Ruby - Feature #20080

Introduce #bounds method on Range

12/22/2023 04:26 PM - stuyam (Stuart Yamartino)

Status:	Feedback
Priority:	Normal
Assignee:	
Target version:	

Description

Followup Reference: #20027

Update 1/11/24: (based on many wonderful suggestions!)

1. Call the method #bounds.

```
first, last = (1..300).bounds # => [1, 300]
first, last = (300..1).bounds # => [300, 1]
first, last = (..300).bounds # => [nil, 300]
first, last = (1..).bounds # => [1, nil]
```

1. Add exclude_end? support so re-hydration of Range works:

```
b = (1..2).bounds  #=> [1,2]
Range.new(*b)      #=> 1..2

b = (1...2).bounds #=> [1,2,true]
Range.new(*b)      #=> 1...2
```

I did a better job of outlining use cases in this comment below so I will let that speak for itself:
https://bugs.ruby-lang.org/issues/20080#note-3

Update: 2/13/24
Browsing the ruby codebase I noticed that the #as_json method on Range (when you require 'json/add/range') does something similar to what the #bounds method we are describing is doing:
https://github.com/ruby/ruby/blob/master/ext/json/lib/json/add/range.rb#L34

```
{
  JSON.create_id  => self.class.name,
  'a'             => [ first, last, exclude_end? ]
}
```

This tells me we are on the right track. Though the difference here is that exclude_end? is always included. I am thinking the #bounds should maybe always be including the exclude_end? piece rather than only include it if it's true. I am inclined to suggest always including the exclude_end? regardless of if it is true or false to avoid confusion or people ever calling #last on the #bounds results thinking it is the #last value where it could be the #last or #exclude_end? value depending on the type or range. This would still satisfy all of the use cases outlined in this comment: https://bugs.ruby-lang.org/issues/20080#note-3

Therefore I think #bounds should just always return: [first, last, exclude_end?]

Original Proposal:
This feature request is to implement a method called #begin_and_end on Range that returns an array of the first and last value stored in a range:

```
(1..300).begin_and_end #=> [1, 300]

first, last = (300..1).begin_and_end
first #=> 300
last #=> 1
```

I believe this would be a great addition to Ranges as they are often used to pass around a single object used to hold endpoints, and this allows easier retrieval of those endpoints.
This would allow easier deconstruction into start and end values using array deconstruction as well as a simpler way to serialize to a more primitive object such as an array for database storage.

This implementation was suggested by @mame (Yusuke Endoh) in my initial feature suggestion regarding range deconstruction: https://bugs.ruby-lang.org/issues/20027

This implementation would work similar to how #minmax works where it returns an array of two numbers, however the difference is that #minmax doesn't work with reverse ranges as @Dan0042 pointed out in the link above:

```
(1..42).minmax #=> [1, 42]
(42..1).minmax #=> [nil, nil]
```

History

#1 - 12/22/2023 05:03 PM - Dan0042 (Daniel DeLorme)

Can you show an example use case that demonstrates the value of the feature?

Because
first, last = (300..1).begin_and_end is simpler as first, last = 300, 1
first, last = r.begin_and_end might as well be first, last = r.first, r.last
or just use r.first and r.last directly instead of local variables

#2 - 12/22/2023 05:30 PM - ufuk (Ufuk Kayserilioglu)

I agree that this would be a good method to add for cases where one is handed a Range instance and accessing the bounds of that range is needed.

However, I think the name should be #bounds and it should work as:

```
(1..300).bounds # => [1, 300]
(1...300).bounds # => [1, 300]
(..300).bounds # => [nil, 300]
(1..).bounds # => [1, nil]
(nil..).bounds # => [nil, nil]

(300..1).bounds # => [300, 1]
(300...1).bounds # => [300, 1]
(300..).bounds # => [300, nil]
```

#3 - 12/22/2023 05:55 PM - stuyam (Stuart Yamartino)

@ufuk (Ufuk Kayserilioglu) I like #bounds as a name also, great suggestion, let's try that. And thank you for showing beginless and endless examples, I agree with that usage.

