Ruby master - Feature #6727

Add Array#rest (with implementation)

07/13/2012 01:25 PM - duckinator (Marie Markwell)

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

Description

=begin
I run into many instances where I end up using (({arr[1..-1]})), so I decided to add (({arr.rest})) to make that a bit less hideous.

Branch on github: ((URL:https://github.com/duckinator/ruby/compare/feature/array_rest))

Patch: ((URL:https://github.com/duckinator/ruby/compare/feature/array_rest.patch))

Diff: ((URL:https://github.com/duckinator/ruby/compare/feature/array_rest.diff))
=end

History

#1 - 07/13/2012 01:55 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Open to Rejected

There are other choices besides rest = arr[1..-1]:

```
rest = arr.drop(1)
_, *rest = arr
```

See also http://bugs.ruby-lang.org/projects/ruby/wiki/HowToRequestFeatures

#2 - 07/13/2012 02:22 PM - duckinator (Marie Markwell)

> rest = arr.drop(1)
> _, *rest = arr

Those two methods you mentioned "work," but the first isn't very clear on its intent, and the second cannot be used as a statement (which is where I have personally seen [1..-1] used the most).

> See also http://bugs.ruby-lang.org/projects/ruby/wiki/HowToRequestFeatures

1. Insure it's a meaningful improvement
   - Yes, this improvement is discussed in multiple Google search results and has received positive feedback when I mentioned it elsewhere.
   - There are already ways to achieve similar results, but they don't convey their purpose well.
   - Are their cases where it would be useful? I have seen, and used arr[1..-1] countless times. Also see above statement regarding positive feedback.
2. Think about it
   - What's a good name? Array#rest
   - What exact arguments does it accept? None.
   - What does it return? If the Array is empty, nil, otherwise, a new Array.
   - Any risk of incompatibility? No.
3. Write it up
   - The title was apparently good, as it was not ignored.
   - The current situation is improved because Array#rest is clear about your intent,
   - I made a concise but complete proposal.
   - I did not address the objection of .drop(1), because I had forgotten about it. I do hope _, *rest = arr was purely for demonstrative purposes.
4. Feature request already opened, with patch.

Is there any reason for this being rejected besides the existence of arr.drop(1) and your other method which (in my opinion) should not be necessary unless you're doing first, *rest = arr?

#3 - 07/13/2012 02:33 PM - duckinator (Marie Markwell)

And I just double-checked if those behave the same, and they do *not*:

Incorrect:

```
[].drop(1)
=> []

first, *rest = []
=> []
first
=> nil
rest
=> nil
```

Correct:

```
[][1..-1]
=> nil

[].rest
=> nil
```

So it would appear there are not other choices besides rest = arr[1..-1], at least not that fit on a single line.

#4 - 07/13/2012 02:46 PM - programble (Curtis McEnroe)

+1, This is a much cleaner way to achieve the exact same as ary[1..-1]

#5 - 07/13/2012 03:01 PM - tsion (Scott Olson)

I agree, I see and use ary[1..-1] quite a lot, and ary.rest would convey the meaning a lot better.

And it isn't just the ugliness of the syntax of ary[1..-1] that makes it undesirable, there's also the fact that it is zero-based from the front end and one-based from the back end (I know this can't really be helped because 0 == -0, but it does make Array#rest more desirable).

On top of that, I think #first and #rest make a nice pair, like head and tail from Haskell or any language with cons lists.

#6 - 07/13/2012 08:30 PM - Eregon (Benoit Daloze)

tsion (Scott Olson) wrote:

> On top of that, I think #first and #rest make a nice pair, like head and tail from Haskell or any language with cons lists.

An Array is not a cons list. #rest is O(n) here, compared to O(1) with cons.
So, doing ary[1..-1] or tail is something not very efficient, which you probably do not want most of the time (at least, not in a loop taking one element at a time with #first/#rest).
If it is for removing the first items, because for example, the Array is a set of lines and the first is a header, I'm fine with drop(1).

duckinator (Marie Markwell): Could you show us a real use case for Array#rest ?

#7 - 07/13/2012 10:44 PM - naruse (Yui NARUSE)

- Status changed from Rejected to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

#8 - 07/13/2012 11:31 PM - marcandre (Marc-Andre Lafortune)

So Matz will decide one day, then.

Here are further objections in the meantime.

duckinator (Nick Markwell) wrote:

```
rest = arr.drop(1)
```

_, *rest = arr

> Those two methods you mentioned "work," but the first isn't very clear on its intent

How is the result of dropping the first element not clear? Drop the first element and give me the rest...

