Ruby master - Feature #15897

`it` as a default block parameter

06/04/2019 05:12 AM - mame (Yusuke Endoh)

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

Description

How about considering "it" as a keyword for the block parameter only if it is the form of a local varaible reference and if there is no variable named "it"?

```
[1, 2, 3].map { it.to_s } #=> ["1", "2", "3"]
```

If you are familiar with Ruby's parser, this explanation is more useful: NODE_VCALL to "it" is considered as a keyword.

Examples:

```
public def it(x = "X")
  x
end

[1, 2, 3].map { it.to_s }    #=> ["1", "2", "3"]
[1, 2, 3].map { self.it }    #=> ["X", "X", "X"] # a method call because of a receiver
[1, 2, 3].map { it() }       #=> ["X", "X", "X"] # a method call because of parentheses
[1, 2, 3].map { it "Y" }     #=> ["Y", "Y", "Y"] # a method call because of an argument
[1, 2, 3].map { it="Y"; it } #=> ["Y", "Y", "Y"] # there is a variable named "it" in this scope

it = "Z"
[1, 2, 3].map { it.to_s }    #=> ["Z", "Z", "Z"] # there is a variable named "it" in this scope
```

Pros:

- it is the best word for the feature (according to matsuda (Akira Matsuda))
- it is reasonably compatible; RSpec won't break because their "it" requires an argument

Cons:

- it actually brings incompatibility in some cases
- it is somewhat fragile; "it" may refer a wrong variable
- it makes the language semantics dirty

Fortunately, it is easy to fix the incompatible programs: just replace it with it(). (Off topic: it is similar to super().)
Just inserting an assignment to a variable "it" may affect another code. This is a bad news, but, IMO, a variable named "it" is not so often used. If this proposal is accepted, I guess people will gradually avoid the variable name "it" (like "p").
The dirtiness is the most serious problem for me. Thus, I don't like my own proposal so much, honestly. But it would be much better than Perlish @1. (Note: I don't propose the removal of @1 in this ticket. It is another topic.) In any way, I'd like to hear your opinions.

An experimental patch is attached. The idea is inspired by jeremyevans0 (Jeremy Evans)'s proposal of @.

P.S. It would be easy to use _ instead of it. I'm unsure which is preferable.

Related issues:

Related to Ruby master - Misc #15723: Reconsider numbered parameters	**Feedback**

History

#1 - 06/04/2019 05:12 AM - mame (Yusuke Endoh)

- Related to Misc #15723: Reconsider numbered parameters added

#2 - 06/04/2019 07:28 AM - Hanmac (Hans Mackowiak)

_ can't be used as default block parameter because it already has a special meaning when using block variables like {|a,_,_,b| }

#3 - 06/04/2019 07:42 AM - mame (Yusuke Endoh)

Hanmac (Hans Mackowiak),

You cannot use both it and the ordinal parameter |a,_,b| simultaneously. It will cause a SyntaxError like this.

```
$ ./miniruby -e '1.times {|a,_,b| it }'
-e:1: ordinary parameter is defined
```

#4 - 06/04/2019 07:51 AM - Hanmac (Hans Mackowiak)

mame (Yusuke Endoh) you got me wrong, i mean you might not use _ for this like you said in your P.S.

#5 - 06/04/2019 08:15 AM - shevegen (Robert A. Heiler)

I was about to write a very lengthy reply, but I think it would be too difficult to read for others, so this is a somewhat shorter variant. So just the main gist:

(1) I don't quite like the name "it", mostly due to semantics (the name does not tell me much at all), but one advantage is that "it" is short to type. I do not have a good alternative name, though. _ as a name would be even shorter, and avoids some of the semantic-meaning problem, but may have other small issues - see a bit later what I mean here.

(2) Even though I do not like the name "it", to me personally, it would be better to see that BOTH @1 @2 and "it" would be added, in the sense that then people could just pick what they prefer. Obviously I see no problem with @1 @2 at all, so I am biased. But that way people could just use whatever they prefer. (There could be another name, of course ... I thought about some way for acessing block/procs data ... such as **BLOCK** or ProcData or something like that. One problem is that this is all longer to type than e. g. "it" or @1 but perhaps we could use some generic way to access what a proc/block represents anyway, even aside from the proposal itself, similar to ... **LINES** or **method** or **FILE** or so. Just so that we may have a concept for it; although it will probably not be widely used. But I have not really thought about this much. Note that I was wondering about something like this for case/when structures as well, so that we could access them more easily, also from "outside" of methods, but I digress here).

