Ruby master - Feature #16818

Rename `Range#%` to `Range#/`

04/26/2020 02:15 AM - sawa (Tsuyoshi Sawada)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Range#% was introduced as an alias of Range#step by 14697, but it is counter-intuitive and confusing.

Iteration in the following:

```
((5..14) % 3).each{|i| p i}
#>> 5
#>> 8
#>> 11
#>> 14
```

is not based on x % y in any sense. In fact, actually applying % 3 to the selected elements returns a unique value 2, and it is not obvious how this is related to the iteration.

```
[5, 8, 11, 14].map{|i| i % 3}
# => [2, 2, 2, 2]
```

Rather, the concept seems to be based on /. Applying / 3 to the relevant elements returns a sequence 1, 2, 3, 4.

```
[5, 8, 11, 14].map{|i| i / 3}
# => [1, 2, 3, 4]
```

Hence, (5..14).step(3) can be interpreted like this: Iterate over the equivalence class (quotient set) of range 5..14 yielded by / 3.

Notice that the number of elements in [5, 8, 11, 14] is 4, which is (14 - 5 + 1) / 3.0).ceil, but is not related to %.

So I propose that the alias of Range#step should be Range#/, and Range#% should be deprecated as soon as possible before its use accumulates:

```
((5..14) / 3).each{|i| p i}
#>> 5
#>> 8
#>> 11
#>> 14
```

P.S.

And if Range#% were to be introduced at all, I would rather expect it to behave like the following:

```
((5..14) % 3).each{|i| p i}
#>> 5
#>> 6
#>> 7
```

which is why I claimed above that the current Range#% is confusing.

History

#1 - 04/26/2020 02:19 AM - sawa (Tsuyoshi Sawada)

- Description updated

#2 - 04/26/2020 10:14 AM - Eregon (Benoit Daloze)

Just my opinion, but I find % a lot more intuitive, and would find / very confusing in this context.

One interpretation of % here is by or "so that (element % n) is always the same, starting with the Range#begin value"

#3 - 04/26/2020 04:54 PM - shevegen (Robert A. Heiler)

I somewhat agree with sawa's comment that % on Range may confuse some ruby folks; I think % is more typically the modulo operator? I can not say how strong this confusion may be, perhaps small, perhaps not, but I concur with his original comment in regards to %.

However had, I also happily admit that I really have no particularly strong opinion either way. I am also indifferent (or, more accurately, a bit clueless) about /. Since it is quite unlikely for me to use % or / on Range anyway (I tend to stick to oldschool matz-ruby whenever possible ;)), I let those folks who make use of the range operation, or those who proposed it, to comment on that either way since I am not really attached to that operation to begin with.

It should be pointed out that this has follow-up effects - see zverok's example of extending it towards Arithmetic* (and there I also don't have a big opinion either way, just wanted to point out it that whatever way is chosen, it may be good to decide on it before ruby 3.0 release I think).

#4 - 04/26/2020 06:28 PM - zverok (Victor Shepelev)

I'd say that (5..14) / 3 reads definitely like "split the range into 3 parts" (expecting, IDK, 3 sub-ranges or jumps over (14-5) / 3 spans).

(5..14) % 3 at least reads (for me) like "range 5—14 <something> 3", bearing no immediate association (as we use % for reminders, and for formatting, and for array literals), and can be just memoized.

That being said, I am not sure the syntax has some significant usage currently, I am not sure whether it is due to the syntax itself, or due to the fact no Ruby objects support it (which I am trying to solve in #16812), or that majority of Ruby usage in the wild doesn't require slicing-with-step at all.

#5 - 04/26/2020 06:51 PM - sawa (Tsuyoshi Sawada)

zverok (Victor Shepelev) wrote in #note-4:

> I'd say that (5..14) / 3 reads definitely like "split the range into 3 parts" (expecting, IDK, 3 sub-ranges or jumps over (14-5) / 3 spans).

x / y reads "divide x by y", not "divide x into y parts". Following this, (5..14) / 3 divides 5..14 by 3, yielding the quotient sets [5, 6, 7], [8, 9, 10], [11, 12, 13], and [14]. Iterating over the those quotient sets, taking the representative element from each set gives, 5, 8, 11, 14.

If you don't know what a quotient set is, please take a look at the link I have provided.