@Dan0042 Sorry for the bad example, I didn't show a good use case I just showed show I thought it would work. Here are a few use cases:

Use Case 1: Query filter, array deconstruction example

```
def filter_by_date_range(date_range)
  start_date, end_date = date_range.bounds
  where('start > ? AND end < ?', start_date, end_date)
end
```

(I know rails supports ranges in ActiveRecord but I have needed this for more manual queries)

Use Case 2. Serialize a range to store as an array in a database column:

```
# assumes a table with a `range_column` of type jsonb[] (array)
def store_in_table
  SomeTable.insert('range_column', range.bounds)
end
```

Use Case 3: Convert array of ranges to array of array bounds:

```
range_array = [(1..10), (10..20), (20..30)]
bounds_array = range_array.map(&:bounds) #=> [[1, 10], [10, 20], [20, 30]]
```

Up until now you can only every "deserialize" the data out of a range into other parts using #begin and #end but have never been able to do it in one go. Sometimes you want just one value but often you want both. Often a range is passes between methods to keep the data as a single object especially if they are contextual to each other. For example, rather than passing around start_date and end_date as method params through a bunch of methods, you can just pass around a date_range. But in that case the range is just a way of keeping those values logically together. Or if one is nil such as a beginless or endless range it is more clear when they are kept together. In the end though, the start_date and end_date can be easily pulled out using array deconstruction with the #bounds method.

#4 - 12/22/2023 06:35 PM - Dan0042 (Daniel DeLorme)

If it's for serialization wouldn't you also want to know exclude_end?

```
b = (1..2).bounds  #=> [1,2]
Range.new(*b)      #=> 1..2

b = (1...2).bounds #=> [1,2,true]
Range.new(*b)      #=> 1...2
```

#5 - 12/22/2023 06:49 PM - shan (Shannon Skipper)

An aside, but with Enumerator::ArithmeticSequence, last give you the value excluding end but not so with Range.

```
(1...2).last
#=> 2

((1...2) % 1).last
#=> 1
```

#6 - 12/22/2023 06:57 PM - stuyam (Stuart Yamartino)

@Dan0042 great idea! At first I was against this because I thought it would make deconstruction harder but it actually wouldn't because deconstruction would work the same. I was thinking the second value if deconstructed would be an array like [2, true] but that would only be if you used a star * during deconstruction. I'm liking this...

```
a = (1..2).bounds  #=> [1,2]
b = (1...2).bounds #=> [1,2,true]

Range.new(*a)      #=> 1..2
Range.new(*b)      #=> 1...2

start_num, end_num = a #=> start_num = 1, end_num = 2
start_num, end_num, excluded_end = a #=> start_num = 1, end_num = 2, excluded_end = nil
start_num, end_num = b #=> start_num = 1, end_num = 2
start_num, end_num, excluded_end = b #=> start_num = 1, end_num = 2, excluded_end = true
start_num, *remaining = b #=> start_num = 1, remaining = [2, true]
```

My only thought then is should there be an option to always include exclude_end? even when it is false like (1..2).bounds(true) #=> [1,2,false] so you would get false rather than nil in the array deconstruction example above.
Conversely (1...2).bounds(false) #=> [1,2] if you wanted to not include the excluded_end??

#7 - 12/23/2023 02:39 AM - rubyFeedback (robert heiler)

I have no particular opinion on the suggested feature itself,
but I agree that .bounds() is a better API / name than
.begin_and_end(), even though I understand the rationale
between the latter name making it more explicit. Ruby often
tries to prefer terse names, when that is possible and makes
sense.