IF only "it" alone were to be added, then I would rather prefer that neither @1 @2 nor "it" would be added, and we'd all only use the oldschool way (which is fine, too). But as said, I am biased so I think @1 @2 etc... are perfectly fine.

I think this is also a problem I have with the "this is perl" comments in general - to me it is not like perl anywhere. And the old variant such as:

```
foo.each {|a,b,c|
```

just continue to work fine; so to me the change is primarily about adding more flexibilty. This was a problem I have had with the use cases that were mentioned - people would be very eager to point out what they dislike (understandably so), but at the same time would not want to mention use cases that may be useful. So I think in general, it would be better to be more objective in giving examples, even if a certain functionality is disliked. Use cases should ideally be extensive, not just focusing on what the ruby user at hand may dislike the most (since they may tend to focus on that first, which is understandable, and then ignore anything else; I tend to do so myself sometimes).

To clarify this - my primary problem with "it" is the name itself, not the functionality that is associated with it.

Using _ avoids the name/semantic issue a bit, to some extent, but I think _ has some slight other issues.

For example, the _ variable is used quite a lot in ruby code out there, or at the least in some code bases, so I am not sure it would be a good name here. Note that I use _ as "throwaway" variable a lot in my own code. This should be kept in mind, at the least for when ruby users do something similar (I have no idea whether it is common or not, though). I think hanmac sort of reffered to this too in a way.

Since I like _ a lot, I'd rather see "it" be added than _, because I will most likely not use "it" in my own code ;D , whereas I would still use _ fine, possibly even within blocks, and possibly @1 @2 too, at the least for quick debugging, if it were to stay/remain, which I hope it will. But I am biased. ;)

I should also note that while I think @1 @2 are perfectly fine, I also don't have a big problem if it were not to be added permanently, even though I think it is fine if it would, evidently. The oldschool way is the best.

Since I see @1 @2 mostly as a convenience feature, though, I can continue to work with ruby just fine. I just don't think that all prior statements in particular in the other thread(s) made a whole lot of sense; and I have no problem if "it" would be added either as well, since I can avoid it, and just use @1 @2 for debugging. ;)

I think, realistically, I assume that most ruby users will continue to just use ruby code like it used to be, like:

```
foobar.some_method {|a, b, c, d, _, f|
```

I am quite certain that I will keep on using the above variant, and that neither "it" nor @1 @2 would persist in my own code - but for quick debugging, in particular for longer names, I think @1 @2 is really great. I don't think "it" would be equivalent to this, though; at the least to me, "it" is not the same as e. g. @1 @2 in several ways. But as said, I have no problem at all if both variants would be added.

Of course I am not naive - when features are added/offered, people will use it and play around with it; adults are kids after all, just play with different things. ;) I just don't think that the primary focus for dislike should be limited to just some use cases, without considering situations such as e. g. @1 @2 not be used in production code, but just for debugging purposes alone. I don't have a
problem with the scenario where we can avoid naming parameters, but to me this is not the primary use case I would like to focus myself - for me the

"pp @1; pp @3" variant really is the more important aspect of the suggestion. When you come from this point of view then I think it is easy to understand that "it", aside from the name, is not exactly the same.

Last but not least, as mame wrote - I think if you have not yet commented on either @1 @2 (in other issues) and/or "it" (here in this proposal), it may be good to comment on the suggestion/idea itself here. Matz actually asked for feedback before, not only in the other thread but also the old(er) ones predating these.

#6 - 06/04/2019 08:32 AM - phluid61 (Matthew Kerwin)

Hanmac (Hans Mackowiak) wrote:

> mame (Yusuke Endoh) you got me wrong, i mean you might not use _ for this like you said in your P.S.

Isn't {|_| _ } no more or less conflicting than {|it| it } ? You can either use positional args, or a default arg, but not both. So {|_| _ } means what it currently means, irrespective of this proposal.

Unless you mean there's a chance of an outer scope that already uses _ as an arg, creating a conflict in the inner scope?