> and the second cannot be used as a statement (which is where I have personally seen [1..-1] used the most).

Since you haven't yet given any real world example, that's possible, but I would guess that many times you will also use the first element of the array, no?

Then that's a good pattern to use. Or foo = arr.shift.

1. Insure it's a meaningful improvement
   - Yes, this improvement is discussed in multiple Google search results and has received positive feedback when I mentioned it elsewhere.

It would be interesting to see examples in actual code / gems. In the whole of Rails' code, I found exactly *one* case of array[1..-1]

1. Think about it
   - What's a good name? Array#rest

"rest" from what?

- What exact arguments does it accept? None.

why not? how about [2..-1]?

- What does it return? If the Array is empty, nil, otherwise, a new Array.
  - I did not address the objection of .drop(1), because I had forgotten about it.

Is the distinction between [].rest == nil and [1].rest == [] useful? How/when? In particular, in what kind of case would the *only* difference between arr.drop(1) and arr.rest be useful?

You would like a new method which does exactly what drop(1) does, but with less versatility (no way to do rest(2), say) and with a single difference in the case of an empty array.

- I do hope _, *rest = arr was purely for demonstrative purposes.

The '_' part was, but not the pattern. When you deal with the first part of the array, the pattern can be very useful.

#9 - 07/14/2012 12:00 AM - trans (Thomas Sawyer)

Rather then haphazard method additions in this area I still think a better approach would be a common mixin.

http://rdoc.info/github/rubyworks/facets/master/Indexable

It's like Enumerable, but for a different set of functionality. Depending only on a minimal set of methods to facilitate the rest creates a more robust and flexible system. Call it "Occam's Principle of Interfaces" (though the principle probably already has another name?)

#10 - 07/14/2012 12:34 AM - duckinator (Marie Markwell)

> duckinator (Marie Markwell): Could you show us a real use case for Array#rest ?

I'll add another update in a bit with examples.

Marc-Andre:

You raised some good objections. If you don't think I answer all of them let me know, because I want this added, but I want it added properly.

> How is the result of dropping the first element not clear? Drop the first element and give me the rest...

Sorry, that was bad phrasing on my part. I meant it's not as clear as it *could be.*

> It would be interesting to see examples in actual code / gems. In the whole of Rails' code, I found exactly *one* case of array[1..-1]

I'll look around for some and add another update in a bit.

> "rest" from what?

The "rest" of the Array, since it goes with "first." An alternative name would be "tail," but it's usually head/tail so may lead people to expect "head" to work. That's a minor issue, but IMO worth noting. It's also serving the same purpose as "cdr," but this isn't lisp, and that name's about as clear as mud to anyone who's not used a lisp dialect before. Do you have a name you feel would fit better than "rest"?

> Is the distinction between [].rest == nil and [1].rest == [] useful? How/when? In particular, in what kind of case would the *only* difference between arr.drop(1) and arr.rest be useful?

To be honest, I'm not entirely sure. That was probably an issue not worth raising. Which return value is really "expected" is rather iffy because quite a few of the languages I have seen that implement a similar function are languages that implement their equivalent of nil as an empty list. If nobody else (programble (Curtis McEnroe), tsion (Scott Olson)?) can come up with a need for [].rest == nil, I'd be more than willing to let it return an empty list.

> The '_' part was, but not the pattern. When you deal with the first part of the array, the pattern can be very useful.

Alright. I apologize if I came off as rude: my (rather foolish, admittedly) assumption was that you meant that as a way to *only* get rest, and that came across as a bit odd, to put it mildly.

Thomas:

> Rather then haphazard method additions in this area I still think a better approach would be a common mixin.

I did not know about Indexable, thanks for mentioning it. I agree that using a common mixin would be a far better approach. Will take a closer look at that after I have lunch.

#11 - 10/28/2012 12:14 AM - yhara (Yutaka HARA)

- Category set to core

- Target version set to 2.6

#12 - 10/28/2012 11:25 AM - duckinator (Marie Markwell)

Sorry that I forgot to check this again before.

I've done a bit of thinking and poking around since I ran across this again today.

Regarding [].rest == nil vs [].rest == []:
[] is treated as a truthy value, so there if rest returns that, then there is a significant difference when #rest is used in place of [1..-1] -- you'd have to check if the original list is empty yourself, so *if this gets added* it is of no use unless it returns nil under those circumstances.

