Ruby master - Feature #4830

Provide Default Variables for Array#each and other iterators

06/05/2011 05:23 PM - lazaridis.com (Lazaridis Ilias)

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

Description

for arrays: use "item" by default
for hashes: use "key" and "value" by default

names = ["Jane", "Michele", "Isabella"]
names.each { |name| print name, "\n" }
names.each { print item, "\n" }

contact = {name:"Jane", phone:"1234567"}
contact.each { |key, value| print key, ": ", value, "\n"}
contact.each { print key, ": ", value, "\n"}

-

The benefits are:

- more compact code (without loosing clarity of the code).
- no repetitions ("names, name, name") in a one-liner with {} block

This extension does not break any existent behaviour.

History

#1 - 06/05/2011 07:23 PM - judofyr (Magnus Holm)

What happens in this case?

```
item = 1
[1, 2, 3].each { print item }
```

// Magnus Holm

#2 - 06/05/2011 08:04 PM - lazaridis.com (Lazaridis Ilias)

[please, if possible, delete the non-relevant quoted message in your reply.]

Magnus Holm wrote:

> What happens in this case?
>
> item = 1
> [1, 2, 3].each { print item }

Possibly the same as with

```
item = 1
[1, 2, 3].each { |item| print item }
```

The outer local variable gets shadowed by the inner one.

But:

the first step is to determine if such feature is to be included into the language.

If it's decided, then the implementation details, possible problems and tradeoffs become relevant.

#3 - 06/06/2011 12:20 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

- Priority changed from Normal to 3

#4 - 06/06/2011 12:23 AM - nobu (Nobuyoshi Nakada)

Hi,

At Sun, 5 Jun 2011 17:23:26 +0900,
Lazaridis Ilias wrote in [ruby-core:36750]:

> for arrays: use "item" by default
> for hashes: use "key" and "value" by default

Why different names?

There is no way to tell what class the receiver is to the
parser. Your proposal needs very big change under the hood.

--
Nobu Nakada

#5 - 06/06/2011 01:06 AM - lazaridis.com (Lazaridis Ilias)

Nobuyoshi Nakada wrote:

> Hi,
>
> At Sun, 5 Jun 2011 17:23:26 +0900,
> Lazaridis Ilias wrote in [ruby-core:36750]:
>
> > for arrays: use "item" by default
> > for hashes: use "key" and "value" by default
>
> Why different names?
>
> There is no way to tell what class the receiver is to the
> parser. Your proposal needs very big change under the hood.

I don't know the underlying implementation, so I suggested simply from a users view.

If the arrays gets "value" instead of "item", it would be fine, of course.

#6 - 06/06/2011 03:53 AM - Cezary (Cezary Baginski)

On Mon, Jun 06, 2011 at 01:07:05AM +0900, Lazaridis Ilias wrote:

> Issue #4830 has been updated by Lazaridis Ilias.
>
> Nobuyoshi Nakada wrote:
>
> > At Sun, 5 Jun 2011 17:23:26 +0900,
> > Lazaridis Ilias wrote in [ruby-core:36750]:
> >
> > > for arrays: use "item" by default
> > > for hashes: use "key" and "value" by default
> >
> > There is no way to tell what class the receiver is to the
> > parser. Your proposal needs very big change under the hood.

Maybe something like special variables, e.g. $1, $2, ... instead?

That would remove the need to check the type, but reduce readability,
etc.

Personally, I prefer to be able to give specific names other than
"key", "value", "item", like "person", "date", "sex", etc.

Might be useful for more generic code, OTOH.

--
Cezary Baginski

#7 - 06/06/2011 03:59 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

I would just like to point it out that in Groovy, one can write either:

[1, 2].each{ println it }

or

[1, 2].each{ number -> println number }

[key_name: 'value', another_key: 'another_value'].each{ println "${it.key}: ${it.value}"}

#8 - 06/06/2011 03:59 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

ignore... can't remove this comment - ammended with the above

#9 - 06/06/2011 04:32 PM - lazaridis.com (Lazaridis Ilias)

[please, if possible, delete the non-relevant quoted message in your reply. You can still do this via http://redmine.ruby-lang.org/issues/4830]

Rodrigo Rosenfeld Rosas wrote:

> Sorry, forgot to say that for hashes this becomes:
>
> [key_name: 'value', another_key: 'another_value'].each{ println
> "${it.key}: ${it.value}"}
> [...]
> Em 05-06-2011 15:56, Rodrigo Rosenfeld Rosas escreveu:
>
>> I would just like to point it out that in Groovy, one can write either:
>>
>> [1, 2].each{ println it }
>> [...]

