Ruby master - Feature #16120

Omitted block argument if block starts with dot-method call

08/23/2019 03:22 PM - Dan0042 (Daniel DeLorme)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

How about considering this syntax for implicit block parameter:

```
[10, 20, 30].map{ .to_s(16) }  #=> ["a", "14", "1e"]
```

Infinite thanks to maedi (Maedi Prichard) for the idea

This proposal is related to #4475, #8987, #9076, #10318, #10394, #10829, #12115, #15302, #15483, #15723, #15799, #15897, #16113 (and probably many others) which I feel are all trying to solve the same "problem". So I strongly believe all these feature requests should to be considered together in order to make a decision. And then closed together.

This "problem" can be more-or-less stated thus:

- There is a very common pattern in ruby: posts.map{ |post| post.author.name }
- In that line, the three 3 "post" in close proximity feel redundant and not DRY.
- To reduce the verbosity, people tend to use a meaningless one-character variable in the block
- But even so posts.map{ |p| p.author.name } still feels redundant.
- This "problem" is felt by many in the ruby community, and is the reason people often prefer posts.map(&:author)
- But that only works for one method with no arguments.
- This results in many requests for a block shorthand than can do more.

I realize that many people feel this is not a problem at all and keep saying "just use regular block syntax". But the repeated requests over the years, as well as the widespread usage of (&:to_s), definitely indicate this is a wish/need for a lot of people.

Rather than adding to #15723 or #15897, I chose to make this a separate proposal because, unlike it or @ implicit variables, it allows to simplify **only** { |x| x.foo }, not { |x| foo(x) }. This is on purpose and, in my opinion, a desirable limitation.

The advantages are (all in my opinion, of course)

- Extremely readable: posts.map{ .author.name }
  - Possibly even more than with an explicit variable.
- Of all proposals this handles the most important use-case with the most elegant syntax.
  - It's better to have a beautiful shorthand for 90% of cases than a non-beautiful shorthand for 100% of cases.
  - A shorthand notation is less needed for { |x| foo(x) } since the two x variables are further apart and don't feel so redundant.
- No ascii soup
- No potential incompatibility like _ or it or item
- Very simple to implement; there's just an implicit |var| var at the beginning of the block.

- In a way it's similar to chaining methods on multiple lines:

```
posts.map{ |post| post
  .author.name
}
```

It may be interesting to consider that the various proposals are not *necessarily* mutually exclusive. You *could* have [1,2,3].map{ .itself + @ + @1 }. Theoretically.

I feel like I've wanted something like this for most of the 16 years I've been coding ruby. Like... **this** is what I wanted that (&:to_s) could only deliver half-way. I predict that if this syntax is accepted, most people using (&:to_s) will switch to this.

History

#1 - 08/23/2019 03:36 PM - osyo (manga osyo)

hi.
Its soo good idea.
However, I think it is difficult to parse in the following cases.

```
[10, 20, 30].map{
  # 42.to_s(16)
  # or
  # pp 42
  # argument1.to_s(16)
  pp 42
  .to_s(16)
}
```

#2 - 08/23/2019 11:42 PM - shan (Shannon Skipper)

A bit of an aside, but it's often just as fast to do two iterations with small collections, since the shorthand parses faster.

```
posts.map(&:author).map(&:name)
```

I agree with osyo that it seems this proposal collides with existing parser behavior. It would introduce incompatibility.

#3 - 08/24/2019 12:59 AM - Dan0042 (Daniel DeLorme)

I think there's a misunderstanding because this proposal doesn't collide with existing parser behavior. [].each{ .method } is currently a SyntaxError.

osyo (manga osyo) wrote:

> However, I think it is difficult to parse in the following cases.

It parses just like this:

```
[10, 20, 30].map{ |v| v
  pp 42
  .to_s(16)
}
```

In other words the block argument is not used, and .to_s(16) applies to 42, just like regular method chaining.

#4 - 08/24/2019 01:07 AM - Dan0042 (Daniel DeLorme)

- Description updated

#5 - 08/24/2019 01:57 AM - mame (Yusuke Endoh)

Hi,

> So I strongly believe all these feature requests should to be considered together in order to make a decision.

