Ruby master - Feature #14777

Add Range#offset ?

05/19/2018 09:22 AM - owst (Owen Stephens)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Hi,

As mentioned in https://bugs.ruby-lang.org/issues/14473#note-17 an addition to Range that we find useful is an Range#offset(n) method that adds (or subtracts) n to the range, for example:

```
(1..10).offset(2)   # => (3..12)
(1...10).offset(1)  # => (2...11)
(1..10).offset(-10) # => (-9..0)
```

Similarly to Range#step we can support non-Numeric objects if they implement succ:

```
('a'..'e').offset(2) # => ('c'..'g')
```

Alternative names could be Range#shift (i.e. shift the elements of the Range up or down) or perhaps >>/<<:

```
(1..10).shift(2) # => (3..12)
(1..10) >> 2 # => (3..12)
(1...10) << 1 # => (0...9)
```

However, I don't think the operators are clear enough, so I prefer offset or shift.

An example pure Ruby implementation is:

```
class Range
  def offset(n)
    add_n = ->(x) do
      if x.is_a?(Numeric)
        x + n
      elsif x.respond_to?(:succ)
        n.times { x = x.succ }
        x
      else
        raise ArgumentError, "Can't offset #{x.class}"
      end
    end

    if exclude_end?
      (add_n.call(first)...add_n.call(last))
    else
      (add_n.call(first)..add_n.call(last))
    end
  end
end
```

Please let me know your thoughts, I can then look to implement this properly in C.

Regards,
Owen.

History

#1 - 05/19/2018 11:17 AM - shevegen (Robert A. Heiler)

I personally prefer the name .offset() here over .shift(), mostly because

my brain remembers Array shift. But the name, while important,
is probably not the most relevant part - more important is what
the method does and the use case.

I think the use case is understandable, e. g. given a range object,
the ruby hacker may want to shift the range easily, up or down,
without necessarily wanting to create a new range object (even though
that is trivial, too). (Actually, I used the word shift just there ...
so perhaps it is not such a bad name after all ... so perhaps it
should be Range#shift :D).

I don't know how frequent that use case is; I don't remember having
had a need to modify range objects like that so far, but I can
understand use cases that exist for it, and after all that would
only give a bit more flexibility to Range, so I am mostly neutral
and somewhat positive on the issue here, so +1/2 up to +1 on it.

(I am using a range here to mildly upvote ... :D)

> Please let me know your thoughts, I can then look to implement
> this properly in C.

I guess it may help if someone from the core team can comment on
the issue here. I myself do not see any problem with the suggested
feature at all but I am not among the core team.

#2 - 05/19/2018 03:50 PM - lucasbuchala (Lucas Buchala)

Hello. I don't have any strong opinion about this feature, but I guess I would welcome such feature if it used standard operators. After seeing this issue, I remembered that I have tried this in the past, so I'm just sharing a snippet to give some ideas. I took this approach as a beginner. Hopefully someone more experienced will have a more idiomatic solution:

```
class Range
  def _build(sym, *args)
    b = self.begin.__send__(sym, *args)
    e = self.end.__send__(sym, *args)
    self.class.new(b, e, exclude_end?)
  end

  def +(n); _build(:+, n) end
  def -(n); _build(:-, n) end
  def *(n); _build(:*, n) end
  def /(n); _build(:/, n) end
end

p (0..4) + 5  #=> 5..9
p (5..9) - 5  #=> 0..4
p (1..4) * 2  #=> 2..8
p (2..8) / 2  #=> 1..4

p (0...4) + 5  #=> 5...9
p (5...9) - 5  #=> 0...4
p (1...4) * 2  #=> 2...8
p (2...8) / 2  #=> 1...4
```

As a comparision, Perl 6 ranges implement the same behavior:

```
say (0..4) + 5;   #=> 5..9
say (5..9) - 5;   #=> 0..4
say (1..4) * 2;   #=> 2..8
say (2..8) / 2;   #=> 1.0..4.0

say (0..^4) + 5;  #=> 5..^9
say (5..^9) - 5;  #=> ^4
say (1..^4) * 2;  #=> 2..^8
say (2..^8) / 2;  #=> 1.0..^4.0
```

Also, in a superficial search, I found a similar already proposed feature: https://bugs.ruby-lang.org/issues/7580