Ruby master - Feature #14784

Comparable#clamp with a range

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

Status:	Closed
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.

Associated revisions

Revision 929d5fd3 - 10/15/2019 04:42 PM - nobu (Nobuyoshi Nakada)

Comparable#clamp with a range [Feature #14784]

Revision 42c652d1 - 10/25/2019 02:30 PM - nobu (Nobuyoshi Nakada)

Fixed range argument condition [Feature #14784]

Allows a beginless/endless range, and an end-exclusive range
unless the receiver is smaller than its end.

Revision cf934413 - 10/26/2019 03:52 AM - nobu (Nobuyoshi Nakada)

Raise on end-exclusive ranges [Feature #14784]

Raises an error on end-exclusive ranges unless endless, regardless
the receiver.

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.

#10 - 08/16/2019 07:29 AM - svoop (Sven Schwyn)

Here's a real life use case for clamp with Range support:

Our Rails app has quite a few app settings which define ranges of permitted values e.g. goal_range = (10..1000). Values outside of those ranges are clamped which leads to things like:

```
value.clamp(Rails.application.config.x.goal_range.min, Rails.application.config.x.goal_range.max)
```

It is IMO more readable if clamp accepts Ranges:

```
value.clamp(Rails.application.config.x.goal_range)
```

#11 - 08/29/2019 06:56 AM - matz (Yukihiro Matsumoto)

Accepted. It should raise an error on end-exclusive ranges (...).

Matz.

#12 - 10/15/2019 04:42 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset git|929d5fd3b99c1413f737ff16cf0680698036e60f.

Comparable#clamp with a range [Feature #14784]

#13 - 10/25/2019 06:48 AM - zverok (Victor Shepelev)

nobu (Nobuyoshi Nakada) is there any chance you can reconsider your implementation?

The important justification for the proposal was using #clamp with open ranges (begin- and end-less), but your implementation just rejects them all (with misleading error message).

```
1.clamp(0..) # ArgumentError (cannot clamp with an exclusive range)
1.clamp(2..) # ArgumentError (cannot clamp with an exclusive range)
1.clamp(..2) # ArgumentError (cannot clamp with an exclusive range)
1.clamp(..0) # ArgumentError (cannot clamp with an exclusive range)
1.clamp(0...3) # ArgumentError (cannot clamp with an exclusive range)
```

Here is behavior of my implementation (I developed it yesterday, because Redmine doesn't send "Closed via changeset" notifications so I believed the proposal still waits for implementation):

```
1.clamp(0..) #=> 1
1.clamp(2..) #=> 2
1.clamp(..2) #=> 1
1.clamp(..0) #=> 0
1.clamp(0...3) #=> 1
1.clamp(-1...0)
# ArgumentError: #clamp with excluding end can't clamp from top -- the only prohibited situation
```

I believe this behavior is much more PoLS and useful for lot of cases.