"it" is not speakable.

"with each it"
"for each it"

where "item" or "value" is speakable

"with each item" or "with each value"
"for each item" or "for each value"

#10 - 06/06/2011 08:15 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I'm not saying we should copy Groovy syntax or ideas. I'm just showing that this is already done in Groovy.

At first I liked the idea of not needing to define an internal variable, but as I started to use it, I run into trouble where it was a bit difficult to figure out what was going wrong in cases like this:

```
someCollection.each {
   doSomeOperationWith(it)
   Thread.start { println it } // "it" here is not an item from the collection
}
```

#11 - 06/08/2011 05:01 PM - headius (Charles Nutter)

I've gone back and forth on whether I like this feature in Groovy. For simple iterations or chained iterations, it definitely shortens things up:

some_array.map { foo(it) }.select { bar(it) }.each { baz(it) }

versus

some_array.map {|it| foo(it) }.select {|it| bar(it) }.each {|it| baz(it) }

Perhaps something more Scala-like:

some_array.map { foo() }.select { bar() }.each { baz(_) }

That's actually fairly clean in the normal form and explicitly passing arguments as normal:

some_array.map {|| foo() }.select {|| bar() }.each {|| baz() }

Yes, it looks like ass. But I think it's better to be explicit than implicit most of the time (anti-magic variable) and better to just use words rather than symbols (anti-special char or $ variable).

Ask me again tomorrow and I might have changed my mind and love the feature.

FWIW, I have implemented "it" in JRuby previously, just for fun. It's not hard to add, if the powers decide it's a good feature for Ruby.

#12 - 06/09/2011 07:54 PM - lazaridis.com (Lazaridis Ilias)

Charles Nutter wrote:
[...]

> FWIW, I have implemented "it" in JRuby previously, just for fun. It's not hard to add, if the powers decide it's a good feature for Ruby.

Some important details:

- the user can choose to use explicit variables.
- existent code behaves exactly the same
- only if existent code is changed to use defaults, care must be take to not shadow local variables (should be very rare that an "accident" happens)
- a user can decide to write only new code with defaults

#13 - 06/10/2011 09:28 AM - rbjl (Jan Lelis)

There is (almost) a case where it is already possible:

```
"string".gsub(/./){ |e|
  # use e
}
```

vs.

```
"string".gsub(/./){
  # use $&
}
```

I always (except in code-golfing) end up with the first solution. However, when using subgroups, I just love to use them:

"string".gsub(/com(plex)_reg(ex)/){
# use $1, $2, ...
}

To get back to the original problem, I'd prefer: $item (which is -of course- not a global variable).

PS: Since $dollar variables are rarely used anyway, lets transform them all into special variables :D

#14 - 06/10/2011 10:53 PM - aprescott (Adam Prescott)

On Fri, Jun 10, 2011 at 1:28 AM, Jan Lelis mail@janlelis.de wrote:

> "string".gsub(/com(plex)_reg(ex)/){
> # use $1, $2, ...
> }
>
> To get back to the original problem, I'd prefer: $item (which is -of
> course- not a global variable).
>
> PS: Since $dollar variables are rarely used anyway, lets transform them all
> into special variables :D

I think the ${1,2,3,...} variables match the gsub(regex, "a $1 replacement
$2 here $3").

While using $item, or any other identifier, as the implicit argument might
make some code a few characters shorter to write, what other benefits are
there? All I can see is that it adds one more thing to learn in the language
and introduces greater complexity, both for people who've never seen it, and
for scoping rules. I see nothing wrong with the explicit { |item| ... }.

It's readable, it works.

#15 - 06/10/2011 10:53 PM - aprescott (Adam Prescott)

On Fri, Jun 10, 2011 at 2:32 PM, Adam Prescott adam@aprescott.com wrote:

> I think the ${1,2,3,...} variables match the gsub(regex, "a $1 replacement
> $2 here $3").