#7 - 06/04/2019 08:32 AM - phluid61 (Matthew Kerwin)

shevegen (Robert A. Heiler) wrote:

> I was about to write a very lengthy reply, but I think it would be too difficult to read for others, so this is a somewhat shorter variant.

Good grief.

#8 - 06/04/2019 02:29 PM - mikegee (Michael Gee)

> RSpec won't break because their "it" requires an argument

Unfortunately this is not accurate. RSpec has a shorthand style like this:

subject { fortytwo }
it { is_expected.to eq 42 }

#9 - 06/04/2019 02:40 PM - jeremyevans0 (Jeremy Evans)

mikegee (Michael Gee) wrote:

> > RSpec won't break because their "it" requires an argument
>
> Unfortunately this is not accurate. RSpec has a shorthand style like this:
>
> subject { fortytwo }
> it { is_expected.to eq 42 }

That's a block argument :). In any case, the parser treats it differently as NODE_ITER/NODE_FCALL, not as NODE_VCALL:

```
RubyVM::AbstractSyntaxTree.parse("it").children
# => [[], nil, #<RubyVM::AbstractSyntaxTree::Node:VCALL@1:0-1:2>]

RubyVM::AbstractSyntaxTree.parse("it{}").children
# => [[], nil, #<RubyVM::AbstractSyntaxTree::Node:ITER@1:0-1:4>]

RubyVM::AbstractSyntaxTree.parse("it{}").children.last.children
# => [#<RubyVM::AbstractSyntaxTree::Node:FCALL@1:0-1:2>, #<RubyVM::AbstractSyntaxTree::Node:SCOPE@1:2-1:4>]
```

Regarding the proposal itself, the dirtying of the semantics bothers me about this as well. However, I can see where people would find it cleaner than @ in terms of syntax, so this is really a tradeoff between the cleanliness of semantics and syntax. I don't have a strong opinion on it compared to @, but I think either is preferable to @1 or _.

#10 - 06/04/2019 10:24 PM - Eregon (Benoit Daloze)

I like the proposal and I think it reads nicer than @.
It's a bit magical that giving it an argument changes the semantics, but that's somewhat similar to having a p local variable, and I think it's worth the better readability and syntax.
Not so many people seem confused by p so I guess it would not be too surprising and just intuitive in common cases.

I also like _ because _ is "unnamed" (rather than abstract like it) and a "placeholder for the missing argument name" and this whole feature is about removing the need to name the block argument.
Showing them in code for an easy comparison:

```
[1, 2, 3].map { @ * 3 }
[1, 2, 3].map { @1 * 3 }
[1, 2, 3].map { _ * 3 }
[1, 2, 3].map { it * 3 }
[1, 2, 3].map { |n| n * 3 }
```

In the case of nested unnamed block arguments ([1].map { _ * 3.then { _ } }), both _ and it would refer to the outer block's argument, and consider the inner block(s) have no arguments.
That could be confusing, it might be worth warning or rejecting such cases, although they are probably rare.

> But it would be much better than Perlish @1

Strongly agreed.

#11 - 06/05/2019 01:20 AM - shugo (Shugo Maeda)

> Thus, I don't like my own proposal so much, honestly. But it would be much better than Perlish @1.

I don't like both proposals, but I prefer @1 to it because @1 looks ugly and may help prevent overuse.
Furthermore, the proposed it may be more Perlish in the sense that it depends on the context.

#12 - 06/05/2019 04:16 PM - Eregon (Benoit Daloze)

shugo (Shugo Maeda) wrote:

> I don't like both proposals, but I prefer @1 to it because @1 looks ugly and may help prevent overuse.

I think we should never purposefully introduce something ugly in the language.
Preventing overuse is I think best done by limiting to a single argument (as argued in #15723).

#13 - 06/14/2019 02:54 AM - shugo (Shugo Maeda)

Eregon (Benoit Daloze) wrote:

> shugo (Shugo Maeda) wrote:
>
> > I don't like both proposals, but I prefer @1 to it because @1 looks ugly and may help prevent overuse.
>
> I think we should never purposefully introduce something ugly in the language.

So let's reject both proposals.

> Preventing overuse is I think best done by limiting to a single argument (as argued in #15723).