Agreed. And, #15723 (a numbered parameter) is only one proposal that is all-purpose, though I don't like it so much.

> A shorthand notation is less needed for { |x| foo(x) } since the two x variables are further apart and don't feel so redundant.

I personally agree. I don't think that the variable name is redundant. But people seem to think so. Actually, a shorthand for Object#method is planned for 2.7 (#12125), and I hear many people want to use it as: map(&JSON.:parse). The syntax you propose cannot absorb this style.

#6 - 08/26/2019 07:34 PM - Dan0042 (Daniel DeLorme)

The syntax I propose is definitely not *meant* to absorb all styles. I think any attempt to be everything to everyone is doomed to failure. I do not believe this is a race where only one of the various proposals can win; considering the various proposals together means finding the right balance, not finding a single all-purpose solution.

In fact I find that map{ .to_s(16) } and map(&JSON.:parse) are very complementary...

- map{ .to_s(16) } is shorthand for map{ |x| x.to_s(16) }; each element is the receiver of a message; this is OO style. I would use that a lot.
- map(&JSON.:parse) is shorthand for map{ |x| JSON.parse(x) }; each element is the argument of a function; this is functional style, for people who want first-class functions in ruby. I would likely never use that. But I don't mind others who want to use that style.

#7 - 08/26/2019 11:49 PM - mame (Yusuke Endoh)

I believe that map(&JSON.:parse) must be considered together because it is strongly related to the motivation of your proposal. Note that map(&JSON.:parse) is incomplete. People will next want to omit a parameter of map {|x| JSON.parse(x, symbolize_names: true) }. The game is not

ended, and the next proposal will definitely come, like map(&JSON.:parse.(_, symbolize_names: true)) or what not. Only the numbered parameter can end the game.

#8 - 08/27/2019 12:18 AM - nobu (Nobuyoshi Nakada)

https://github.com/nobu/ruby/tree/feature/implicit-block-param

#9 - 08/27/2019 01:39 AM - Dan0042 (Daniel DeLorme)

- Description updated

mame (Yusuke Endoh), The motivation of *this* proposal is related to the **side-by-side** proximity/repetition of x in {|x|x.foo}. Other proposals may be different. I can only guess at their true motivations. It just *seems* to me that the people asking for that kind of shorthand really intend to use it for {|x|x.foo} and just throw in {|x|foo(x)} because why not. But @ is similar to $_ in that it's only useful for debugging or throwaway code. {.foo} can actually be used in production code and make it clearer.

The motivation for (&JSON.:parse)... honestly it seems like it's an entirely different beast. I don't think it's only about shortening the block. I have the feeling it's really about functional programming, and that {JSON.parse(@)} would not satistify the "requirement" for first-class functions. This one seems to be more about function composition, currying and whatnot, and less about avoiding verbosity.

You make a painfully good point about how unendingly persistent these proposals are. But if the numbered parameter could really end the game, I'm quite sure there would not be so much opposition to it in #15723.

#10 - 08/27/2019 02:16 AM - jeremyevans0 (Jeremy Evans)

Dan0042 (Daniel DeLorme) wrote:

> mame (Yusuke Endoh), The motivation of *this* proposal is related to the **side-by-side** proximity/repetition of x in {|x|x.foo}. Other proposals may be different. I can only guess at their true motivations. It just *seems* to me that the people asking for that kind of shorthand really intend to use it for {|x|x.foo} and just throw in {|x|foo(x)} because why not. But @ is similar to $_ in that it's only useful for debugging or throwaway code. {.foo} can actually be used in production code and make it clearer.

I disagree. { foo(@) } and { @.foo } are not for debugging or throwaway code, they are natural replacements for { |x| foo(x) } and {|x| x.foo }. The @ single implicit parameter approach is just as clear and is significantly more flexible than this approach (lacking a better name, the omitted parameter approach).

