Ruby master - Feature #12648

`Enumerable#sort_by` with descending option

08/02/2016 09:57 AM - sawa (Tsuyoshi Sawada)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

I would like to pass an optional argument to Enumerable#sort_by or Enumerable#sort_by! to allow descending sort. When the sort key is singular, this could be done by passing a single optinal boolean variable that represents ascending when false (default) and descending when true:

```
[3, 1, 2].sort_by(&:itself)        # => [1, 2, 3]
[3, 1, 2].sort_by(false, &:itself) # => [1, 2, 3]
[3, 1, 2].sort_by(true, &:itself)  # => [3, 2, 1]
```

When there are multiple sort keys, corresponding numbers of arguments should be passed:

```
[3, 1, 2, 0].sort_by{|e| [e % 2, e]}                # => [0, 2, 1, 3]
[3, 1, 2, 0].sort_by(false, false){|e| [e % 2, e]} # => [0, 2, 1, 3]
[3, 1, 2, 0].sort_by(false, true){|e| [e % 2, e]}  # => [2, 0, 3, 1]
[3, 1, 2, 0].sort_by(true, false){|e| [e % 2, e]}  # => [1, 3, 0, 2]
[3, 1, 2, 0].sort_by(true, true){|e| [e % 2, e]}   # => [3, 1, 2, 0]
```

Related issues:

Related to Ruby master - Feature #15725: Proposal: Add Array#reverse_sort, #r...	**Rejected**

History

#1 - 08/02/2016 10:00 AM - sawa (Tsuyoshi Sawada)

When the number of arguments passed is less than the sort keys, sort should be ascended or descended at a higher array level.

```
[3, 1, 2, 0].sort_by{|e| [e % 2, e]}        # => [0, 2, 1, 3]
[3, 1, 2, 0].sort_by(false){|e| [e % 2, e]} # => [0, 2, 1, 3]
[3, 1, 2, 0].sort_by(true){|e| [e % 2, e]}  # => [3, 1, 2, 0]
```

In the last two examples above, the single argument false or true should describe ascending or descending sort of the array [e % 2, e] as a whole.

#2 - 08/02/2016 11:32 AM - duerst (Martin Dürst)

On 2016/08/02 18:57, sawadatsuyoshi@gmail.com wrote:

> Issue #12648 has been reported by Tsuyoshi Sawada.
>
> Feature #12648: Enumerable#sort_by with descending option
> https://bugs.ruby-lang.org/issues/12648

I have felt the need for such an additional argument (or something
similar) quite recently. But the examples with numbers aren't very
convincing, just changing

```
array.sort_by { |e| e }
```

to

```
array.sort_by { |e| -e }
```

will do the job. But there are many cases where that's not possible,
starting with strings.

#3 - 08/02/2016 01:34 PM - nobu (Nobuyoshi Nakada)

I prefer more descriptive option, e.g., enum.sort_by(:descend) {|e| e}.

https://github.com/ruby/ruby/compare/trunk...nobu:feature/12648-sort_by-order

#4 - 08/02/2016 03:27 PM - sawa (Tsuyoshi Sawada)

Nobuyoshi Nakada wrote:

> I prefer more descriptive option, e.g., enum.sort_by(:descend) {|e| e}.
>
> https://github.com/ruby/ruby/compare/trunk...nobu:feature/12648-sort_by-order

That's good too.

#5 - 08/03/2016 03:17 AM - knu (Akinori MUSHA)

Maybe the shorter forms :asc / :desc like in SQL would sound more familiar.

#6 - 08/03/2016 05:25 PM - sawa (Tsuyoshi Sawada)

Akinori MUSHA wrote:

> Maybe the shorter forms :asc / :desc like in SQL would sound more familiar.

Actually, I also had that in mind as one way to go.

#7 - 08/05/2016 03:42 AM - MSP-Greg (Greg L)

In concept, I agree, but, although it's common to return an array from the block, any object can be returned that supports <=>.

Hence, a better solution might be adding a method like sort_keys or sort_key, where an array is returned by the block, and an array is used as the single parameter for ascending/descending info. I might suggest, rather than true / false, or :asc / :desc, use 1 for ascending and -1 for descending.