Here is the implementation:

```
VALUE cmp_clamp2(VALUE x, VALUE min, VALUE max)
{
    int c;

    if (cmpint(min, max) > 0) {
    rb_raise(rb_eArgError, "min argument must be smaller than max argument");
    }

    c = cmpint(x, min);
    if (c == 0) return x;
    if (c < 0) return min;
    c = cmpint(x, max);
    if (c > 0) return max;
    return x;
}

VALUE cmp_clamp_range(VALUE x, VALUE range)
{
    VALUE beg, end;

    if (!rb_obj_is_kind_of(range, rb_cRange)) {
        rb_raise(rb_eArgError, "#clamp with 1 argument expects Range");
    }

    beg = RANGE_BEG(range);
    end = RANGE_END(range);

    if (!NIL_P(beg) && !NIL_P(end) && cmpint(beg, end) > 0) {
        rb_raise(rb_eArgError, "range begin is larger than range end");
    }
    if (!NIL_P(beg) && cmpint(x, beg) <= 0) {
        return beg;
    }
    if (!NIL_P(end) && cmpint(x, end) >= 0) {
        if (RANGE_EXCL(range)) {
            rb_raise(rb_eArgError, "#clamp with excluding end can't clamp from top");
        }
        return end;
    }

    return x;
}


/*
 *  call-seq:
 *     obj.clamp(min, max) ->  obj
 *     obj.clamp(range)    ->  obj
 *
 * In <code>(min, max)</code> form, returns _min_ if _obj_
 * <code><=></code> _min_ is less than zero, _max_ if
 * _obj_ <code><=></code> _max_ is greater than zero and
 * _obj_ otherwise.
 *
 *     12.clamp(0, 100)         #=> 12
 *     523.clamp(0, 100)        #=> 100
 *     -3.123.clamp(0, 100)     #=> 0
 *
 *     'd'.clamp('a', 'f')      #=> 'd'
 *     'z'.clamp('a', 'f')      #=> 'f'
 *
 * In <code>(range)</code> form, returns <i>range.begin</i> if <i>obj</i>
 * <code><=></code> <i>range.begin</i> is less than zero,
 * <i>range.end</i> if <i>obj</i> <code><=></code> <i>range.end</i>
 * is greater than zero and <i>obj</i> otherwise.
 *
 *     12.clamp(0..100)          #=> 12
 *     523.clamp(0..100)         #=> 100
 *     -3.123.clamp(0..100)      #=> 0
 *
 *     # Works with endless/beginless ranges:
 *     -20.clamp(0..)            #=> 0
 *     523.clamp(..100)          #=> 100
 *
 *     # When range excludes end, and the value is more than end,
 *     # an exception is raised.
 *     523.clamp(0...100)       # ArgumentError: #clamp with excluding end can't clamp from top
 *
 */

static VALUE
cmp_clamp(int argc, const VALUE *argv, VALUE x)
{
    rb_check_arity(argc, 1, 2);
    if (argc == 2) {
        return cmp_clamp2(x, argv[0], argv[1]);
    }
    return cmp_clamp_range(x, argv[0]);
}
```

...and tests for it:

```
  def test_clamp_range
    cmp->(x) do 0 <=> x end
    assert_equal(1, @o.clamp(1..2))
    assert_equal(-1, @o.clamp(-2..-1))
    assert_equal(@o, @o.clamp(-1..3))

    assert_equal(1, @o.clamp(1..1))
    assert_equal(@o, @o.clamp(0..0))

    assert_equal(1, @o.clamp(1..))
    assert_equal(-1, @o.clamp(..-1))
    assert_equal(@o, @o.clamp(-2...1))
    assert_raise_with_message(ArgumentError, "#clamp with excluding end can't clamp from top") {
      @o.clamp(-2...-1)
    }

    assert_raise_with_message(ArgumentError, 'range begin is larger than range end') {
      @o.clamp(2..1)
    }

    assert_raise_with_message(ArgumentError, '#clamp with 1 argument expects Range') {
      @o.clamp(2)
    }
  end
```

#14 - 10/25/2019 01:30 PM - nobu (Nobuyoshi Nakada)

Sounds reasonable, and matz says “an error on end-exclusive ranges” at https://bugs.ruby-lang.org/issues/14784#change-81254 and ditto in the log of 20190829 dev meeting, but nothing about beginless/endless ranges.
Seems I had been confused somewhat.
Thank you for pointing out.

#15 - 10/25/2019 02:27 PM - Dan0042 (Daniel DeLorme)

> 1.clamp(0...3) #=> 1
> 1.clamp(-1...0) # ArgumentError: #clamp with excluding end can't clamp from top -- the only prohibited situation

I don't think it's a good idea for the error to depend on the value being clamped. When Matz said "It should raise an error on end-exclusive ranges" I'm pretty sure it was supposed to mean a simple argument check regardless of the value. Otherwise that code *will* eventually raise an error. Or was that the purpose? In that case the error message should be more like "value out of bounds". But I don't think that's the role/purpose of clamp.

#16 - 11/26/2019 11:01 AM - zverok (Victor Shepelev)

> Sounds reasonable, and matz says “an error on end-exclusive ranges” at https://bugs.ruby-lang.org/issues/14784#change-81254 and ditto in the log of 20190829 dev meeting, but nothing about beginless/endless ranges.

nobu (Nobuyoshi Nakada) Will it be fixed before 2.7 final? Can I help somehow?

#17 - 11/26/2019 03:06 PM - nobu (Nobuyoshi Nakada)

zverok (Victor Shepelev), I think cf934413 made it work as you described, doesn't it?
If something is wrong, let me know.

#18 - 11/26/2019 03:25 PM - zverok (Victor Shepelev)

nobu (Nobuyoshi Nakada) Ugh, sorry, I didn't check the latest one (just judged from the ticket state). All is awesome! Thank you.