

Ruby master - Feature #4475

default variable name for parameter

03/06/2011 10:48 PM - jordi (jordi polo)

| | |
|--|-------------------------|
| Status: | Closed |
| Priority: | Normal |
| Assignee: | nobu (Nobuyoshi Nakada) |
| Target version: | |
| Description | |
| =begin | |
| There is a very common pattern in Ruby: | |
| <pre>object.method do variable_name variable_name doing something end</pre> | |
| Many times in fact the name of the object is so self explanatory that we don't care about the name of the variable of the block. It is common to see things like : | |
| <pre>@my_sons.each { s s.sell_to_someone }</pre> | |
| or | |
| <pre>Account.all.each { a my_account << a.money }</pre> | |
| People tend to choose s or a because we have the class or the object name just there to remind you about the context. | |
| I would like to know if can be a good idea to have a default name for that parameter. I think it is Groovy that does something like: | |
| <pre>Account.all.each { my_account << it.money }</pre> | |
| Where it is automagically filled and it doesn't need to be declared. | |
| I think it is as readable or more (for newbies who don't know what is) and we save some typing :) | |
| =end | |
| Related issues: | |
| Related to Ruby master - Misc #15723: Reconsider numbered parameters | Feedback |

Associated revisions

Revision 12acc751 - 03/17/2019 05:21 AM - nobu (Nobuyoshi Nakada)

Numbered parameters [Feature #4475]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@67278 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 67278 - 03/17/2019 05:21 AM - nobu (Nobuyoshi Nakada)

Numbered parameters [Feature #4475]

History

#1 - 03/21/2011 06:57 PM - wardrop (Tom Wardrop)

I like the suggestion. The magic variable I'd use for this pattern would be 'this'. For example:

```
posts.each { this.author = 'Santa Clause' }
```

The rule would be: The first argument of any block would be accessible from within the block through the special variable 'this'.

#2 - 03/18/2012 07:05 PM - nahi (Hiroshi Nakamura)

- Description updated

- Category set to core
- Status changed from Open to Assigned
- Assignee set to matz (Yukihiko Matsumoto)

#3 - 11/20/2012 09:26 PM - mame (Yusuke Endoh)

- Target version set to 2.6

#4 - 12/25/2017 06:14 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#5 - 04/19/2018 07:51 AM - knu (Akinori MUSHI)

I would like the feature, but we have many things to think about.

We would not be able to make "it" a reserved keyword because that would destroy all existing RSpec code written in tens of thousands of projects.

If "it" were to be implemented as a method, how could we make it work inside of a BasicObject instance? What if a method of the same name was defined?

If "it" were to be implemented as a local variable, should its name be included in local_variables?

All things considered, I guess the variable name would have to be \$-something, if any.

#6 - 04/19/2018 11:42 AM - matz (Yukihiko Matsumoto)

This is a very interesting idea but at the same time, it's difficult to keep compatibility. At least simple addition of it does not work well.

Matz.

#7 - 09/13/2018 05:44 AM - long_long_float (kazuki niimi)

I have one idea. We can use `\it` (backslash-it) or `\1` instead of `it`. `\1` means the first argument and `\n` refers nth argument.

`\-something` is not used, but `\` means line continuation.

#8 - 11/06/2018 02:15 PM - shevegen (Robert A. Heiler)

I also like the idea in principle, largely because I can relate to the idea. I encounter this every now and then when writing ruby code, in that I want a simple way to access a concept or argument.

For example, consider ruby code like this:

```
object_that_responds_to_each.each {|x|
object_that_responds_to_each.each {|this_file|
```

Both variants essentially are the same here, e. g. we iterate over a collection, and we refer to the content through the name for the block parameter. In the first case, I use `x` as name of that variable; in the second, I use the longer `this_file` name.

The second variant is a bit more understandable, I think, e. g. if we iterate over the content of a directory, such as `Dir["*"]` or something like that. Then it may be easier to understand to do things such as:

```
if File.file? this_file
```

or, if we don't know that it is a file yet, I tend to call it entry:

```
if File.file? entry
```

Still, this variant is shorter:

```
if File.file? x
```

The first variant, aka the variant with "x", or any other single letter, including "_", is something I use A LOT, and the reason is actually because it is shorter and simpler to type for me.