If the parameter array is shorter, remaining keys could default to ascending, if it's longer, it's truncated.

Or (and maybe better), it could just raise an error if the arrays are different length.

#8 - 08/05/2016 05:58 AM - nobu (Nobuyoshi Nakada)

Greg L wrote:

> Hence, a better solution might be adding a method like sort_keys or sort_key, where an array is returned by the block, and an array is used as the single parameter for ascending/descending info.

Could you make clear what object these sort_key/sort_keys methods belong to?
The array to be sorted?
Or the returned object (it may not be an array) from the block?

> I might suggest, rather than true / false, or :asc / :desc, use 1 for ascending and -1 for descending.
>
> If the parameter array is shorter, remaining keys could default to ascending, if it's longer, it's truncated.
>
> Or (and maybe better), it could just raise an error if the arrays are different length.

Do you mean enum.sort_by(-1) {...} for descending?

#9 - 08/05/2016 01:01 PM - MSP-Greg (Greg L)

Nobuyoshi Nakada wrote:

> Greg L wrote:
>
>> Hence, a better solution might be adding a method like sort_keys or sort_key, where an array is returned by the block, and an array is used as the single parameter for ascending/descending info.
>
> Could you make clear what object these sort_key/sort_keys methods belong to?
> The array to be sorted?
> Or the returned object (it may not be an array) from the block?

I mentioned the fact that an array is often not returned by the block, hence, my suggestion for adding a new method.

Sorry, I should have shown a signature, below would be a possibility. The example shows three sort keys, 1st and 3rd are ascending, 2nd is descending.

t = enum.sort_key([1, -1, 1]) { |x| [f(x), g(x), h(x)] }

> Do you mean enum.sort_by(-1) {...} for descending?

Yes, but in an array, as above. IOW, both the block return and the single parameter must be arrays.

#10 - 08/05/2016 01:52 PM - MSP-Greg (Greg L)

Taking a step back, we are using arrays simply because that is a object that allows sorting via multiple criteria (if criteria a is equal, test with criteria b, etc).

It is also my understanding that sort_by creates arrays of [enum_item, sort_value], and we are also using an array for sort_value.

I'm not much for writing (or reading) c, but it would seem that a new method could use whatever structures were most efficient, regardless of the fact that arrays are used for the parameter and the block return.

#11 - 08/09/2016 06:26 AM - matz (Yukihiro Matsumoto)

I think we are talking about two things at once.

First, adding reverse (or descending) option to sort_by.
I think it may be useful for some cases, but it's only slightly better than sort_by().reverse.

Second, adding secondary key sort order to sort_by.
It may be useful too for some cases, but it should be a separate method.
Do you have any name suggestion?

Matz.

#12 - 08/13/2016 06:36 PM - MSP-Greg (Greg L)

Yukihiro Matsumoto wrote:

> I think we are talking about two things at once.
>
> First, adding reverse (or descending) option to sort_by.
> I think it may be useful for some cases, but it's only slightly better than sort_by().reverse.

I would concur.

> Second, adding secondary key sort order to sort_by.

Just to clarify, I'm not sure what you mean by 'adding secondary key sort order', key word being 'secondary'. Original poster (Tsuyoshi Sawada) stated 'When there are multiple sort keys', so (hopefully) sort key limit of more than two...

> It may be useful too for some cases, but it should be a separate method.
> Do you have any name suggestion?

Maybe something like

```
sort_keys([sort order values]) { |x| [f(x), g(x), ...] }
```

where [sort order values] is an array of composed of 1 or -1 values. Most people would associate 1 with ascending and -1 with descending. I think true and false don't convey an order as well. As to 'rules' for whether the two arrays must be identical in length, or if not, what occurs, that's a lot of options.

At a minimum, I think [sort order values] should be optional, in which case all keys would sort ascending. As to length, it might be easiest to raise an error unless both are the same length.

Sorry for the delay.

#13 - 05/12/2017 07:46 AM - knu (Akinori MUSHA)

How does sort_r_by sound when we already have grep_v?

#14 - 05/12/2017 06:25 PM - stomar (Marcus Stollsteimer)

Personally, I really do not like grep_v, or generally using options from command line tools in method names.