I guess it will be overused when a block takes only one argument.

#14 - 06/14/2019 09:28 PM - Eregon (Benoit Daloze)

shugo (Shugo Maeda) wrote:

> > I think we should never purposefully introduce something ugly in the language.
>
> So let's reject both proposals.

That's not what I meant. I'd rather not have something ugly in the language at all.
But I think we can make it not ugly, either with _ or it proposed here.

I think readability matters a lot to many people, we typically read code more often than we write.
_ or it seem much better for readability than @ or @1.

> > Preventing overuse is I think best done by limiting to a single argument (as argued in #15723).

> I guess it will be overused when a block takes only one argument.

Maybe, but that harm would be IMHO very little, because it and _ read nicely and easily,
compared to spreading multiple numbered Perlish variables in Ruby code.

#15 - 06/15/2019 10:38 AM - janosch-x (Janosch Müller)

Kotlin has implemented it like this (docs).

From purely personal experience, after doing just a little bit of Kotlin, I often feel a temptation to use it when writing Ruby, just to notice that I can't. I found it in Kotlin natural, easy to get used to, and easy to parse visually and understand when re-reading my code after a while.

#16 - 06/17/2019 08:01 AM - shugo (Shugo Maeda)

Eregon (Benoit Daloze) wrote:

> shugo (Shugo Maeda) wrote:
>
> > > I think we should never purposefully introduce something ugly in the language.
> >
> > So let's reject both proposals.
>
> That's not what I meant. I'd rather not have something ugly in the language at all.
> But I think we can make it not ugly, either with _ or it proposed here.

it doesn't look ugly at first glance, but it makes the language semantics dirty as mame admitted in his proposal.

> I think readability matters a lot to many people, we typically read code more often than we write.
> _ or it seem much better for readability than @ or @1.

If it is a normal reserved word, I agree with you.
However, the semantics of it depends on the context, and therefore @1 is more readable for me.

#17 - 06/17/2019 08:38 AM - sawa (Tsuyoshi Sawada)

I propose to use a new keyword item.

- I feel that using a keyword spelt in letters is the right way here since keywords like self are used in other cases where we reach for things out of the blue without receiving them through argument signature.
- "Item" is close enough to "it", so we may achieve sympathy from some of the people opting for "it", but is not "it", so it does not have the problem that "it" has.

- \item is used in LaTeX as a command to introduce bullet points in listed structures, which is analogous to elements in a block led by each, map and their kins.

  ```
  [1, 2, 3].map{item ** 2} # => [1, 4, 9]
  ```

- At the same time, "item" does not exclusively mean "element". It is a neutral term regarding that. So it would not be unnatural to be used in blocks led by methods like then, tap and their kins.

  ```
  "foo".then{item + "bar" + item} # => "foobarfoo"
  ```

- I have a concern that "it" somewhat implies the receiver since it means something whose referent has been fixed in the context. In fact, the method itself returns the receiver.

#18 - 06/17/2019 09:03 AM - janosch-x (Janosch Müller)

sawa (Tsuyoshi Sawada) wrote:

> I propose to use a new keyword item.

I think that is a great proposal.

it is nice to read when passed to methods of other objects or when used with binary operators:

```
strings.each { puts it }
pathnames.map { File.read it }
numbers.map { it + 2 }
```

unfortunately, it is quite awkward to read when calling its own methods (which is probably the more common case in Ruby):

```
strings.each { it.chomp!('foo') }
pathnames.map { it.read }
numbers.map { it.next.next }
```

item works well for both cases:

```
strings.each { puts item }
pathnames.map { File.read item }
numbers.map { item + 2 }

strings.each { item.chomp!('foo') }
pathnames.map { item.read }
numbers.map { item.next.next }
```

#19 - 06/17/2019 10:44 AM - mame (Yusuke Endoh)

Don't think that this proposal can be applied to any words. A common name is much more dangerous than a pronoun like it because it is much more frequently used as a method name.

