# Ruby - Feature #12133

## Ability to exclude start when defining a range

03/01/2016 08:31 PM - slash_nick (Ryan Hosford)

| | |
|---|---|
| **Status:** | Feedback |
| **Priority:** | Normal |
| **Assignee:** | |
| **Target version:** | |

**Description**

An intuitive, approach would be to allow defining ranges like so:

```
[1..10]
[1..10)
(1..10]
(1..10)
```

... where a square bracket indicates boundary inclusion and a parenthesis represents boundary exclusion. The syntax there is obviously not going to work, but it demonstrates the idea.

A more feasible, still intuitive, solution might look like the following

```
(1..10)                 # [1..10]
(1...10)                # [1..10) ... Alternatively: (1..10).exclude_end
(1..10).exclude_start   # (1..10]
(1...10).exclude_start # (1..10) ... Alternatively: (1..10).exclude_start.exclude_end
```

For consistency, I think we'd also want to add #exclude_start? & #exclude_end methods.

### History

#### #1 - 03/15/2016 06:57 AM - shyouhei (Shyouhei Urabe)

*- Status changed from Open to Feedback*

Do you have any practical situation where this is useful? The proposed grammar is ultra-hard to implement at sight (if not impossible). You are advised to show us why this feature is worth tackling.

#### #2 - 03/15/2016 08:14 AM - nobu (Nobuyoshi Nakada)

Combination of different parens would be problematic for many reasons, e.g., editors.
Maybe, neko operator in Perl6?

#### #3 - 03/24/2016 04:30 PM - slash_nick (Ryan Hosford)

Please accept apologies for confusion caused by non-matching parenthesis/brackets -- this notation is just a standard way of denoting intervals in mathematics (ref. ISO 31-11 & Wikipedia entry for including excluding endpoints in intervals).

I'm not proposing we support non-matching parenthesis or brackets. Here's what I'm proposing:

```
# Including both endpoints ("a" and "b").
(a..b)

# Excluding endpoint "b"
(a...b) # OR
(a..b).exclude_end

# Excluding endpoint "a"
(a..b).exclude_start

# Excludes both endpoints ("a" and "b").
(a...b).exclude_start # OR
(a..b).exclude_start.exclude_end
```

I think this is useful because it would give the ruby language a more complete implementation of ranges/intervals. I recently built a feature that requires me to cover the range from **a** to **b** with 2 to 5 sub-ranges, validating that there are no gaps and no overlaps in the sub-ranges. I also needed to be flexible with choosing into which sub-range an endpoint should fall. Examples: If your systolic blood pressure is 120 mmHg, do I say you are in a normal range or do I say you have Prehypertension? If your HBA1C is 6.5%, are you in a pre-diabetic condition or do you have diabetes?

I believe I could've written a simpler solution with a more complete range implementation.

Thank you, Shyouhei and Nobu, for your responses.

#### #4 - 03/25/2016 08:43 PM - Eregon (Benoit Daloze)

Ryan Hosford wrote:

> I'm not proposing we support non-matching parenthesis or brackets. Here's what I'm proposing:

Am I missing something or the ability to exclude start could be done with just (start.next..last) ?
If that's the case, it seems to be too much of an effort to add new syntax, or to complicate further Range's implementation.

#### #5 - 03/25/2016 10:12 PM - Eregon (Benoit Daloze)

On Fri, Mar 25, 2016 at 9:58 PM, Matthew Kerwin  wrote:

> That's only true for ranges of discrete values. A range like: (0.0..1.0).exclude_first would look absolutely horrible as 0.0.next_float..1.0, and I don't think there's even an equivalent for Rationals or Times.

Ah right, in the case a Range is used mostly for #include? and co, then .next is not really a good fix.
Please disregard my comment then.
Maybe Range should use another method for iterating over values so it would behave more consistently.

