Ruby trunk - Feature #14784

Comparable#clamp with a range

05/23/2018 01:04 PM - zverok (Victor Shepelev)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Proposal

Allow "one-sided" clamp to limit only upper bound (and, ideally, only lower too).

Proposed implementation: allow clamp(begin..end) call sequence (without deprecating clamp(begin, end)), to take advantage from open-ended ranges with clamp(begin..).

Reasoning about range

I looked through #clamp discussion, but couldn't find there why syntax clamp(b, e) was preferred to clamp(b..e). The only one I could think of is possible confuse of how clamp(b..e) and clamp(b...e) behaviors should differ.

The problem becomes more important with the introduction of open-ended ranges. I believe this is pretty natural:

```
some_calculation.clamp(0..)     # now, I use clamp(0, Float::INFINITY)
timestamp.clamp(Date.today..)
# now, I typically use clamp(Date.today..INFINITE_FUTURE_DATE) with custom defined constant
```

Counter-arguments:

1. This is "one-sided", you can't do clamp(..Date.today). To this I can answer than from my experience "clamping only minimum" is more frequent, and if you need to clamp only maximum, most of the time there is some "reasonable minumum". Another idea is that maybe this is a proof why "start-less" ranges are necessary, after all, doubted here
2. Why not just leave current clamp(b, e) and allow clamp(b)? Answer: because when you see clamp(10), is it clamp(10, nil), or clamp(nil, 10) (yes, logically it is the first argument that is left, but from readability point of view it is not that obvious). Possible alternative: clamp(min: 0, max: 10), where you can omit any of two.
3. Why do you need one-sided clamp at all? Because alternatives is much more wordy, making reader think:

```
# with clamp
chain.of.calculations.clamp(0..)

# without clamp
v = chain.of.calculations
v < 0 ? 0 : v

# or, with yield_self (renamed to then)
chain.of.calculations.then { |v| v < 0 ? 0 : v }
```

Both alternatives "without #clamp" shows intentions much less clear.

History

#1 - 05/23/2018 08:44 PM - shevegen (Robert A. Heiler)

Considering that Ranges allow a ruby hacker to omit the end value,
for infinity/endless, since about ... a month or so, I think your
example makes sense in this regard, e. g.

```
begin .. end
```

being the same as:

```
begin ..
```

or
begin..

Perhaps also the converse, but I have to admit that
all these examples look very strange to my eyes. Like:

```
clamp(..Date.today)
```

I always look at it as if something is missing. Personally
I prefer explicit "begin .. end".

The:

```
clamp(min: 0, max: 10)
```

seems to be a nice API, in my opinion. At the least the names "min"
and "max" appear explicit and make sense (to me).

I agree, mostly for consistency, that if endless range has been
accepted, being able to do so via #clamp may seem a logical
continuation (to me). I am mostly neutral to the issue though,
as I do not (yet) use clamp in my own ruby code.

#2 - 05/24/2018 04:55 AM - nobu (Nobuyoshi Nakada)

zverok (Victor Shepelev) wrote:

> 1. Why do you need one-sided clamp at all? Because alternatives is much more wordy, making reader think:

Why not [chain.of.calculations, 0].max?

#3 - 05/24/2018 08:39 AM - zverok (Victor Shepelev)

nobu (Nobuyoshi Nakada)

> Why not [chain.of.calculations, 0].max?

Because this chain.of.calculations in reality could be something like

```
[paragraphs
  .group_by(&:speaker)
  .map { |speaker, paragraphs| paragraph.sort_by(&:length).first }
  .sort_by { |para| para.words.count }
  .first.speaker.name.length, 3].max
```

Jumping to start and end of this super-huge array while writing and reading is a pain.

So, now I'll probably write something like

```
paragraphs
  .group_by(&:speaker)
  .map { |speaker, paragraphs| paragraph.sort_by(&:length).first }
  .sort_by { |para| para.words.count }
  .first.speaker.name.length
  .yield_self { |len| [len, 3].max }
```

...which is OK-ish, but I never really liked how [value, MIN].max represents the idiom "limit this number to minimum possible value". The [value, MIN] somehow represents two values as equally important, while in fact one of them is "the main calculated value", and another one is "just one parameter of the calculation".

So, for me, this looks 200% better:

```
paragraphs
  .group_by(&:speaker)
  .map { |speaker, paragraphs| paragraph.sort_by(&:length).first }
  .sort_by { |para| para.words.count }
  .first.speaker.name.length
  .clamp(3..)
```

#4 - 07/18/2018 05:45 AM - akr (Akira Tanaka)

I feel this proposal is needlessly big: it needs range support for Comparable#clamp and startless range.
I think just supporting nil for 1st and 2nd argument of Comparable#clamp is enough.

#5 - 10/07/2018 12:05 PM - zverok (Victor Shepelev)

akr (Akira Tanaka) The proposal is "Comparable#clamp with a range". It also justifies the possible need for a startless range, which is extracted to #14799.

I believe that clamp(3..), clamp(..10) is 200% more Ruby-idiomatic than clamp(3, nil), clamp(nil, 10).
But even WITHOUT startless range, I believe that "clamp accepting Range" is just natural.

#6 - 10/10/2018 06:29 AM - matz (Yukihiro Matsumoto)

Please don't put multiple proposals in one issue. It's hard for us to tell which is important clamp to accept ranges or having one-sided clamp. This kind of mixture leads to our confusion that hinders final decision.

Matz.

#7 - 10/10/2018 06:44 AM - shyouhei (Shyouhei Urabe)

- Subject changed from One-sided Comparable#clamp to Comparable#clamp with a range

So, let's focus on making Comparable#clamp accept Ranges. Subject updated.

#8 - 10/11/2018 12:26 AM - akr (Akira Tanaka)

If Ruby support clamp(range), the behavior of clamp(b...e) should be considered.

What returns x.clamp(b...e) when e < x ?

10.clamp(0...20) would return 19.

But There is Rational.
How about 10r.clamp(0r...20r) ?
"Maximum Rational value less than 20" is not exist.

#9 - 04/30/2019 04:38 PM - jonathanhefner (Jonathan Hefner)

I agree this would be a useful feature. I have a library that implements this as Comparable#at_least(min) and Comparable#at_most(max), but clamp(min..) and clamp(..max) feel more idiomatic (although they do require an extra object allocation).

> 10.clamp(0...20) would return 19.
> But There is Rational.
> How about 10r.clamp(0r...20r) ?

This is also a concern. I think the most appropriate behavior, unfortunately, is to raise an ArgumentError if range.end && range.exclude_end?. Maybe Integer#clamp could override this behavior only when the range end is also an Integer, but the convenience may not justify the extra complexity and possible inconsistency.