Actually, the count of "def item()" is 20 times more than "def it()" in gem-codesearch result.

```
$ csearch "^\s*def it\(?\)?$" | wc -l
12
$ csearch "^\s*def item\(?\)?$" | wc -l
225
```

Furthermore, I found some codes that will be broken if "item" becomes a soft keyword.

https://github.com/ginty/cranky/blob/ca7176da2b8e69c37669afa03fee1a242338e690/lib/cranky/job.rb#L49
https://github.com/carlosipe/mercado-libre/blob/ebb912a7c4e942eb38e649d8e11a005c288ebc92/test/mercadolibre.rb#L44

#20 - 07/02/2019 07:31 AM - akim (Akim Demaille)

FWIW, wrt "This is Perlish": at least one modern language has adopted a similar feature: Swift. I've used it in practice, and it is really nice to use.

This page has a running example expressed in different ways: https://docs.swift.org/swift-book/LanguageGuide/Closures.html

```
func backward(_ s1: String, _ s2: String) -> Bool {
    return s1 > s2
}
var reversedNames = names.sorted(by: backward)

reversedNames = names.sorted(by: { (s1: String, s2: String) -> Bool in
    return s1 > s2
})
```

This one shows the common inheritance from Smalltalk.

```
reversedNames = names.sorted(by: { s1, s2 in return s1 > s2 } )

reversedNames = names.sorted(by: { s1, s2 in s1 > s2 } )

reversedNames = names.sorted(by: { $0 > $1 } )

reversedNames = names.sorted(by: >)
```

#21 - 07/02/2019 10:24 AM - Benoit_Tigeot (Benoit Tigeot)

jeremyevans0 (Jeremy Evans) wrote:

> mikegee (Michael Gee) wrote:
>
> > RSpec won't break because their "it" requires an argument
>
> Unfortunately this is not accurate. RSpec has a shorthand style like this:
>
> subject { fortytwo }
> it { is_expected.to eq 42 }

> That's a block argument :). In any case, the parser treats it differently as NODE_ITER/NODE_FCALL, not as NODE_VCALL:

Thanks for the clarification. As member of the RSpec team but expressing my own view I was worried about a strong conflict. I still think this not a good idea to use "it" because it is used a lot in the tests your write using RSpec. It can easily lead to confusion. I think we should send other proposal for this. Sorry Mame.

#22 - 07/02/2019 11:45 AM - Benoit_Tigeot (Benoit Tigeot)

- File mame_its_proposal.patch added

I added an updated patch version. The patch has been tested with existing examples that mame shared initially.

Also I tested the patch against rspec-core without any issues. https://github.com/rspec/rspec-core/issues/2645

#23 - 07/02/2019 12:30 PM - Eregon (Benoit Daloze)

Benoit_Tigeot (Benoit Tigeot) wrote:

> Thanks for the clarification. As member of the RSpec team but expressing my own view I was worried about a strong conflict. I still think this not a good idea to use "it" because it is used a lot in the tests your write using RSpec. It can easily lead to confusion. I think we should send other proposal for this. Sorry Mame.

What do you think about _?

I'm unsure if it's confusing in practice, they are used in fairly different contexts, and it is never used alone (no args, no block) by RSpec AFAIK.
It's pretty clear in this example, isn't it?

```
describe "Array#map" do
  it "should transform the values" do
    expect([1,2,3].map { it * 2 }).to eq([2,4,6]) # RSpec 3+ style
    [1,2,3].map { it * 2 }.should == [2,4,6] # MSpec/RSpec 2 style
  end
end
```

#24 - 07/03/2019 09:26 PM - JonRowe (Jon Rowe)

□□Hello! As the current maintainer of RSpec I'm concerned about the confusion here.

```
RSpec.describe do
  it "will do the thing" do
    it # now refers to the first argument to the block? (Which ironically is the example itself)
  end
  it # creates a pending example with no implementation
end
```

So if I get a vote I'd love this to have a different name. As pointed out I think _ is confusing as it's being reused from elsewhere, but what about _1 or $1 or some other special syntax?

#25 - 07/04/2019 10:17 AM - Benoit_Tigeot (Benoit Tigeot)

Eregon (Benoit Daloze) wrote:

> What do you think about _?
>
> I'm unsure if it's confusing in practice, they are used in fairly different contexts, and it is never used alone (no args, no block) by RSpec AFAIK.

I think Jon express my feelings with a clear example.