#### #6 - 03/26/2016 01:38 PM - naruse (Yui NARUSE)

Ryan Hosford wrote:

> I think this is useful because it would give the ruby language a more complete implementation of ranges/intervals. I recently built a feature that requires me to cover the range from **a** to **b** with 2 to 5 sub-ranges, validating that there are no gaps and no overlaps in the sub-ranges. I also needed to be flexible with choosing into which sub-range an endpoint should fall. Examples: If your systolic blood pressure is 120 mmHg, do I say you are in a normal range or do I say you have Prehypertension? If your HBA1C is 6.5%, are you in a pre-diabetic condition or do you have diabetes?

On such case, an implementation will be something like following.
If you want more easy to read one, which declares inclusive-or-exclusive range and validates there's no duplication,
it seems something different one from current simple range object, but more complex one.

```
case ARGV.shift.to_r
when 0..20.1.to_r
  puts "low"
when 20.1.to_r...23.4.to_r
  puts "middle"
when 23.4.to_r..99
  puts "high"
end
```

#### #7 - 03/26/2016 03:29 PM - slash_nick (Ryan Hosford)

Yui NARUSE wrote:

> On such case, an implementation will be something like following.
> If you want more easy to read one, which declares inclusive-or-exclusive range and validates there's no duplication,
> it seems something different one from current simple range object, but more complex one.
>
> ```
> case ARGV.shift.to_r
> when 0..20.1.to_r
>   puts "low"
> when 20.1.to_r...23.4.to_r
>   puts "middle"
> when 23.4.to_r..99
>   puts "high"
> end
> ```

Here you've used the case statement to exclude start on the middle range. The problem is the code is largely static while the values of the endpoints (and which ranges should include those endpoints) may change. Physicians may all agree that 20.1 is in a low range today, but tomorrow?

I think this workaround could be made to work ( if 'middle' range is exclude start, 'low' needs #..; if 'middle' range is not exclude start, 'low' needs #...) but it would be tricky, unintuitive, complicated, etc..

#### #8 - 03/27/2016 04:57 PM - naruse (Yui NARUSE)

Ryan Hosford wrote:

> Here you've used the case statement to exclude start on the middle range. The problem is the code is largely static while the values of the endpoints (and which ranges should include those endpoints) may change. Physicians may all agree that 20.1 is in a low range today, but tomorrow?
>
> I think this workaround could be made to work ( if 'middle' range is exclude start, 'low' needs #..; if 'middle' range is not exclude start, 'low' needs #...) but it would be tricky, unintuitive, complicated, etc..

What I want to ask is "Is the range extension really helps developers?".

I can implement the same behavior with separating data and code like following:

```
x = 23.5r
ranges = [(0..20.1r), (20.1...23.4r), (23.4r..99)]
values = %w[low middle high]
idx = ranges.index{|range|range.include?(x)}
puts values[idx]
```

Does the new feature enables more clear code?

#### #9 - 03/29/2016 05:45 AM - slash_nick (Ryan Hosford)

Here's what I would've written: (see: sample range_sections)

```
def which_range?(value)
  range = nil

  range_sections.each do |rs|
    range = Range.new(rs.start, rs.end, rs.end_condition != "=", rs.start_condition != "=")

    break if range.include?(value)

    range = nil
  end

  range
end
```

And here is what I can write with what's possible now:

```
def which_range?(value)
  range = nil

  range_sections.each do |rs|
    range = Range.new(rs.start, rs.end, rs.end_condition == "=")

    break if range.include?(value) && (value == range.begin && rs.start_condition != "=")

    range = nil
  end

  range
end
```

I believe the first is more clear. The real benefit to the first is that the returned range would convey the appropriate boundary information.

What this issue is asking:

- Is #exclude_end? meaningful and useful?
- Would #exclude_start? also be meaningful and useful?
- If #exclude_end? is meaningful and useful, should we add #exclude_start??