That's not to say the omitted parameter approach is bad. In the cases it does handle, it does save a character compared to the implicit parameter approach. I don't think that character saving makes the code clearer than the single implicit parameter approach, though. In my opinion they are even in terms of clarity.

> The motivation for (&JSON.:parse)... honestly it seems like it's an entirely different beast. I don't think it's only about shortening the block. I have the feeling it's really about functional programming, and that {JSON.parse(@)} would not satistify the "requirement" for first-class functions. This one seems to be more about function composition, currying and whatnot, and less about avoiding verbosity.

I think the implicit parameter approach ({JSON.parse(@)}) is a simpler and more readable approach than the dot-colon approach ((&JSON.:parse)). Especially when you start function composition ({JSON.parse(JSON.generate(@))} vs (&(JSON.:parse << JSON.:generate))). Especially when you consider things like additional block arguments and keyword arguments (:symbolize_keys) passed to the methods.

> You make a painfully good point about how unendingly persistent these proposals are. But if the numbered parameter could really end the game, I'm quite sure there would not be so much opposition to it in #15723.

By "end the game", I think mame means that it is the most flexible approach, not necessarily the best approach. And it doesn't really "end the game", as it doesn't handle block or keyword arguments :).

#11 - 08/27/2019 02:43 PM - Dan0042 (Daniel DeLorme)

- Description updated

- Subject changed from Implicit block argument if block starts with dot-method call to Omitted block argument if block starts with dot-method call

nobu (Nobuyoshi Nakada), wow, thank you so much. I never imagined it would be THAT simple to implement.
O_O @_@ m(_ _)m

But I do think it would be better with (parser_numbered_param(p, 0)) in the commit here m(_ _)m

---

jeremyevans0 (Jeremy Evans) wrote:

> In the cases it does handle, it does save a character compared to the implicit parameter approach. I don't think that character saving makes the code clearer than the single implicit parameter approach, though. In my opinion they are even in terms of clarity.

I totally agree that "saving" a single character makes no difference. But all these proposals are not about reducing mere character count, they're about reducing... I don't know the right word... cognitive complexity? lexical redundancy? conceptual overhead? It's the reason why people propose {item.foo} even though it has *zero* characters less than {|x|x.foo}. It's the reason why people who use nice descriptive variable and method names can also propose {@.foo} even though it's an insignificant three character saving. It's the reason why human languages use omissions and pronouns. Allow me to make a comparison with english:

```
omitted   {.foo}      John went to the market and bought apples
implicit  {@.foo}     John went to the market and he bought apples
numbered  {@1.foo}    John went to the market and HE bought apples
explicit  {|x|x.foo}  John went to the market and John bought apples
```

There's a reason why the first form is the most natural. When people talk about a block shorthand, I really think they mean shorter in the sense of cognition, not character count (although the two are somewhat related). So rather than thinking of a 1-char saving, it's more like explicit has 2x overhead, implicit has 1x, and omitted has 0x. Yes, we're talking about a very very tiny amount of overhead, I'll grant you, but enough to have these proposals keep popping up. That's not to say the implicit parameter approach is bad, in fact I rather like it. I just happen to think the omitted approach has so much better "flow". 1x/0x = Infinity kind of thing.

> I think the implicit parameter approach ({JSON.parse(@)}) is a simpler and more readable approach than the dot-colon approach ((&JSON.:parse)).

I totally agree there again. I was trying to present the perspective of functional-style first-class-function people (which I am not). Maybe trying to argue on behalf of others is a mistake in itself.

#12 - 08/27/2019 03:13 PM - Hanmac (Hans Mackowiak)

Dan0042 (Daniel DeLorme) in your list about implicit and explicit you forgot { foo } depending on the method who gets the block, it might does an instance_eval thing where the block self is the block variable

i know that would need to change of the method, but this one might be possible too

#13 - 08/27/2019 05:25 PM - Dan0042 (Daniel DeLorme)

- Description updated

#14 - 08/27/2019 05:36 PM - Dan0042 (Daniel DeLorme)

Hanmac (Hans Mackowiak) wrote:

> Dan0042 (Daniel DeLorme) in your list about implicit and explicit you forgot { foo } depending on the method who gets the block, it might does an instance_eval thing where the block self is the block variable

There was something like that in #10394, but I think it changes the semantics of the block too much. Should {foo(bar)} really be equivalent to {|v|v.foo(v.bar)} ? I don't think so.

