using concurrent-ruby 1.1.6
Using nokogiri 1.10.10
Using rack-test 1.1.0
Using websocket-driver 0.7.3
Using thor 1.0.1
Using mail 2.7.1
Using i18n 1.8.3
Using tzinfo 2.0.2
Using sprockets 4.0.2
Using loofah 2.6.0
Using activesupport 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using rails-html-sanitizer 1.3.0
Using rails-dom-testing 2.0.3
Using globalid 0.4.2
Using activemodel 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using actionview 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using activejob 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using actionpack 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using activerecord 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using actioncable 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using activestorage 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using railties 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using sprockets-rails 3.2.1
Using actionmailer 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using actionmailbox 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using actiontext 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
Using rails 6.1.0.alpha from https://github.com/rails/rails.git (at master@bcfa51f)
-- create_table(:posts, {:force=>true})
D, [2020-07-16T18:20:56.790796 #66220] DEBUG -- :    (0.7ms)  SELECT sqlite_version(*)
D, [2020-07-16T18:20:56.791115 #66220] DEBUG -- :    (0.0ms)  DROP TABLE IF EXISTS "posts"
D, [2020-07-16T18:20:56.791443 #66220] DEBUG -- :    (0.2ms)  CREATE TABLE "posts" ("id" integer PRIMARY KEY A
UTOINCREMENT NOT NULL, "title" varchar)
   -> 0.0038s
D, [2020-07-16T18:20:56.814223 #66220] DEBUG -- :    (0.1ms)  CREATE TABLE "ar_internal_metadata" ("key" varch
ar NOT NULL PRIMARY KEY, "value" varchar, "created_at" datetime(6) NOT NULL, "updated_at" datetime(6) NOT NULL
)
D, [2020-07-16T18:20:56.824002 #66220] DEBUG -- :   ActiveRecord::InternalMetadata Load (0.1ms)  SELECT "ar_in
ternal_metadata".* FROM "ar_internal_metadata" WHERE "ar_internal_metadata"."key" = ? LIMIT ?  [["key", "envir
onment"], ["LIMIT", 1]]
D, [2020-07-16T18:20:56.827703 #66220] DEBUG -- :   TRANSACTION (0.0ms)  begin transaction
```

```
D, [2020-07-16T18:20:56.827989 #66220] DEBUG -- :   ActiveRecord::InternalMetadata Create (0.1ms)  INSERT INTO
 "ar_internal_metadata" ("key", "value", "created_at", "updated_at") VALUES (?, ?, ?, ?)  [["key", "environmen
t"], ["value", "development"], ["created_at", "2020-07-16 09:20:56.827304"], ["updated_at", "2020-07-16 09:20:
56.827304"]]
D, [2020-07-16T18:20:56.828184 #66220] DEBUG -- :   TRANSACTION (0.0ms)  commit transaction
/Users/yahonda/.rbenv/versions/2.8.0-dev/lib/ruby/gems/2.8.0/bundler/gems/rails-bcfa51fba6ee/activemodel/lib/a
ctive_model/validations/length.rb:14:in `floor': Infinity (FloatDomainError)
    from /Users/yahonda/.rbenv/versions/2.8.0-dev/lib/ruby/gems/2.8.0/bundler/gems/rails-bcfa51fba6ee/activemo
del/lib/active_model/validations/length.rb:14:in `max'
    from /Users/yahonda/.rbenv/versions/2.8.0-dev/lib/ruby/gems/2.8.0/bundler/gems/rails-bcfa51fba6ee/activemo
del/lib/active_model/validations/length.rb:14:in `initialize'
    from /Users/yahonda/.rbenv/versions/2.8.0-dev/lib/ruby/gems/2.8.0/bundler/gems/rails-bcfa51fba6ee/activemo
del/lib/active_model/validations/with.rb:86:in `new'
    from /Users/yahonda/.rbenv/versions/2.8.0-dev/lib/ruby/gems/2.8.0/bundler/gems/rails-bcfa51fba6ee/activemo
del/lib/active_model/validations/with.rb:86:in `block in validates_with'
    from /Users/yahonda/.rbenv/versions/2.8.0-dev/lib/ruby/gems/2.8.0/bundler/gems/rails-bcfa51fba6ee/activemo
del/lib/active_model/validations/with.rb:85:in `each'
    from /Users/yahonda/.rbenv/versions/2.8.0-dev/lib/ruby/gems/2.8.0/bundler/gems/rails-bcfa51fba6ee/activemo
del/lib/active_model/validations/with.rb:85:in `validates_with'
    from /Users/yahonda/.rbenv/versions/2.8.0-dev/lib/ruby/gems/2.8.0/bundler/gems/rails-bcfa51fba6ee/activere
cord/lib/active_record/validations/length.rb:20:in `validates_length_of'
    from diag.rb:29:in `<class:Post>'
    from diag.rb:28:in `<main>'
%
```

#13 - 07/16/2020 05:17 PM - jeremyevans0 (Jeremy Evans)

I've added a pull request to restore compatibility for Range#max with integer beginning and Float::Infinity end: https://github.com/ruby/ruby/pull/3326

I think this pull request should require matz approval because it is a deliberate tradeoff of correctness in order to keep compatibility. The correct behavior should be raising RangeError, as that is what an endless range returns. If you would like matz to consider it, please bring it up for discussion at a future Ruby developers meeting.