If we decide we want it, we'd need to consider another thing:

- Do we need a shorthand? (something like the neko operator as Nobu mentioned). Examples:

```
^1..10^ #=> something we can't do now, but equivalent to ^1...10: 1  through 10, excluding both 1 and 10
10...20 #=> something we can do now, it is equivalent to 10..20^: 10 through 20, excluding 20
20..30  #=> something we can do now, it is equivalent to 10..20:  20 through 30
```

#### #10 - 11/04/2017 01:09 PM - shevegen (Robert A. Heiler)

> where a square bracket indicates boundary inclusion and a parenthesis represents boundary exclusion.

I do not like the suggestion but I hope you are not too discouraged.

The reason is ... actually a weird one. I have fairly bad eyesight, and I'd appreciate if I would not have to look hard between differences of [] and () for Ranges. There is already, IMO, a lot of additional meanings in Ruby; Array [], Hash-like notation {} and blocks, and to be completely honest, I'd rather like it if ruby would not add too many hard-to-differ syntax situations in general.

I think matz once said that about the lonely person staring at a dot operator, that the notation foo&.bar() was slightly harder to read than foo.&bar() or something like that. I actually think that both notations aren't that easy to read but I agree that one style may be a bit harder than the other.

Another reason why I dislike the proposal is, well ... take perl.

Perl used a lot of the $ variables such as $: $& and so forth. And while this can be useful (I use $1 $2 for regexes a lot, for example, but they are different because the numbers are easy for my mind to map to) since you don't have to type much, I find them very difficult to remember offhand. The english variable names are a bit easier to remember but also not ... perfect. The reason I mention this is because your suggestion started with "An intuitive, approach", and to be honest, to me it is not intuitive at all that [] would behave differently than () for Ranges.

I'd much prefer to simply keep the way how Ranges worked in ruby.

#### #11 - 04/07/2021 06:49 AM - arimay (yasuhiro arima)

Definition of a range object that exclude a start position, and its shorthand notation.

Since we have #exclude_end? we also want #exclude_begin? .

```
class Range
  def initialize(begin, end, exclude_end=false, exclude_begin=false)
  def exclude_begin?
```

Shorthand, by known as the neko operator.

```
notation        meaning         new
b..e            b <= v <= e     Range.new(b, e)
b..^e           b <= v <  e     Range.new(b, e, true) # the same as "b...e"
b^..e           b <  v <= e     Range.new(b, e, false, true)
b^..^e          b <  v <  e     Range.new(b, e, true,  true)

with nil.

..e                 v <= e      Range.new(nil, e)
^..e                v <= e      Range.new(nil, e)
..^e                v <  e      Range.new(nil, e, true)
^..^e               v <  e      Range.new(nil, e, true)
b..             b <= v          Range.new(b, nil, false, false)
b..^            b <= v          Range.new(b, nil, false, false)
b^..            b <  v          Range.new(b, nil, false, true)
b^..^           b <  v          Range.new(b, nil, false, true)
```

The priority of the range operators is the same.

```
< Higher >
||
..  ...  ..^  ^..  ^..^
?:
< Lower >
```

Situation: Notation.

```
range  =  0^..100     # 0    < v <=  100
range  =  100^..1000  # 100  < v <= 1000
range  =  1000^..     # 1000 < v
```

Situation: Pass to argument for method call.

```
[ 0^..100,  100^..1000,  1000^.. ].each do |range|
  pp  Items.where( price: range ).all
end
```

Situation: Build expression.(e.g. query script)

```
def build( v, range )
  b  =  range.begin
  e  =  range.end
  exprs  =  []
  if range.exclude_begin?
    exprs  <<  "#{b} < #{v}"
  elsif b
    exprs  <<  "#{b} <= #{v}"
  end
  if range.exclude_end?
    exprs  <<  "#{v} < #{e}"
  elsif e
    exprs  <<  "#{v} <= #{e}"
  end
  exprs.join(" and ")
end
```

The new feature will enables more clear code.
We believe this feature will be very useful for developers.