#15 - 09/20/2019 07:22 AM - shevegen (Robert A. Heiler)

The idea is very interesting to me, purely from a conceptual point of view alone. So from this point of view, I like the idea itself, or the thoughts behind the idea.

HOWEVER had, at the same time, I actually dislike the syntax.

My brain wants to associate the ".something" method call with some specific object, and in the proposal we would then have a situation where the method invocation is not directly attached to an object. Again, I understand the intent, but I dislike it from a syntax point of view.

> In a way it's similar to chaining methods on multiple lines:

```
posts.map{ |post| post
  .author.name
}
```

Yes, I get the intent, but it is not really similar in this sense; it is actually different because we would then be allowed to omit the reference to the object "magically".

So if I were to have a say, my personal opinion is -1 on the proposal as such, even though oddly enough I also like it at the same time (that has not happened often before where I was against something even though liking it in some ways).

Jeremy wrote:

> I totally agree that "saving" a single character makes no difference.

In the context of e. g. numbered parameters, I agree. But when it comes to Symbols versus Strings, to me the single character matters a lot. :-)

I understand that this has not been why matz added Symbols, but I like them and use them a lot, even simply as shortcut "identifier" in method colours e. g:

```
disable :colours
```

(It's actually convenient to me more generally than e. g. passing a 'string', even though symbols and strings have a different meaning/purpose.)

> You could have [1,2,3].map{ .itself + @ + @1 }

Now I happily admit that I dislike itself, but I think I dislike it even more in a .itself variant, together with @ and @1. Actually, @1 is better than @. I think @ alone is actually the worst here.

But we should not really tie a suggestion such as this one here to other not-directly related suggestions such as numbered parameter. (I think matz went for another syntax meanwhile, so suggestions may become outdated, too. :))

I agree that:

```
posts.map{ .author.name }
```

is, oddly enough, very readable. I like that part. Still I am against it; I completely understand it where we may refer to whatever is "held" inside the block, but I don't like free floating method calls in ruby code to be honest.

Last but not least about numbered paramaters; I bring this due to the brief discussion about between mame and jeremy.

I think people will always focus about different parts about numbered parameters. For me the biggest advantage is that we can quickly use it to debug longer "data structures" without having to remember the name as-is. I don't think I want it to remain in my own code though, since it is not super-elegant; on the other hand I am also fine with it being in code, in principle. But this opinion does not seem to be the one that you can read most often. People either love it, or hate it; sort of. It's strange to me. :-)

Even then I think one last thing should not be forgotten about ALL changes to ruby in the last ~3 or 4 years or so - specific changes were made that gave new options. Even though I myself dislike the syntax part of some suggestions, or my poor eye sight in regards to object.:method, we should include the possibility of new features leading to scenarios where less code (or simpler code) might be use; a good example for that is the safe navigation / lonely person staring at a dot, change.

#16 - 09/23/2019 05:05 PM - Dan0042 (Daniel DeLorme)

Since I wasn't at the developer meeting I'll post my thoughts/responses to the log here.

> matz: I prefer this style to .map(&:to_s). But I understand it is not flexible enough. Difficult to determine.

This shorthand simplifies one of the most common block idioms, no more no less (dotadiw). IMHO it's meant to be elegant rather than flexible; for flexibility one should always use the regular block syntax.

> shevegen: if I were to have a say, my personal opinion is -1 on the proposal as such, even though oddly enough I also like it at the same time

That's... an interesting paradox. I would say to you (and matz): just trust your gut feeling on this and accept that it "looks really nice" without overanalyzing :-)

mame: If this style is accepted, Symbol#to_proc is deprecated? It sounds good.
matz: Sounds like a plan.

