Ruby - Feature #17830

Add Integer#previous and Integer#prev

04/26/2021 05:27 PM - rafasoares (Rafael Soares)

Status:	Assigned	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

I think Integer#pred is great as the inverse of #succ, but it reads a bit weird as the inverse of #next, which might be preferable for those going for a more "reads like English" approach.

On that note, #previous reads better, but it's also twice as long as #pred (or even #next). Which is why I've also added the shorthand #prev

Since Ruby strives for readability, I always thought it was weird that the team omitted this improvement.

Also, I thought about writing a gem for this, but:

1. Do we really want to add another gem for such a simple change to every project?
2. Monkey-patching gems feel dirty.

Finally, I want to mention that I tried looking for previous discussions on this topic, as it seems likely someone would've brought this up at some point, but was unsuccessful. Probably due to the massive amount of baggage in the core issue tracker and mailing lists, I could've missed something among the noise.

I've created a fork on GitHub (https://github.com/rafasoares/ruby/commit/05119848b1f480db2e809f964528799030cc7ebb) in order to open a PR, but decided to open this ticket as well, after reading the contributing guide more carefully.

History

#1 - 04/26/2021 05:41 PM - marcandre (Marc-Andre Lafortune)

I'm against this proposal.

In general, I don't favor aliases as they usually increase cognitive load, make code less uniform and add very little.

More importantly in this case, Integer#pred is not a frequently used method. Actually, I can not recall using it, or seeing it used ever. A method that isn't very useful should not have 3 different names.

#2 - 04/26/2021 05:56 PM - rafasoares (Rafael Soares)

marcandre (Marc-Andre Lafortune) wrote in #note-1:

> In general, I don't favor aliases as they usually increase cognitive load, make code less uniform and add very little.

In general, I agree. But in this case, we already have Integer#succ and Integer#next. It's the only reason why I think this is a reasonable addition. We also have several different convenience aliases in the language, so it seems arbitrary to me that at some point there's a line in the sand saying "we don't add aliases anymore, no matter how readable they would make the code". Also, since renaming Integer#pred is out of the question, as it would unnecessarily break older code, aliases seem like a decent alternative.

> More importantly in this case, Integer#pred is not a frequently used method. Actually, I can not recall using it, or seeing it used ever. A method that isn't very useful should not have 3 different names.

I use it from time to time, I think it reads better than number - 1, especially when you need to follow it with a method call (e.g. (number - 1).times { ... }). And, honestly, I'd use it more if it didn't read so weird. I think number.pred has a bigger cognitive load than number.prev or number.previous, since I need to remember what the hell does pred stand for in the first place. Again, compare it with number.next vs number.succ.

#3 - 04/26/2021 06:06 PM - austin (Austin Ziegler)

To be fair, the lack of a useful name for a method could also be a reason it isn't used. I don't think that is the case here; I don't think that I've ever used Integer#succ, since it is *mostly* only useful in cases where count += 1 is shorter and more expressive than count += count.succ.

-a

On Mon, Apr 26, 2021 at 1:41 PM marcandre-ruby-core@marc-andre.ca wrote:

> Issue #17830 has been updated by marcandre (Marc-Andre Lafortune).
>
> I'm against this proposal.
>
> In general, I don't favor aliases as they usually increase cognitive load, make code less uniform and add very little.
>
> More importantly in this case, Integer#pred is not a frequently used method. Actually, I can not recall using it, or seeing it used ever. A method that isn't very useful should not have 3 different names.
>
> ---
>
> Feature #17830: Add Integer#previous and Integer#prev
> https://bugs.ruby-lang.org/issues/17830#change-91688
>
> - Author: rafasoares (Rafael Soares)
> - Status: Open
> - Priority: Normal
>
> ---
>
> I think Integer#pred is great as the inverse of #succ, but it reads a bit weird as the inverse of #next, which might be preferable for those going for a more "reads like English" approach.
>
> On that note, #previous reads better, but it's also twice as long as #pred (or even #next). Which is why I've also added the shorthand #prev
>
> Since Ruby strives for readability, I always thought it was weird that the team omitted this improvement.
>
> Also, I thought about writing a gem for this, but:
>
> 1. Do we really want to add another gem for such a simple change to every project?
> 2. Monkey-patching gems feel dirty.
>
> Finally, I want to mention that I tried looking for previous discussions on this topic, as it seems likely someone would've brought this up at some point, but was unsuccessful. Probably due to the massive amount of baggage in the core issue tracker and mailing lists, I could've missed something among the noise.
>
> I've created a fork on GitHub (https://github.com/rafasoares/ruby/commit/05119848b1f480db2e809f964528799030cc7ebb) in order to open a PR, but decided to open this ticket as well, after reading the contributing guide more carefully.
>
> --
> https://bugs.ruby-lang.org/

#4 - 04/27/2021 06:39 AM - sawa (Tsuyoshi Sawada)

> I think Integer#pred is great as the inverse of #succ, but it reads a bit weird as the inverse of #next, which might be preferable for those going for a more "reads like English" approach.

I do not understand this logic. If "predecessor" and "successor" sound good as a pair, why does that go against reads-like-English approach?

Your point should rather be that "next" lacks its counterpart. That would be more understandable. Against such claim, I have the following opinions.

