

Ruby master - Feature #14799

Startless range

05/31/2018 07:56 AM - zverok (Victor Shepelev)

Status:	Closed
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	
Description	
<p>On introduction of endless range at #12912, "startless range" was discussed this way:</p> <p>sowieso (So Wieso): Not having the opposite (..5 and ..-2) feels like this is rather a hack than a thoroughly planned feature.</p> <p>duerst (Martin Dürst): I don't understand the need for a ..5 Range. The feature is called "endless range". Although mathematically, it's possible to think about startless ranges, they don't work in a program. Maybe some programming languages have ..5 as a shortcut for 0..5, but that's in any way a usual, bounded, range with a start and an end. It's conceptually totally different from 5.., which is a range with a start but no end, an unbound range.</p> <p>In the context of that ticket (ranges used mostly for slicing arrays) having ..5 was indeed hard to justify, but there are other cases when ..5 being -Infinity..5 is absolutely reasonable:</p> <pre>case release_date when ..1.year.ago puts "ancient" when 1.year.ago..3.months.ago puts "old" when 3.months.ago..Date.today puts "recent" when Date.today.. puts "upcoming" end</pre> <pre>log.map(&:logged_at).grep(..Date.new(1980)) # => outliers due to bad log parsing...</pre> <p>E.g., whenever case equality operator is acting, having startless range to express "below this value" is the most concise and readable way. Also, for expressing constants (mostly decorative, but very readable):</p> <pre># Celsius degrees WORK_RANGES = { ..-10 => :turn_off, -10..0 => :energy_saving, 0..20 => :main, 20..35 => :cooling, 35.. => :turn_off }</pre> <p>In addition, my related proposal #14784 suggests that this kind of ranges could be utilized by more powerful clamp too:</p> <pre>updated_at.clamp(..Date.today)</pre> <p>Uncertainty points:</p> <ul style="list-style-type: none">• Would it be hard to add to parser? I am not sure, I am not very good at it :(• Should .. be a thing? I guess not, unless there would be convincing real-life examples, which for me it is hard to think of.	
Related issues:	
Related to Ruby master - Bug #15745: There is no symmetry in the beginless ra...	Closed

Associated revisions

Revision 67422 - 04/03/2019 08:11 AM - mame (Yusuke Endoh)

Introduce beginless range [Feature#14799]

History

#1 - 05/31/2018 08:10 AM - shyouhei (Shyouhei Urabe)

No strong opinion on this. However let me leave one question: how should `Range#each` work for this kind of ranges?

#2 - 05/31/2018 09:14 AM - zverok (Victor Shepelev)

how should `Range#each` work for this kind of ranges?

Most probably it should not (the same as `Enumerable#reverse_each` or `last(5)` doesn't have a sense for already implemented endless ranges), just raise.

#3 - 05/31/2018 09:31 AM - shyouhei (Shyouhei Urabe)

zverok (Victor Shepelev) wrote:

how should `Range#each` work for this kind of ranges?

Most probably it should not (the same as `Enumerable#reverse_each` or `last(5)` doesn't have a sense for already implemented endless ranges), just raise.

Sounds reasonable, except it seems endless range does not raise for `#reverse_each` :)

```
zsh % ruby -ve '(1..).reverse_each {|i| p i }'  
ruby 2.6.0dev (2018-05-29 trunk 63514) [x86_64-darwin15]  
^CTraceback (most recent call last):  
  2: from -e:1:in `<main>'  
  1: from -e:1:in `reverse_each'  
-e:1:in `each': Interrupt
```

#4 - 05/31/2018 09:51 AM - zverok (Victor Shepelev)

except it seems endless range does not raise for `#reverse_each` :)

Funny!

Though, in fact, raises some big questions about `Enumerable` implementations. As far as I can understand, most of `Enumerable`'s methods aren't redefined in `Range`, which leaves a room for optimization (and sometimes a huge room): `last(N)` and `reverse_each` are just work by converting enumerable to an `Array`, and then taking the last elements. Probably, providing custom `#reverse_each` for where it is acceptable, and making `#last` rely on this could be both more effective, and produce more reasonable result in edge cases (like `open -- semi-infinite -- sequences`).