I'm pretty sure everyone will voluntarily switch from (&:to_s) to {.to_s} but even so is there really a need to go so far as to deprecate Symbol#to_proc ?

someone: Can we write 1.times { .foo; any-statement }?
matz: No

I have no idea why someone would want to write this, but why is there a need to disallow it? The shorthand is simpler to explain and implement if it's just {.foo is equivalent to {|x|x.foo, without special exceptions or restrictions to account for.

nobu: It is difficult to implement… Currently my patch even allows 1.times { .foo(.bar) }
matz: No

Wow, I did not realize your patch allowed that! I think it's safer if it's only at the beginning of the block. I've tried writing a patch for that. Please forgive if it's too hacky. https://github.com/dan42/ruby/commit/51e5fa4d4a56c29e5f0ceb1e5544822e9215148f
Much better now: https://github.com/dan42/ruby/commit/09c49f36d65338a8a2d5f0c1f1d8e41e734eacc2 (It still crashes Ripper though)

#17 - 09/24/2019 10:08 PM - Dan0042 (Daniel DeLorme)

I ended up rebasing my commits to the latest master and then fiddling with parse.y until I got everything clean and working, including Ripper.
So this is my implementation of omitted parameters based on nobu's original patch:
https://github.com/dan42/ruby/commit/62628cb739748bcca2297c8aaec1195d7565f100

#18 - 10/16/2019 03:58 PM - Dan0042 (Daniel DeLorme)

Rebased my patch to the latest master and cleaned up the code a bit:
https://github.com/dan42/ruby/commit/11f609af003370396d0e82381b57ea5a73ff6d8a

#19 - 10/17/2019 08:42 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

After reconsideration, I came to the conclusion that we should use numbered parameters instead of this feature.
Thank you for a pretty interesting idea. It was good food for thought.

Matz.

#20 - 10/17/2019 01:27 PM - Dan0042 (Daniel DeLorme)

I am so sad :-(

I was really looking forward to using this beautiful syntax instead of numbered parameters which are so ugly it's better to use an explicit block variable.

Am I the only one who thinks the omitted parameter is by far the most DRY and readable in these examples? imho, omitted > explicit > numbered

```
posts.map{ |p| p.author.name } #explicit looks kinda ugly (for this case) so I want to simplify it
posts.map{ _1.author.name }    #numbered is even uglier
posts.map{ .author.name }      #omitted is beautiful

posts.map{ |p| format_author_name_as_html(p) } #this looks fine, I don't feel a need to simplify it
posts.map{ format_author_name_as_html(_1) }    #just as I thought, this looks worse
```

```
array3D                          array3D                        array3D
.each do |array2D|               .each do                       .each do
  array2D.each do |array|          _1.each do                     .each do
    array.each do |point|            _1.each do |point|             .each do |point|
                                    #^error
```

I beg you to reconsider, for the beauty of ruby. m(_ _)m

#21 - 10/18/2019 07:01 AM - nobu (Nobuyoshi Nakada)

Dan0042 (Daniel DeLorme) wrote:

```
posts.map{ .author.name }      #omitted is beautiful
```

Sorry, it doesn't feel beautiful to me.

```
array3D
.each do
  .each do
    .each do |point|
```

It looks confusing.

#22 - 10/18/2019 08:19 AM - decuplet (Nikita Shilnikov)

Dan0042 (Daniel DeLorme) wrote:

> Am I the only one who thinks the omitted parameter is by far the most DRY and readable in these examples?

Even if you are not, what does it change? Personally, I find this proposal interesting as an idea but I prefer _1, likely because of my familiarity with Clojure and Scala (it has a different meaning in Scala but nevertheless).

#23 - 10/18/2019 08:30 AM - Hanmac (Hans Mackowiak)

Dan0042 (Daniel DeLorme) wrote:

```
array3D
.each do
  .each do
    .each do |point|
```

my main problem with omitting parameters there is that it can't know which binding you want to use?

i am already have problems with that because i think it should have used the outer scope

depending on the data structure shouldn't your array3D already have methods to iterate over points there? like an each_point?