Regarding examples of it used in practice:

Just from code I happened to already have on my system, there is 501 instances of [1..-1] in Ruby code.

https://gist.github.com/3967189

Out of what I have on my system it is, surprisingly, used mostly in relation to Ruby interpreters: 217 instances in MRI's git repo, 162 instances in Rubinius' git repo, 95 instances in JRuby's git repo.

Aside from that, on my system the usages are mostly small: 4 uses by Bundler, 11 instances by what I believe are WebKit's build scripts, 4 times by cinch, and 8 times in various other things.

Other thoughts:

Perhaps Array#rest(n=1) being equivalent to Array#[n..-1], including returning nil for [].rest, would be a better approach? That way it isn't strict to the point of being nearly useless, but is still short and to the point.

#13 - 10/28/2012 08:25 PM - Eregon (Benoit Daloze)

duckinator (Nick Markwell) wrote:

> Regarding examples of it used in practice:
>
> Just from code I happened to already have on my system, there is 501 instances of [1..-1] in Ruby code.
>
> https://gist.github.com/3967189

Thank you for searching for examples!

I guess you are aware many of these examples are using String#[1..-1].
A few examples are using MatchData#[1..-1], which should be replaced by MathData#captures.

For Bundler, only 1 example is Array#[], the 3 others are String#[].
For Cinch, 2 of 4 are Array#.
Oddly enough, 2 out of these 3 examples of Array#[] are actually removing the first line of an Exception backtrace.

For my part, I don't feel a real need for Array#rest, and I dislike #rest because it would imply the structure has an head and a tail, which Array does not. I think having a long/not-so-good-looking way to take all elements but the first is an advantage as it does not encourage this operation which is rarely optimal. As I said earlier, I think Array#drop is nicely used in this case.

#14 - 11/10/2013 05:12 AM - baweaver (Brandon Weaver)

As this seems to have been either dead-ended or otherwise, I'd like to bring it back up.

Most of the arguments I head as to why not to include a rest or tail method is that Ruby is not Lisp, or that there's a hack-around method that works the same way. The two objections I have to such reasoning are that Matz himself designed Ruby in part after Lisp, and that the point of Ruby is to be succinct and clear.

I strongly believe that array.rest or array.tail are clearer than array[1..-10] or rest = array.drop. The point is to be clear and concise, and this clearly aims to improve upon such.

As per the usefulness of such a construct, tail-recursion and functional constructs come heavily to mind.

Within the last year I would have made an argument that lambda was not needed in the language because I had not tried to use it, and I have been heavily proven wrong in that thinking. A C programmer may think closures are useless because they have never used one in production. You use the tools you are given, and in some cases become biased towards them.

That being said, we could also add car and cdr just for warm fuzzy feelings while we're at it ;)

#15 - 11/10/2013 05:53 AM - fuadksd (Fuad Saud)

I proposed it as well in #9023 in the CommonRuby list. I don't think the arguments given there are enough to justify not implementing such an useful method.
On Nov 9, 2013 6:13 PM, "baweaver (Brandon Weaver)" <
brandon_weaver@baweaver.com> wrote:

> Issue #6727 has been updated by baweaver (Brandon Weaver).
>
> As this seems to have been either dead-ended or otherwise, I'd like to bring it back up.
>
> Most of the arguments I head as to why not to include a rest or tail method is that Ruby is not Lisp, or that there's a hack-around method that works the same way. The two objections I have to such reasoning are that Matz himself designed Ruby in part after Lisp, and that the point of Ruby is to be succinct and clear.
>
> I strongly believe that array.rest or array.tail are clearer than array[1..-10] or rest = array.drop. The point is to be clear and concise, and this clearly aims to improve upon such.
>
> As per the usefulness of such a construct, tail-recursion and functional constructs come heavily to mind.
>
> Within the last year I would have made an argument that lambda was not needed in the language because I had not tried to use it, and I have been heavily proven wrong in that thinking. A C programmer may think closures are useless because they have never used one in production. You use the tools you are given, and in some cases become biased towards them.
>
> That being said, we could also add car and cdr just for warm fuzzy
>
> ## feelings while we're at it ;)
>
> Feature #6727: Add Array#rest (with implementation)
> https://bugs.ruby-lang.org/issues/6727#change-42829
>
> Author: duckinator (Nick Markwell)
> Status: Assigned
> Priority: Normal

Assignee: matz (Yukihiro Matsumoto)
Category: core
Target version: next minor

=begin
I run into many instances where I end up using (({arr[1..-1]})), so I
decided to add (({arr.rest})) to make that a bit less hideous.