#14 - 07/19/2020 01:17 AM - yahonda (Yasuo Honda)

I have confirmed https://github.com/ruby/ruby/pull/3326 addresses the Active Model failure reported at https://bugs.ruby-lang.org/issues/17017#note-12

#15 - 07/19/2020 06:41 AM - marcandre (Marc-Andre Lafortune)

jeremyevans0 (Jeremy Evans) wrote in #note-13:

> I've added a pull request to restore compatibility for Range#max with integer beginning and Float::Infinity end: https://github.com/ruby/ruby/pull/3326

Good, thanks.

> I think this pull request should require matz approval because it is a deliberate tradeoff of correctness in order to keep compatibility. The correct behavior should be raising RangeError, as that is what an endless range returns.

Well, (1.0..).max raises a RangeError while (1.0..Float::INFINITY).max returns Float::INFINITY; which is "correct"?

> If you would like matz to consider it, please bring it up for discussion at a future Ruby developers meeting.

I'm surprised by the lack of response from other Ruby committers.

Unless I'm mistaken, this behavior change was not approved by Matz (or anybody else), changes a behavior that dates back to Ruby 1.8, breaks (that we know of) activemodel and rubocop, doesn't even raise the right error and isn't tested, and is a change that I and others disapprove of. Yet when you propose that that commit remains as is and that someone else has to bring up the situation with Matz, no other committer seems confused by this.

I would love to have pointers on what I'm doing wrong. I've had a commit reverted twice in a row because a committer disagreed with me on a much smaller and obscure bug fix. I've endured verbal abuse. I've been reprimanded for not waiting to get the approval from Matz for committing an improvement to the wording of a warning (even though nobody thought the updated warning had any issue with it, or that it could potentially break any code out there). I envy you, and I really would like to understand □□

#16 - 07/19/2020 09:59 AM - Eregon (Benoit Daloze)

marcandre (Marc-Andre Lafortune) wrote in #note-15:

> Unless I'm mistaken, this behavior change was not approved by Matz (or anybody else), changes a behavior that dates back to Ruby 1.8, breaks (that we know of) activemodel and rubocop, doesn't even raise the right error and isn't tested, ...

Which change is that?
The first PR above, https://github.com/ruby/ruby/pull/3306 ?
That seems to have some tests.

> Yet when you propose that that commit remains as is and that someone else has to bring up the situation with Matz, no other committer seems confused by this.

I think the first PR should already have matz's or at least someone's else review.

Range is full of edge and special cases though, and there seems to be no global vision for it.
Maybe a good step here would be try to define the semantics we'd like for it. Obviously it's quite some work.

If reverting the first PR helps, maybe we should do that first?

#17 - 07/19/2020 10:02 AM - Eregon (Benoit Daloze)

BTW I think we should rewrite Range#max and others in pure Ruby.
It would be so much easier to read and discuss the semantics.
I find range_max() is very hard to read.

#18 - 07/19/2020 12:38 PM - jeremyevans0 (Jeremy Evans)

marcandre (Marc-Andre Lafortune) wrote in #note-15:

> jeremyevans0 (Jeremy Evans) wrote in #note-13:
>
> > I think this pull request should require matz approval because it is a deliberate tradeoff of correctness in order to keep compatibility. The correct behavior should be raising RangeError, as that is what an endless range returns.
>
> Well, (1.0..).max raises a RangeError while (1.0..Float::INFINITY).max returns Float::INFINITY; which is "correct"?

Range#max behavior for integer ranges is different than behavior for float ranges. (1..2.1).max and (1.0..2.1).max have different result even though 1 == 1.0. However, I see your point, that there is precedent for behavior to differ between endless ranges and ranges with infinite end.

> Unless I'm mistaken, this behavior change was not approved by Matz (or anybody else), changes a behavior that dates back to Ruby 1.8, breaks (that we know of) activemodel and rubocop, doesn't even raise the right error and isn't tested, and is a change that I and others disapprove of. Yet when you propose that that commit remains as is and that someone else has to bring up the situation with Matz, no other committer seems confused by this.

The behavior change in this issue is to fix an obvious bug, which is that (1..2.1).max returned 2.1 instead of 2. In general we don't require Matz's approval to fix obvious bugs. Were you against fixing that bug?

After the change was committed, that is when there were reports that the (1..Float::Infinity).max case broke. The only reason that case worked previously is it was relying on the previous bug (that Range#max returned a float end for an integer range). It's not like I removed a special case for it. I even worked on a pull request to add a special case for it for compatibility. However, since I think it makes Range#max behavior inconsistent for integer ranges, I think it is something that should be approved by Matz. Now, I may be wrong here. As committers go, I'm fairly junior. If another committer with more experience thinks it something that can be committed without approval from Matz, they can merge my pull request.