There are trade-offs between readability and ease-of-use, and since I am quite a lazy person, I often use the first, shorter variant. (I do so less when there is more than

one block parameter, but sometimes I do too, such as via `iterator.each {|a, b, c, d| }`.

Matz mentioned above that compatibility is an issue, but if we ignore this for the moment, I really think that the proposal is a very interesting one. So I support the idea behind the suggestion (not necessarily that particular implementation, but the idea).

As for the specific syntax change, such as in:

```
Account.all.each { |a| my_account << a.money }
Account.all.each { my_account << it.money }
```

I have no particular pro or con opinion.

I think the first old variant, the status quo, while longer, is more explicit; in the second variant there is a bit more "magic", unless we know that "it" would refer to the first block parameter.

I am not sure if "it" is the ideal name, but I think the idea has merits on its own, irrespective of which name is ultimately chosen or which concept. (May have to be some special object perhaps, that is like an array, so we could do ... `it[0] it[1]` too... if we can access more than one parameter; and perhaps also support `it.method_call_here`, but again, these are mostly specific details I think.)

Also I would like to mention that in the much longer case, such as this one here:

```
object_that_responds_to_each.each {|this_file|
```

I may have more variables like:

```
object_that_responds_to_each.each {|this_file, that_directory, that_thing|
```

And previously before, I also wanted a programmable way to refer to the first, second, and third block parameter, something short like the regex `$1`, `$2`, `$3`.

I like the short global regex variables, such as `$1` or `$2`, mostly because it is so trivial to refer to them. `MatchData` also allows for this e. g. `[0]`, `[1]` and so forth, but I found that typing `$1`, `$2` is so much easier.

I also wanted to have a programmatic way to access block parameters, similar to `$1`, `$2` - but I was unable to come up with a good syntax proposal. (All proposals should ideally be short, because if they are long then they have a smaller net benefit. `$1` is short; `[0]` is a bit longer; even longer names are diminishing some of the advantage, since we ideally need to type less, rather than more.)

So I think one big advantage of the proposal is that we could drop block parameters, if we were to have some implicit default (and one advantage with "it", even though the name may not be perfect, is that it is very short to type).

To the suggestion of "it", specifically - I do not like it mostly because I think `\` is not a very good character there. I disliked that in PHP for namespaces, even though I have not really used PHP since a long time.

If the question would be between `"\it"` and `"it"` then I would pick the second variant, `"it"`.

But I am not sure about the name "it" as such - perhaps we may have to refer to some general new name that can refer to the internal of a block? A bit like "self" but only specific for block variables or something like that? I can't come up with a good name though. Perhaps it is ok to call this "it" informally for the time being, even though I am not sure it is a great name. But whatever the name, I think it should be short. "self" is quite short. `yield_self` was never that short but I think matz added an alias to `yield_self` some time ago (if it is "then", which has the advantage that it is short, even though `yield_self` may be more explicit. But is significantly longer.).

Anyway, apologies for the length of my comment; I think the idea is a good one. If it can not happen before ruby 3.x, perhaps we can aim for it in a long path towards ruby 4.x (if we can have more incompatibilities along the way).

#9 - 11/06/2018 02:33 PM - Hanmac (Hans Mackowiak)

shevegen (Robert A. Heiler) wrote:

The first variant, aka the variant with "x", or any other single letter, including "_", is something I use A LOT, and the reason is actually because it is shorter and simpler to type for me.

"_" is special there. it doesn't cause Syntax Error
like you can do this `iterator.each { |a, _, c, _| }`
but `iterator.each { |a, x, c, x| }` does crash

#10 - 01/19/2019 05:58 AM - jeremyevans0 (Jeremy Evans)

matz (Yukihiro Matsumoto) wrote:

This is a very interesting idea but at the same time, it's difficult to keep compatibility.
At least simple addition of it does not work well.

If we want to support implicit block arguments, in order to best keep compatibility, we would probably want to add support for new syntax, using something that is currently a syntax error. knu mentioned `$-something`, but I think in most cases that would not be a syntax error. One possible option is a bare `@` for the implicit block argument:

```
Account.all.each { my_account << @.money }
```

This could be extended to support multiple block arguments:

```
{1=>2, 3=>4}.map { @1 + @2 }  
# => [3, 7]
```

I'm not sure it's a good idea to support this, especially for multiple block arguments, because it can make code harder to understand. In a lot of cases, variable names used in blocks are helpful when reading code to understand what the code is doing. Granted, if you are using single letter variable names for block arguments, switching to this syntax for implicit arguments would not result in much loss of understanding.

Another issue with implicit block arguments would be arity:

```
lambda{ @ }.arity # 0 or 1?  
lambda{ [@1, @2] }.arity # 0 or 2?  
lambda{ @2 }.arity # 0 or 2?  
s = '@2'  
lambda{ eval s }.arity # ???
```

#11 - 02/07/2019 07:35 AM - matz (Yukihiro Matsumoto)

I still feel weird when I see `@` and `@1` etc. Maybe I will get used to it after a while.
I need time.

Matz.

#12 - 02/08/2019 04:35 PM - janfri (Jan Friedrich)

I like the ampersand prefix of Elixir:

```
Enum.map [1, 2, 3], &(&1 + 2) # => [3, 4, 5]  
Enum.reduce [1, 2, 3], &(&1 + &2) # => 6
```

The ampersand (`&`) relates more to a block in Ruby than an at-sign (`@`) which is associated with an instance variable context. So my preferred syntax for Ruby would be:

```
[1, 2, 3].map { &1 + 2 } # => [3, 4, 5]  
[1, 2, 3].reduce { &1 + &2 } # => 6
```

But I'm not sure if this could be realized without syntax conflicts.

#13 - 02/08/2019 04:55 PM - jeremyevans0 (Jeremy Evans)

janfri (Jan Friedrich) wrote:

So my preferred syntax for Ruby would be:

```
[1, 2, 3].map { &1 + 2 } # => [3, 4, 5]  
[1, 2, 3].reduce { &1 + &2 } # => 6
```

But I'm not sure if this could be realized without syntax conflicts.

The problem with that is the syntax can be valid Ruby already, so it could break backwards compatibility:

```
class Integer  
  def to_proc  
    proc{|v| self + v}
```

```
end
end
[1, 2, 3].map{|x| ([x]*2).map(&1 + 2)}
```

#14 - 03/11/2019 04:46 AM - matz (Yukihiro Matsumoto)

- Assignee changed from matz (Yukihiro Matsumoto) to nobu (Nobuyoshi Nakada)

I accept @1 idea. [nobu \(Nobuyoshi Nakada\)](#) will implement it. Feel free to try it when it is merged.

Matz.

#15 - 03/11/2019 02:36 PM - nobu (Nobuyoshi Nakada)

<https://github.com/ruby/ruby/compare/trunk...nobu:feature/4475-numarg>

#16 - 03/17/2019 05:21 AM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

Applied in changeset [trunk|r67278](#).

Numbered parameters [Feature [#4475](#)]

#17 - 03/18/2019 11:42 AM - shevegen (Robert A. Heiler)

I think @1 @2 is ok, syntax-wise; it reminds me of \$1 \$2 for regex matching.

The main two advantages I see is that it is short to type; and also easy to remember.

There is only one thing that I would like to add and this came from examples I saw elsewhere; in that the suggestion seems to allow us to completely omit specifying parameters in blocks, like:

```
collection.each {|a, b, c|
}
```

versus

```
collection.each {
}
```

Can we use @1 @2 in both variants?

I think I may have had a slightly similar suggestion to the one listed here above 8 years ago, but my thought was that we'd still have to specify the names to the parameter (variables), so I thought we would use something like:

```
collection.each {|long_name_a, long_name_b, long_name_c|
  pp @1
  pp @3
}
```

I do not mind either way, by the way - but when it comes to documentation and also examples, I think it would help to mention it up front which variant(s) people could use. The best may be to allow flexibility in both variants, e. g. to specify the name of the parameters, but to also be able to omit it, while still being able to use the positional targeting here, such as through @1 @2 and so forth. And, if this is the case, to also give an example or two in the main documentation for this functionality.

Thanks.

#18 - 03/18/2019 03:43 PM - schneems (Richard Schneeman)

Would it be possible to make this more inline with other "magic" variables? Such as \$1 and \$2 for regex. Maybe \$blockarg1 and \$blockarg2?