Sorry, that wasn't really clear, and the latter code should be \1 not $1. I
shouldn't rush!

What I mean by this is that the variables have a clear association to
capturing group numbers. Experience with regular expressions in Ruby and
elsewhere will suggest to you what $1 will get replaced to in the block-form
of gsub. The general implicit block argument $item does not, at least to me.

#16 - 09/21/2011 07:36 AM - alexeymuranov (Alexey Muranov)

I agree with Adam.
In my opinion, this would be "convention over configuration" pushed to extreme.
A person seeing for the first time a piece of code with these default variables will have no way to know what is going on.
"Conventions over configuration" should be easy to override, but here it is expected to become a static syntax.
Are there any other default variable names in Ruby at all (other than self)?
If not, this will be a complicating innovation.

Alexey.

#17 - 09/21/2011 09:25 AM - agrimm (Andrew Grimm)

=begin
For a simple case, such as

(({some_array.map { foo(it) }.select { bar(it) }.each { baz(it) }}))

You can do

(({some_array.map(&method(:foo)).select(&method(:bar)).each(&method(:bar))}))

instead.

> PS: Since $dollar variables are rarely used anyway, lets transform them all into special variables :D

Maybe we could migrate global variables into katakana, so we'd have local_variable, CONSTANT, $special_variables and □□!
=end

#18 - 02/08/2012 05:04 PM - shevegen (Robert A. Heiler)

The last example:

some_array.map(&method(:foo)).select(&method(:bar)).each(&method(:bar))

Is no advantage. First, it is longer. Second, using & is ugly.

I think it would be nice to access block arguments by position somehow, without naming them specifically.

The problem with:

some_array.map { foo(it) }.select { bar(it) }.each { baz(it) }

Is that it is not obvious where "it" comes from.

How about this though:

some_array.map { |it| foo(it) }.select { bar(it) }.each { baz(it) }

In this example, if a block variable was not found, it first checks
whether another block variable with its name was defined for the
object the operation is done on. In the above example, bar(it) would
understand that "it" is the first block argument.

I am not sure if this would complicate things.

What I myself sometimes want to do is this:

array.map { foo() }.select { bar() }.each { baz(_) }

This looks cleaner to me than:

array.map { || foo() }.select { || bar() }.each { || baz() }

A compromise could be:

array.map { || foo() }.select { bar() }.each { baz() }

But it's still not all too terribly beautiful either compared to the:

array.map { foo() }.select { bar() }.each { baz(_) }

It would be nice if Ruby could support some kind of implicit names
though. There already are some ruby conventions, such as constants
have to start with an UPCASED character. And noone hates such a
convention.

In principle it could be possible to allow a naming convention that
can be accepted by folks.

PS: some_array.map(&method(:foo)).select(&method(:bar)).each(&method(:bar))
is really ugly. It is much uglier than what Ilias suggested too!

names.each { |name| print name, "\n" }
names.each { print item, "\n" }

My only gripe with his proposal is that noone really knows where "item"
came from. The advantage of _ would at least be that noone needs a
name for it, but also noone quite knows where it came from. :/

How about another crazy proposal though:

names.each { print argument_1, "\n" }

This is such an unlikely name ... could be made available in every
method too!

argument_1
argument_2
argument_3

PSS: I know that noone likes it. But hey, it can still be used for
discussion.

#19 - 11/20/2012 09:10 PM - mame (Yusuke Endoh)

- Target version set to 2.6

#20 - 09/25/2013 02:59 AM - alexeymuranov (Alexey Muranov)

How should nested blocks behave?

1, 2], [3, 4.map { item.map { item + 1 } }

#21 - 09/25/2013 03:29 AM - fuadksd (Fuad Saud)

I don't like this. The only thing that doesn't hurt so much would be
something like scala's underscores for one parameter blocks, but I'm not
really sure about those either.

--
Fuad Saud

twitter http://twitter.com/fuadsaud |
linkedinhttp://www.linkedin.com/in/fuadksd|
coderwall http://coderwal.com/fuadsaud | githubhttp://github.com/fuadsaud|

#22 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#23 - 01/24/2018 08:40 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

Rejected. It would cause too much confusion than convenience.

Matz.