1. If your aim is to retain symmetry in the terminology system by supplementing a missing counterpart, then there may be a room for claiming for "prev", but not "previous", as we already have "succ" and "pred" in abbreviated form instead of "successor" and "predecessor". Introducing "previous" would break symmetry.
2. The use of terminology "succ" or "successor" is probably following the tradition of set theoretic definition of natural numbers, and hence more legitimate than "next". Just concentrate on "succ", and forget about "next". Then, you already have the counterpart "pred".

On top of that, my understanding is that Ruby encourages the use of internal iterators, which makes the use of successor and predecessor less frequent in the first place. In most cases, it is preferable to avoid using them, or replace the relevant code with with_index.

#5 - 04/27/2021 04:13 PM - rafasoares (Rafael Soares)

sawa (Tsuyoshi Sawada) wrote in #note-4:

> > I think Integer#pred is great as the inverse of #succ, but it reads a bit weird as the inverse of #next, which might be preferable for those going for a more "reads like English" approach.

> I do not understand this logic. If "predecessor" and "successor" sound good as a pair, why does that go against reads-like-English approach?

Because pred isn't an actual word. Any abbreviation adds cognitive load, since you have to think about what it means -- if we're going to get really technical about it.

> Your point should rather be that "next" lacks its counterpart. That would be more understandable. Against such claim, I have the following opinions.

That **is** my point.

> 1. If your aim is to retain symmetry in the terminology system by supplementing a missing counterpart, then there may be a room for claiming for "prev", but not "previous", as we already have "succ" and "pred" in abbreviated form instead of "successor" and "predecessor". Introducing "previous" would break symmetry.

I agree to an extend. previous reads better and would be more easily guessable for newcomers. I personally prefer prev for shortness and symmetry, but I'm not agains the more "verbose", yet clearer, alternative - for completeness.

> 1. The use of terminology "succ" or "successor" is probably following the tradition of set theoretic definition of natural numbers, and hence more legitimate than "next". Just concentrate on "succ", and forget about "next". Then, you already have the counterpart "pred".

I don't know about you, but "the tradition of set theoretic definition of natural numbers" is not something I keep in mind when developing web applications for my day job...

> On top of that, my understanding is that Ruby encourages the use of internal iterators, which makes the use of successor and predecessor less frequent in the first place. In most cases, it is preferable to avoid using them, or replace the relevant code with with_index.

I'm not talking about iteration, though. Only when you need, or just want, a more readable alternative to (my_number - 1).

Just to reiterate, the main reason why I think this is important and makes sense is because of Ruby's ideals (emphasis mine):

- (...) a focus on *simplicity and productivity* .
- It has an elegant syntax that is *natural to read* and easy to write.
- He [matz] has often said that he is “trying to make Ruby *natural, not simple*”

From https://www.ruby-lang.org and https://www.ruby-lang.org/en/about/

If we were talking about a language with more focus on a smaller API rather than readability, I would agree with all the opposing arguments presented so far.

#6 - 04/27/2021 06:13 PM - sawa (Tsuyoshi Sawada)

rafasoares (Rafael Soares) wrote in #note-5:

> He [matz] has often said that he is “trying to make Ruby *natural, not simple*”
> ...
> From https://www.ruby-lang.org and https://www.ruby-lang.org/en/about/

Not the main issue, but I need to point this out. Assuming that you are referring to this passage from the page you linked:

> Ruby is simple in appearance, but is very complex inside, just like our human body.

I think you completely misunderstood what is written. You are claiming almost the opposite of what Matz said. He said that Ruby is simple (for users). And most likely that is intentional. However, in order to achieve that, he had to (perhaps unwillingly) make its implementation complex.

Also, the other page you linked says:

> A dynamic, open source programming language with a focus on simplicity and productivity.

#7 - 04/27/2021 06:30 PM - rafasoares (Rafael Soares)

sawa (Tsuyoshi Sawada) wrote in #note-6:

> rafasoares (Rafael Soares) wrote in #note-5:
>
> > He [matz] has often said that he is “trying to make Ruby *natural, not simple*”
> > ...

> > From https://www.ruby-lang.org and https://www.ruby-lang.org/en/about/
>
> Not the main issue, but I need to point this out. Assuming that you are referring to this passage from the page you linked:
>
> > Ruby is simple in appearance, but is very complex inside, just like our human body.

No, that's another quote. I'm referring to the one in the paragraph above. Complete excerpt here:

> He has often said that he is “trying to make Ruby natural, not simple,” in a way that mirrors life.
>
> Building on this, he adds:
>
> > Ruby is simple in appearance, but is very complex inside, just like our human body[1].

Note how it says "building on this, he adds" and then there's the quote you referenced. So, to my understanding, the two sentences complement each other - but not exactly the same. To me, the charm of Ruby is precisely how natural it is to read and write. I'm only trying to improve this a bit. :)

> I think you completely misunderstood what is written. You are claiming almost the opposite of what Matz said. He said that Ruby is simple (for users). And most likely that is intentional. However, in order to achieve that, he had to (perhaps unwillingly) make its implementation complex.

How so? My goal here is precisely to make it simpler for users, even if it's a tiny bit.

> Also, the other page you linked says:
>
> > A dynamic, open source programming language with a focus on simplicity and productivity.

Yes. I omitted the "dynamic, open source programming language" part, as I didn't think it was relevant to the discussion.

#8 - 04/27/2021 11:35 PM - duerst (Martin Dürst)

- Assignee set to matz (Yukihiro Matsumoto)

As Matz gets cited more and more, I think it's best to let him decide.

#9 - 04/03/2024 03:50 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned