Ruby trunk - Feature #4475

default variable name for parameter

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

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

Description

=begin

There is a very common pattern in Ruby:

object.method do |variable_name|
variable_name doing something

end

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 :

@my_sons.each { |s| s.sell_to_someone }

or

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

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:

Account.all.each { my_account << it.money }

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

History

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

=begin
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'.
=end

#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 (Yukihiro 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 MUSHA)

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 (Yukihiro 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 vir 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 propsal 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."

ut 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 incompatibilites 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