#5 - 05/31/2018 11:39 AM - mame (Yusuke Endoh)

- Assignee set to matz (Yukihiko Matsumoto)
- Status changed from Open to Assigned
- File `beginless-range.patch` added

I tried `begin-less range` once, and it caused many parser conflicts, so I gave up.

However, I've tried it again and created a patch with no conflicts.

A proof-of-concept patch is attached. It uses the damn lexer state, so I'd like nobu and yui-knk to review it.

Note that the main focus of this patch is `parse.y`, so we need more work to let many builtin methods to support `begin-less range`.

Matz, what do you think?

#6 - 05/31/2018 01:01 PM - janosch-x (Janosch Müller)

Rails devs could also make use of this, e.g. in [queries](#).

#7 - 06/04/2018 11:10 AM - DarkWiiPlayer (Dennis Fischer)

This does make sense when considering the reverse of the range, in other words, counting down from `N` to `-infinity`. This reasoning has two problems though:

- Ranges have no method that reverse them

- Ruby doesn't seem to like N..M ranges where M < N ((5..1).each {|x| puts x} simply prints 5..1 in ruby 2.5

Intuitively, counting down like that seems like a pretty common use case, but I cannot actually recall ever needing it, and it could also be simulated with a forward-counting range and just multiplying the values with -1 before using them, which shouldn't really have any considerable impact on speed.

EDIT: Overriding #each to raise an exception and only allowing #reverse_each with startless ranges might make them *somewhat* useful.

#8 - 06/14/2018 02:34 PM - znz (Kazuhiro NISHIYAMA)

I think version guard of ruby/spec is one of usages.

For example:

```
ruby_version_is "...2.6" do
  # ...
end
```

#9 - 06/14/2018 03:16 PM - Eregon (Benoit Daloze)

znz (Kazuhiro NISHIYAMA) wrote:

I think version guard of ruby/spec is one of usages.

For example:

```
ruby_version_is "...2.6" do
  # ...
end
```

Right, this would be a nice way to express version ranges like

```
ruby_version_is ..."2.6" do
  # ...
end
```

```
ruby_version_is "2.4"... do
  # ...
end
```

However for the case of ruby/spec, we'll have to wait until 2.5 is out of life to actually use those, as the syntax does not work on 2.5 and earlier.

#10 - 06/14/2018 10:30 PM - mame (Yusuke Endoh)

I agree that it is a good use case.

But notice that we will not be able to write `ruby_version_is ..."2.6" do` because it parses as `(ruby_version_is..."2.6") do`. We need to write parentheses: `ruby_version_is(..."2.6") do`.

#11 - 06/15/2018 05:14 PM - Eregon (Benoit Daloze)

[mame \(Yusuke Endoh\)](#) Ah, that is quite unfortunate (it does not look so nice with the parens), but thank you for mentioning the caveat. I guess much like Range literals in general, it's often needed to have parens around them.

#12 - 06/21/2018 09:02 AM - knu (Akinori MUSA)

Let me point out that with this patch applied, `(nil..nil).cover?(1)` would become returning true rather than raising an error. Note that `nil..nil` has been valid for all past versions of ruby.

#13 - 03/11/2019 05:15 AM - matz (Yukihiro Matsumoto)

I have some concerns, especially ones with syntax rules. Let us try it in the trunk, and see if it works well or not.

Matz.

#14 - 03/11/2019 05:29 AM - nobu (Nobuyoshi Nakada)

BDOT2 and BDOT3 don't seem necessary.

#15 - 04/03/2019 08:16 AM - mame (Yusuke Endoh)

- Status changed from Assigned to Closed

Committed at r67422. Please give it a try and let me know if you find any issue.

#16 - 04/20/2019 11:37 AM - naruse (Yui NARUSE)

- Related to Bug #15745: There is no symmetry in the beginless range and the endless range using `Range#inspect` added

Files

beginless-range.patch	3.55 KB	05/31/2018	mame (Yusuke Endoh)
-----------------------	---------	------------	---------------------