For _ same thing. It is already use and it can lead to an issue: https://bugs.ruby-lang.org/issues/15897#note-3

#26 - 07/04/2019 04:24 PM - Eregon (Benoit Daloze)

JonRowe (Jon Rowe) wrote:

> it # creates a pending example with no implementation

Is that used in practice though? I know pending examples, but I would expect they at least have a description.

FWIW, MSpec's it requires the description argument, so it cannot ever be ambiguous for the user.
https://github.com/ruby/mspec/blob/ca2bc422cd8f8ddb23629e972372cf8e49f992fc/lib/mspec/runner/object.rb#L14

Benoit_Tigeot (Benoit Tigeot) wrote:

> For _ same thing. It is already use and it can lead to an issue: https://bugs.ruby-lang.org/issues/15897#note-3

I think that comment is unclear.
It just means named and unnamed arguments are incompatible, just like currently in trunk, foo { |named_arg| @1 } is a SyntaxError.
foo { |_| _ * 3 } and foo { _ * 3 } would both work fine.

#27 - 07/05/2019 07:44 AM - JonRowe (Jon Rowe)

> Is that used in practice though? I know pending examples, but I would expect they at least have a description.

No but as Ruby doesn't allow method overloading you do create that ambiguity, and the possibility for future bugs due to precedence rules.

> FWIW, MSpec's it requires the description argument, so it cannot ever be ambiguous for the user.

Yes it can, as you are creating method / variable shadowing in built to the language, so its always ambiguous, is this the method or the variable.

#28 - 07/06/2019 01:55 PM - Eregon (Benoit Daloze)

JonRowe (Jon Rowe) wrote:

> > FWIW, MSpec's it requires the description argument, so it cannot ever be ambiguous for the user.
>
> Yes it can, as you are creating method / variable shadowing in built to the language, so its always ambiguous, is this the method or the variable.

I think this ambiguity is fairly intuitive and easy to resolve:
it's exactly the same rule as for method calls without arguments, if we consider it is always defined as a local variable.
I.e., if there are no arguments, it's always the variable/default block parameter, otherwise it's a method call with the given arguments.

I think it's going to be extremely rare for it to be confusing in practice, once people know about the default block parameter feature.

#29 - 07/07/2019 09:04 PM - joallard (Jonathan Allard)

Eregon (Benoit Daloze) wrote:

> I think this ambiguity is fairly intuitive and easy to resolve:
>
> [...] if there are no arguments, it's always the variable/default block parameter, otherwise it's a method call with the given arguments.
>
> I think it's going to be extremely rare for it to be confusing in practice, once people know about the default block parameter feature.

I would echo Benoit's comments here. As an avid RSpec user, I employ it "does something" without a block fairly often for prototyping. However, I virtually never use a completely naked it, no arguments or block, which seems to be the issue here. Even then, if I really need a naked 'it', I could always write it().

The following, taken from Jon Rowe's example above, makes total sense to me:

```
RSpec.describe do
  it "will do the thing" do
    it # without arguments, we're referring to a thing whose meaning is given by the context
  end

  it    # meaning given by context: first block argument

  it()  # empty, description-less example (not affected)
  it{}  # also unaffected
end
```

While I do empathize with our RSpec folks, I don't see the value in preventing an expressive keyword to exist (which would be useful in a lot of cases) in order to save the ability to use the empty it, which has a pretty limited use. Especially considering if one really needs the latter, they may just write it(), a simple workaround.

The value of an expressive, language-based keyword is something I value and makes sense to me in Ruby's design based on natural language.

With that said, I'd support finding an alternative keyword, but it seems that based on previous discussions, we've not been able to find one that makes better consensus. I haven't either. Besides, we could make similar criticisms about those other keywords (item, this/that, one/other, ...)

#30 - 07/08/2019 04:36 PM - JonRowe (Jon Rowe)

Its worth pointing out that this would always have to be lower priority than methods and other such locals defined in order to allow code to work. If precedence for this was changed to later override existing definitions of it, or simply ban them as is the case with other ruby keywords, it would break RSpec and Mspec. Consider:

```
it = 'is my string'
something do
  it # what whould this be? the argument to the block or the string?
end
```

From my experience maintaining RSpec I think newcomers would either never know about it, or struggle to realise what is going on.