#15 - 10/05/2017 10:00 AM - knu (Akinori MUSHA)

Something like this can be a solution for sorting with multiple keys with separate ordering directions.

```
module Comparable
  class SortableTuple < Array
    include Comparable

    attr_reader :orderings

    def initialize(orderings)
      # Adding keyword options like `allow_nil` (`:first`/`:last`) would be great.
      replace orderings.map { |key, dir|
        desc =
          case dir
          when :desc
            true
          when :asc
            false
          else
            raise ArgumentError, "direction must be either :asc or :desc: #{dir.inspect}"
          end
        [key, desc]
      }
    end

    def <=>(other)
      if other.instance_of?(self.class)
        other.each_with_index { |(b, desc), i|
          a, = self[i]
          case cmp = a <=> b
          when Integer
            return desc ? -cmp : cmp unless cmp.zero?
          else
            return cmp
          end
        }
      end
    end
  end

  def self.[](*args)
    SortableTuple.new(*args)
  end
end

require 'pp'
require 'time'

pp [
  ["banana", Date.parse("2017-10-03")],
  ["apple",  Date.parse("2017-10-03")],
  ["grape",  Date.parse("2017-10-02")],
  ["melon",  Date.parse("2017-10-02")],
  ["orange", Date.parse("2017-10-01")],
  ["cherry", Date.parse("2017-10-01")],
].sort_by { |name, date| Comparable[date => :asc, name => :desc] }
# [["apple", #<Date: 2017-10-03 ((2458030j,0s,0n),+0s,2299161j)>],
#  ["banana", #<Date: 2017-10-03 ((2458030j,0s,0n),+0s,2299161j)>],
#  ["grape", #<Date: 2017-10-02 ((2458029j,0s,0n),+0s,2299161j)>],
#  ["melon", #<Date: 2017-10-02 ((2458029j,0s,0n),+0s,2299161j)>],
#  ["cherry", #<Date: 2017-10-01 ((2458028j,0s,0n),+0s,2299161j)>],
#  ["orange", #<Date: 2017-10-01 ((2458028j,0s,0n),+0s,2299161j)>]]
```

#16 - 10/05/2017 10:24 AM - knu (Akinori MUSHA)

Another path could be to introduce the sense of "reversed object".

I don't yet have a good name for the method that wouldn't cause name clash, but here it goes.

```
module Comparable
  class ReversedObject
    include Comparable

    def initialize(object)
      @object = object
    end
    attr_reader :object

    def <=>(other)
      other.object <=> object if other.instance_of?(self.class)
    end
  end

  def reversed
    ReversedObject.new(self)
  end
end

p ["aaa", "bbb", "ccc"].sort_by(&:reversed)
# ["ccc", "bbb", "aaa"]

require 'pp'
require 'time'

pp [
  ["banana", Date.parse("2017-10-03")],
  ["apple",  Date.parse("2017-10-03")],
  ["grape",  Date.parse("2017-10-02")],
  ["melon",  Date.parse("2017-10-02")],
  ["orange", Date.parse("2017-10-01")],
  ["cherry", Date.parse("2017-10-01")],
].sort_by { |name, date| [date, name.reversed] }
# [["apple", #<Date: 2017-10-03 ((2458030j,0s,0n),+0s,2299161j)>],
#  ["banana", #<Date: 2017-10-03 ((2458030j,0s,0n),+0s,2299161j)>],
#  ["grape", #<Date: 2017-10-02 ((2458029j,0s,0n),+0s,2299161j)>],
#  ["melon", #<Date: 2017-10-02 ((2458029j,0s,0n),+0s,2299161j)>],
#  ["cherry", #<Date: 2017-10-01 ((2458028j,0s,0n),+0s,2299161j)>],
#  ["orange", #<Date: 2017-10-01 ((2458028j,0s,0n),+0s,2299161j)>]]
```

#17 - 03/23/2019 11:13 AM - mame (Yusuke Endoh)

- Related to Feature #15725: Proposal: Add Array#reverse_sort, #revert_sort!, #reverse_sort_by, and #reverse_sort_by! added