#8 - 12/23/2023 06:06 PM - stuyam (Stuart Yamartino)

- Description updated

#9 - 12/23/2023 06:07 PM - stuyam (Stuart Yamartino)

- Description updated

#10 - 01/02/2024 03:31 PM - stuyam (Stuart Yamartino)

- Subject changed from Implement #begin_and_end method on Range to Implement #bounds method on Range

- Description updated

#11 - 01/02/2024 03:31 PM - stuyam (Stuart Yamartino)

- Subject changed from Implement #bounds method on Range to Introduce #bounds method on Range

#12 - 01/17/2024 04:31 PM - stuyam (Stuart Yamartino)

- Description updated

#13 - 01/18/2024 03:26 AM - hsbt (Hiroshi SHIBATA)

@stuyam Can you add this proposal to next dev-meeting? https://bugs.ruby-lang.org/issues/20075#note-9 was after the deadline and will not be discussed.

#14 - 01/23/2024 06:08 PM - stuyam (Stuart Yamartino)

@hsbt (Hiroshi SHIBATA) I just added to the next meeting, thank you for letting me know! https://bugs.ruby-lang.org/issues/20193#note-4

#15 - 01/27/2024 05:34 PM - AMomchilov (Alexander Momchilov)

Could we implement this as #deconstruct, so Ranges can support destructuring?

```
class Range
  def deconstruct = [self.begin, self.end]
end

case 1..2
in [1, Integer => upper]
  p "matched: #{upper}"
else
  p "not matched"
end
```

#16 - 01/28/2024 02:07 AM - Dan0042 (Daniel DeLorme)

AMomchilov (Alexander Momchilov) wrote in #note-15:

> Could we implement this as #deconstruct, so Ranges can support destructuring?

This was rejected in the original ticket from which this one originated: #20027#note-3

#17 - 01/28/2024 04:12 PM - zverok (Victor Shepelev)

@Dan0042 To be fair, that ticket seems to have rejected "old-style" deconstruction mainly (b, e = range). The possibility of #deconstruct is mentioned in one of the comments, but rejection is more vague on it.

#18 - 01/29/2024 07:44 PM - Dan0042 (Daniel DeLorme)

Oh you're right, it seems like I mixed up the #deconstruct comment in note-2 with the rejection notice in note-3.
Although imho it doesn't feel intuitive to pattern-match a range like that. I would expect to able to do 1..8 in [*,5,*]. So for this case I would rather use (1..2).bounds in [1, Integer => upper]

#19 - 02/13/2024 07:24 PM - stuyam (Stuart Yamartino)

- Description updated

#20 - 02/14/2024 05:12 AM - matz (Yukihiro Matsumoto)

What ever it is, at least it's not "bounds" especially when a range excludes end.
Maybe we seek another name (or behavior), if we really need to add the feature.

Matz.

#21 - 02/15/2024 10:23 PM - stuyam (Stuart Yamartino)

Thanks for the feedback @matz (Yukihiro Matsumoto)! Is it the word bounds that you don't like in relation to the start and end values of a range? I personally think bounds or a boundary can be considered inclusive or exclusive which is why including exclude_end? as part of the array is useful context. But it really comes down to how you might define bounds so I understand if you feel it doesn't fit. Do you think deconstruct might make sense then to support pattern matching or just manually calling it for serialization or array deconstruction? Admittedly I am not super familiar with how pattern matching works so Im not as clear on how that part would work.

I still feel this would be useful for serializing ranges and deconstructing values out of ranges, but I understand if people don't see the value.

#22 - 02/15/2024 11:45 PM - matz (Yukihiro Matsumoto)

Actually, I don't see the clear benefit of the proposal. first, last = range.bounds can be first = range.begin; last = range.end, and Range.new(*range.bounds) can be range.dup.
By adding bounds the code could become a little concise but only just.

In addition, I don't think the name bounds describe the behavior (returning begin, end **and** exclude_end?).

Matz.

#23 - 03/14/2024 08:57 AM - mame (Yusuke Endoh)

- Status changed from Open to Feedback