#19 - 07/19/2020 02:12 PM - marcandre (Marc-Andre Lafortune)

Eregon (Benoit Daloze) wrote in #note-16:

> marcandre (Marc-Andre Lafortune) wrote in #note-15:
>
> > Unless I'm mistaken, this behavior change was not approved by Matz (or anybody else), changes a behavior that dates back to Ruby 1.8, breaks (that we know of) activemodel and rubocop, doesn't even raise the right error and isn't tested, ...
>
> Which change is that?

I should have been more clear: by "this behavior change" I was referring exclusively to the change to (1..Float::INFINITY).max.

jeremyevans0 (Jeremy Evans) wrote in #note-13:

> ...obvious bug, which is that (1..2.1).max returned 2.1 instead of 2...
> Were you against fixing that bug?

Sorry I wasn't clear: I am not against that fix.

#20 - 07/19/2020 02:19 PM - marcandre (Marc-Andre Lafortune)

jeremyevans0 (Jeremy Evans) wrote in #note-18:

> If another committer with more experience thinks it something that can be committed without approval from Matz, they can merge my pull request.

Good, so since you don't mind, I will merge that PR that reinstates the previous behavior (again, to be clear, about how to treat Float::INFINITY end) that is both useful and relied upon, and we can discuss the proper behavior for (1.0..).max and other variants. □□

#21 - 07/19/2020 02:27 PM - jeremyevans (Jeremy Evans)

- Status changed from Open to Closed

Applied in changeset git|05bf811c2839628aaef3d565daedb28be80d47ef.

Special case Range#max for integer beginning and Float::Infinity end

Popular Ruby libraries such as Rails and Rubocop relying on the
previous behavior, even though it is technically a bug. The
correct behavior is probably raising RangeError, since that is what
an endless range raises.

Related to [Bug #17017]

#22 - 08/14/2020 04:09 PM - Dan0042 (Daniel DeLorme)

jeremyevans0 (Jeremy Evans) wrote in #note-18:

> The behavior change in this issue is to fix an obvious bug, which is that (1..2.1).max returned 2.1 instead of 2.

FWIW I consider the previous behavior correct. Intuitively I see 1..2.1 as a float range because one of the ends is a float. And so I expect (1..2.1).max to be equivalent to (1.0..2.1).max

#23 - 08/14/2020 05:54 PM - jeremyevans0 (Jeremy Evans)

Dan0042 (Daniel DeLorme) wrote in #note-22:

> jeremyevans0 (Jeremy Evans) wrote in #note-18:
>
> > The behavior change in this issue is to fix an obvious bug, which is that (1..2.1).max returned 2.1 instead of 2.
>
> FWIW I consider the previous behavior correct. Intuitively I see 1..2.1 as a float range because one of the ends is a float. And so I expect (1..2.1).max to be equivalent to (1.0..2.1).max

integer..float is currently treated as a integer range in all other respects. For example:

```
(1.0..2.1).to_a # TypeError (can't iterate from Float)
(1..2.1).to_a # => [1, 2]
```

So if we want to treat integer..float to be a float range, it will require changes far beyond #max and #minmax.

I think this is a fix for a bug/regression introduced in Ruby 1.9 due to an improper optimization:

```
$ ruby18 -ve 'p((1..2.1).max)'
ruby 1.8.7 (2013-06-27 patchlevel 374) [x86_64-openbsd]
2

$ ruby19 -ve 'p((1..2.1).max)'
ruby 1.9.3p551 (2014-11-13 revision 48407) [x86_64-openbsd]
2.1
```

#24 - 09/01/2020 05:13 AM - matz (Yukihiro Matsumoto)

Range class methods are classified in two. The ones that behave like Enumerable methods (defined by #each), and the others that behave like region (defined by the both ends).
I think #min and #max should belong to the latter. So the old behavior is preferable, that is (1..3.5).max #=> 3.5.

Matz.

#25 - 09/01/2020 02:49 PM - jeremyevans0 (Jeremy Evans)

matz (Yukihiro Matsumoto) wrote in #note-24:

> Range class methods are classified in two. The ones that behave like Enumerable methods (defined by #each), and the others that behave like region (defined by the both ends).
> I think #min and #max should belong to the latter. So the old behavior is preferable, that is (1..3.5).max #=> 3.5.

OK, I'll revert the related commits and also update the documentation to clarify the behavior.

Note that the previous behavior for exclusive integer ranges was like Enumerable and not like region ((1...3).max #=> 2, even though (1...3).include?(2.9) #=> true). I assume that the previous behavior for that should be kept, even if it is inconsistent.

#26 - 09/01/2020 05:53 PM - jeremyevans0 (Jeremy Evans)

I reverted the changes in this pull request, and updated the Range#max documentation to explain the behavior (de10a1f3583adeeffd7f8bcf8f276e0a626ffa0c).