All just my opinions of course :)

#31 - 07/08/2019 11:37 PM - mame (Yusuke Endoh)

JonRowe (Jon Rowe) Thank you for your opinion, but I'd be happy if you could read my original proposal carefully. I've already pointed out the issue (and said my opinion against the issue).

> Cons:
>
> - it is somewhat fragile; "it" may refer a wrong variable
>
> Just inserting an assignment to a variable "it" may affect another code. This is a bad news, but, IMO, a variable named "it" is not so often used. If this proposal is accepted, I guess people will gradually avoid the variable name "it" (like "p").

I hear from some people that they are actually using a variable it. That's unfortunate, but I still think that a soft keyword "it" is the best solution, as long as we need to add a something like @1 (and matz strongly wants to add something).

#32 - 07/09/2019 11:06 AM - JonRowe (Jon Rowe)

mame (Yusuke Endoh) I did, I apologise for not making it clear, I'm reiterating it to add weight to the con, its not just a simple "it is somewhat fragile, it may refer to a wrong variable" its a "this shadows the most commonly used method in the most downloaded rubygem"1, its not one or two people this will affect.

I welcome the feature, I just think the name is wrong, something special like _1 etc would be better.

#33 - 07/09/2019 02:50 PM - Eregon (Benoit Daloze)

JonRowe (Jon Rowe) wrote:

> its a "this shadows the most commonly used method in the most downloaded rubygem"[1], its not one or two people this will affect.

It only "shadows" (by that I understand "no longer works in that case") for the case of it without any description, block and parenthesis, right?
Which seems extremely rare as said before.
So I expect only people using a plain it from RSpec would be affected, and I would expect very few people use that (it doesn't even seem tested in RSpec's test suite).

If using a local variable named it, it continues just referring to that local variable in that method.

> If precedence for this was changed to later override existing definitions of it

I believe it will never change if introduced, breaking RSpec is not an option.

#34 - 07/09/2019 02:53 PM - Eregon (Benoit Daloze)

I think we should listen to RSpec users here like joallard (Jonathan Allard) and would welcome more users to reply on this thread.
People who maintain software defining it probably have a different view than their users.
If RSpec users understand it well, then I think that addresses the "potential confusion" point for RSpec.

#35 - 07/09/2019 07:57 PM - ivoanjo (Ivo Anjo)

As a user of both Kotlin and RSpec, here's my 0.01 cents: I've been using Kotlin for almost 1.5 years, and the implicit it is really nice shortcut, without making the code too harder to understand. Even newcomers seem to get it pretty quickly. I'd definitely use it if I had it in Ruby (and I do miss it when I switch between Ruby and Kotlin).

On the other hand the coincidence with RSpec's it DSL is rather unfortunate. Would be better if this could be made a keyword, but that's clearly not an option, neither for it nor for other options on this thread.

Would it be reasonable to perhaps emit a warning when it is used naked in a scope where it is defined as a method that can be called with zero args? That way we could slowly push away any remaining confusing issues, and perhaps the RSpec maintainers may consider dropping zero args it in a future 4.x release?

#36 - 07/11/2019 01:46 PM - shevegen (Robert A. Heiler)

Just a very brief comment since it was discussed in the last developer meeting, solely on the
syntax issue of %1, %2, versus @1, @2 and :1, :2:

To me personally, %1, %2 is almost the same as @1, @2. I'd still prefer @1 @2 etc... but I
don't have a huge problem with %1, although I find @1 more elegant than %1, purely syntax-wise.

The idea for :1, :2 on the other hand, though, looks somewhat strange to me.

It reminds me more of a symbol than @1 reminds me of an instance variable, oddly enough.

If @1 would not be considered to be good from a syntax point (although I find it just
fine), then I think %1 would be better than :1.

To those who would like to have a look at the developer meeting log, the summary was
provided by mame (I think) here:

https://docs.google.com/document/d/1K61SGIwp8_rNsPyhmayUcERu71vt_etDjXdhqrLmBVY/edit#

#37 - 07/30/2019 04:08 AM - ko1 (Koichi Sasada)

I'm against on this proposal because of compatibility issue.
I think current workaround introduces new confusion.

Files

its.patch	4.92 KB	06/04/2019	mame (Yusuke Endoh)
mame_its_proposal.patch	5.26 KB	07/02/2019	Benoit_Tigeot (Benoit Tigeot)