Branch on github: (())

Patch: (())

Diff: (())
=end

--
http://bugs.ruby-lang.org/

#16 - 11/11/2013 12:27 AM - marcandre (Marc-Andre Lafortune)

Hi,

duckinator (Nick Markwell) wrote:

> Regarding examples of it used in practice:
>
> Just from code I happened to already have on my system, there is 501 instances of [1..-1] in Ruby code.
>
> https://gist.github.com/3967189
>
> Out of what I have on my system it is, surprisingly, used mostly in relation to Ruby interpreters: 217 instances in MRI's git repo, 162 instances in Rubinius' git repo, 95 instances in JRuby's git repo.
>
> Aside from that, on my system the usages are mostly small: 4 uses by Bundler, 11 instances by what I believe are WebKit's build scripts, 4 times by cinch, and 8 times in various other things.

I believe that the vast majority of these examples are for strings, not arrays.

#17 - 11/11/2013 01:47 AM - Hanmac (Hans Mackowiak)

i think in most cases we want something like ary[1..-1].each but wouldnt it be better if we have an each method like

each_only(1..-1) {}

that does that for us without making an new array?

#18 - 11/11/2013 04:23 PM - duerst (Martin Dürst)

On 2013/11/11 0:27, marcandre (Marc-Andre Lafortune) wrote:

> duckinator (Nick Markwell) wrote:
>
> I believe that the vast majority of these examples are for strings, not arrays.

That might suggest adding #rest to String, too.

Regards, Martin.

#19 - 11/11/2013 04:23 PM - duerst (Martin Dürst)

On 2013/11/11 1:47, Hanmac (Hans Mackowiak) wrote:

> i think in most cases we want something like ary[1..-1].each but wouldnt it be better if we have an each method like
>
> each_only(1..-1) {}

#each currently doesn't have arguments. So this could even be

each(1..-1) {}

Regards, Martin.

> that does that for us without making an new array?

#20 - 11/11/2013 06:18 PM - alexeymuranov (Alexey Muranov)

duerst (Martin Dürst) wrote:

> On 2013/11/11 1:47, Hanmac (Hans Mackowiak) wrote:
>
> > i think in most cases we want something like ary[1..-1].each but wouldnt it be better if we have an each method like
> >
> > each_only(1..-1) {}
>
> #each currently doesn't have arguments. So this could even be
>
> each(1..-1) {}

If #each is to take arguments, i would suggest

[1, 2, 'a', :b, 3].each(Integer).map{|x| 2*x } # => 2, 4, 6.each(Integer).to_a # => [1, 2, 3, 4, 5]

Edited

#21 - 11/11/2013 06:27 PM - Hanmac (Hans Mackowiak)

alexeymuranov (Alexey Muranov) wrote:

> If #each is to take arguments, would suggest
>
> [1, 2, 'a', :b, 3].each(Integer).map{|x| 2*x } # => [2, 4, 6]

ok that would be like #grep, but i want something that uses index, not the object itself

#22 - 11/12/2013 02:29 AM - fuadksd (Fuad Saud)

Reading back the comments, I realize Indexable may be a good idea for String/Array polymorphism.

#23 - 11/12/2013 10:23 AM - duerst (Martin Dürst)

On 2013/11/12 2:29, fuadksd (Fuad Saud) wrote:

> Issue #6727 has been updated by fuadksd (Fuad Saud).
>
> Reading back the comments, I realize Indexable may be a good idea for String/Array polymorphism.

The idea is already there. If you check, you'll notice that whenever
something makes sense both for an Array and a String (e.g. #length),
it's the same method with the same arguments. There is no explicit
Indexable module or superclass because for efficiency, implementations
are separate (and sharing implementations would be the main/only reason
for using a module or a superclass in Ruby).

But there is also some aspect in which the two classes don't match. For
Array, #each iterates over elements. But for String, #each doesn't
exist. For the discussion in this issue, String#each would have to
iterate over characters, but in Ruby 1.8, it iterated over lines, and
that's why it was removed altogether from Ruby 1.9, and we have
#each_char, #each_byte, #each_line now.

Regards, Martin.

#24 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#25 - 08/14/2019 08:51 AM - mame (Yusuke Endoh)

- Status changed from Assigned to Feedback

Now we have an endless range which allows us to write arr[1..]. It is much less hideous than arr[1..-1], IMO. Do you still want Array#rest?