Here are connections to some existing Ruby constructions:

```
(5..14).group_by.with_index{|_, i| i / 3}
# => {0=>[5, 6, 7], 1=>[8, 9, 10], 2=>[11, 12, 13], 3=>[14]}
# Represented by 5, 8, 11, 14

(5..14).each_slice(3).to_a
# => [[5, 6, 7], [8, 9, 10], [11, 12, 13], [14]]
# Represented by 5, 8, 11, 14
```

On the other hand, % gives this:

```
(5..14).group_by.with_index{|_, i| i % 3}
# => {0=>[5, 8, 11, 14], 1=>[6, 9, 12], 2=>[7, 10, 13]}
# Represented by 5, 6, 7
```

#6 - 04/26/2020 07:44 PM - zverok (Victor Shepelev)

> x / y reads "divide x by y", not "divide x into y parts".

Makes sense.

It is language difference probably: in my native Ukrainian (and Russian I was taught in school) it reads as (roughly translating) "divide into y"; therefore it is easy to be taught in schools on examples like "you have 9 apples, divide them into three piles".

#7 - 04/26/2020 11:43 PM - duerst (Martin Dürst)

Eregon (Benoit Daloze) wrote in #note-2:

> Just my opinion, but I find % a lot more intuitive, and would find / very confusing in this context.
>
> One interpretation of % here is by or "so that (element % n) is always the same, starting with the Range#begin value"

I agree. Of course, / and % are related, so it's no surprise that there are a lot of connections between these two operators. Also, to some people, % may feel unnatural at first, but once you understand the explanation that Eregon has provided here, it's difficult to forget again.

#8 - 04/27/2020 11:43 PM - inopinatus (Joshua GOODALL)

> x / y reads "divide x by y", not "divide x into y parts"

I'm not sure I understand the difference, but nevertheless I agree with Eragon's intuition.

To me, a range is not a matrix, not a sequence either. I think of ranges as intervals, so their division seems intuitively identical to cutting a line into parts.

I imagine an interval where the objects at each end are complex numbers. This immediately describes a line on the Euclidean plane.

Division is therefore cutting that line into parts, and the modulo operator the inverse i.e. the act of traversing that interval in fixed steps.

#9 - 04/28/2020 12:31 AM - sawa (Tsuyoshi Sawada)

inopinatus (Joshua GOODALL) wrote in #note-8:

> their division seems intuitively identical to cutting a line into parts. [...] Division is therefore cutting that line into parts

That is exactly what I am saying (although I am not sure why you need to refer to complex numbers; and actually, a path is a more complex notion in complex space).

> the modulo operator the inverse i.e. the act of traversing that interval in fixed steps.

I do not understand this statement of yours. The modulo operator divides a sequence into parts as well. Regarding this point, there is no difference between the two operations. The difference is that division divides it into consecutive parts whereas the modulo operator divides it evenly during the process in a manner as you would deal the cards in a card game.

And this corresponds to the difference between:

> "divide x by y", [and] "divide x into y parts"

#10 - 04/28/2020 03:33 AM - inopinatus (Joshua GOODALL)

I visualise a bag of grain. If I'm asked to divide it by three, I will make three piles of grain. If I'm asked to modulate it by three, I will make many piles, each of three grains.

> why you need to refer to complex numbers

This is to illustrate by contradiction that intervals are not sequences, arrays, or matrices in the general case, but measurable subsets of a number set.

> The modulo operator divides a sequence into parts as well

I don't agree that it divides a sequence (since ranges are not sequences), but that's not a crucial difference. For me the key is this:

Both take a interval defined as a subset on a set, and make subsets of it in turn.

In modulus, we specify the measure required of each resulting subset,

In division, we specify the number of subsets required.

You mentioned equivalence classes, and I agree they are relevant, but as a result my expectation is that the equivalence required of the modulo operator is that of the modular arithmetic it comes from i.e. the congruence relation, thus the modulus operator generates the numbers in the range that satisfy the congruence relation for given n (which is how things are now).

It might follow that I'd expect the division operator to produce an enumerator yielding N equal-length range objects spanning the range, e.g.

```
Array (5...14)/3
#=> [5...8, 8...11, 11...14]
```

however the implementation of this for all combinations may be more effort than it is worth.

#11 - 05/07/2020 08:49 AM - mrkn (Kenta Murata)

I borrowed the idea of Range#% from numo-narray.

When I first looked Range#% in numo-narray, I felt it is natural.
I refer one expression from the above example sawa showed:

```
(5..14).group_by.with_index{|_, i| i % 3}
# => {0=>[5, 8, 11, 14], 1=>[6, 9, 12], 2=>[7, 10, 13]}
```

You can see the 0th array of the resulting hash above equals to ((5..14)%3).to_a:

```
((5..14)%3).to_a == (5..14).group_by.with_index{|_, i| i % 3}[0]
# => true
```