#19 - 03/18/2019 05:12 PM - bozhidar (Bozhidar Batsov)

I also dislike the use of @ for this. It looks very weird and evokes strong references to instance variables in my mind. I'm not sure which characters exactly are an option, but I think we should avoid the use of @. Will something like %1 work? (that's what Clojure uses, which % being the same as

%1.

In general I question I lot the need for this and its usefulness, as short names are usually useful only in short blocks with 1/2 params, otherwise the readability of the code is impacted a lot. We already have a good shorthand for blocks with one param (at least in the most common case), so it feels to me we're complicated the parser/syntax without a very strong case for doing so. Many features are useful to some extent, but we should also keep in mind that none of them come for free.

#20 - 03/18/2019 10:48 PM - gkemmy (Gray Kemmey)

Is there a reason to do this other than to save some typing? I'm not sure we've proposed any terribly real-world cases here where this made code better.

I feel like we already have some single-character patterns around blocks with one or two arguments, that aren't that onerous to type and still convey meaning. For example:

```
10.times { |i| ... } # i for index
10.times { |n| ... } # n for number

[1, 2, 3].map { |n| ... } # n for number
[1, 2, 3].map { |e| ... } # e for element

['one', 'two'].map { |s| ... } # s for string

[1, 2, 3].each_with_index { |e, i| ... } # e for element, i for index
{ }.each { |k, v| ... } # k for key, v for value

[[], []].each { |r, c| ... } # two-d arrays: r for row, c for column

File.open('') { |f| ... } # f for file
```

For anything more complex, is it bad too *have* to name your arguments?

In no particular order, some downsides I see:

1. It adds some cognitive load to remember the position of arguments. Like in the `{ }.each` example -- is `@1` the key or the value? It's obvious when using `k` and `v` which is which.
2. It blurs the line between instance variables.
3. I don't know that we use simple blocks like this *most* of the time, which means, as a newcomer, you won't see this regularly, and it'll be confusing when you do. Think about the first time you saw something like `[1,2,3].reduce(&:+)`. Was that clear to anyone? (*Though in the & case, certainly worth the complexity.*)
4. It's always a little strange when things like this aren't 0-based, right?
5. Personally, find it a little jarring to iterate over an array and then use `@1` to reference each element. "At one" just *feels* positional (like an index).

Anyway, this seems like it adds a fair amount of complexity and sacrifices a lot of readability, just to save ~2 characters. Happy to hear why I'm wrong though...

#21 - 03/22/2019 12:19 AM - HarlemSquirrel (Kevin McCormack)

As I'm reading through this I am warming up to this idea. However, I would prefer to keep `@` for instance variables.

Ruby currently does not allow naming variables or methods starting with a number. Could we utilize this?

```
enumerable_object.map { 1a.method(2a) + 3a * 4a }
```

The `a` in `1a`, `2a`, etc. could be thought of as an abbreviation for "argument".

I might also suggest that the sequence start with `0a` since we have what feels like an "array" of parameters and array indexes begin at 0.

```
enumerable_object.map { "#{0a} comes before #{1a}" }
```

#22 - 03/22/2019 12:31 AM - duerst (Martin Dürst)

For those who want to change anything, I suggest to create new feature issues. It doesn't make sense to continue discussing a closed issue.

#23 - 03/22/2019 06:56 AM - pascalbetz (Pascal Betz)

gkemmy (Gray Kemmey) wrote:

Is there a reason to do this other than to save some typing? I'm not sure we've proposed any terribly real-world cases here where this made

code better.

Exactly my thought: Why? what will this make better or easier?

IMHO it does exactly the opposite, it makes Ruby harder to read. By introducing @vars that are NOT instance vars. Code will be harder to read and understand (if it has an @ and starts with a number it is this, otherwise it is that). It saves some typing but short block-vars already possible now:

```
each {|a| puts a}
```

vs.

```
each { puts @1 }
```

I love ruby because it allows me to write concise and readable code. Naming variables is part of this.

#24 - 03/22/2019 01:11 PM - sos4nt (Stefan Schübler)

Since this ticket is closed, I've opened a new one to express my concerns:

<https://bugs.ruby-lang.org/issues/15723>

#25 - 03/22/2019 01:49 PM - mame (Yusuke Endoh)

- Related to Misc #15723: Reconsider numbered parameters added