Ruby - Feature #6201

do_something then return :special_case (include "then" operator)

03/26/2012 10:34 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Status:	Rejected
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	3.0

Description

=begin
I've read several aproaches to deal with this case and this just feels like Ruby doesn't have a good idiom yet for the common use case.

You want to do some special action under certain conditions and then return the method.

I've once proposed some way to return from the callee:

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/41681

But I agree with other that found that approach a dangerous one.

One of the common idioms I see is writing code like this:

render action: NOT_FOUND and return

The problem with this is that it won't work if render returns nil or false. And render return value isn't supposed to have any meaning.

So while reading some articles this issue got back to me, but this time I think I found a more elegant solution for improving the Ruby language in a way to better support this pretty common use case.

If you read the article below, you'll see the suggested code by its author, Avdi Grimm:

http://devblog.avdi.org/2012/03/23/unless-readable-else-confused/

return follow(response['location']) if response.redirect?

I also prefer this style, but I don't like this other common approach to the same problem: to include a meaningless return in the front of the special case statement.

What if you really want to return something else?

So, I'd like Ruby to support "then" besides "and" and "or" so that they will be used for what they were intended for:

save_file and return :saved # you only want to return :saved if save_file returned true

special case, it wasn't possible to save the file

notify_error_by_email or raise EmailNotWorking
handle_error

But for the example in the mentioned article, I'd prefer to write code like this:

follow response['location'] if response.redirect? then return :redirected

or

follow response['location'] then return :redirected if response.redirect?

I'm not sure about what precedence "then" should have.

It means: no matter the return value, I want this to be run after that method.

Another approach, since this would be used for earlier exit, would be to have something like then_return instead (a bit more constraint, but still useful):

render :redirected then_return if should_be_redirected?

Any of those approaches would enable better readable code than the current alternatives.

Related issues:

Has duplicate Ruby - Feature #6222: Use ++ to connect statements	**Rejected**	**03/29/2012**

History

#1 - 03/26/2012 10:51 AM - nobu (Nobuyoshi Nakada)

- Description updated

=begin
(follow response['location']; return :redirected) if response.redirect?
=end

#2 - 03/26/2012 10:52 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Feedback

#3 - 03/26/2012 11:14 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

nobu (Nobuyoshi Nakada) wrote:

> =begin
> (follow response['location']; return :redirected) if response.redirect?
> =end

Yes, Nobu, that is the other common idiom (the one I actually use), but I still don't find this readable.

#4 - 03/26/2012 09:21 PM - mame (Yusuke Endoh)

- Target version changed from 2.0.0 to 3.0

- Status changed from Feedback to Assigned

Hello, Rodrigo

In short, you are proposing a syntactic sugar:

A then B

as

(A; B)

, so that you can rewrite the following idiom:

(render(...); return) if condition?

with:

render(...) then return if condition?

, right?

I think it will cause compatibility issue.

```
if A then
B
C
end
```

will be parsed as:

```
if (A; B)
C
end
```

So, 2.0 cannot include your idea. I mark this ticket as 3.0.

Personally, I like your idea very much.
I often hesitate to write *whopping* four lines:

if C
A
B
end

when A, B and C are all short and simple.
I feel like defeat when I use a semicolon ;-)

In addition, it is cool to use "then" which is one of
the least useful keywords ("for", "alias", ...) in Ruby.

So, I'm sorry that I can't accept your idea in 2.0.

BTW, when you write a proposal, I recommend you put a short
example first to demonstrate your idea.
It will not only be attractive to more people, but also make
our ticket management task easier :-)

Thanks,

--
Yusuke Endoh mame@tsg.ne.jp

#5 - 03/26/2012 09:36 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Thank you for your feedback Yusuke. And yes, you understood it right. I actually tried to quickly find out first if "if ... then" was a valid syntax in Ruby, but I couldn't find it. It seems I didn't look hard enough...

I think my proposal was very clear in the subject/title (the example), but I think I can still edit it so that the example comes first in the description too. What do you think about this?

And I agree with you, if we're gonna use "then" and it is already an existent keyword, 2.0.0 can't be the target version :(

#6 - 03/30/2012 04:25 PM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

Both "then" and "++" could compatibility problem. Your "a then b" or "a ++ b" can be implemented by "(a; b)" without any addition to the language.

Matz.

#7 - 04/10/2012 01:32 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Sorry if I'm being boring here. Just let me know and I'll never ask about this again.

My understanding is that adding a new keyword to Ruby will break backward-compatibility and so it should be avoided at most.

In the other side, 'then' is not an option because it would conflict with if-then syntax.

My understanding is that "then" is totally optional in "if" statements, and it wouldn't make any difference if we just removed it.

So I thought that maybe we could deprecate "then" in "if" statements for Ruby 2.0, emiting warnings for it, while keeping it a reserved word.

After that we could include the proposed 'then' syntax in Ruby 3.0.

If you really don't want a syntax sugar for (a;b) just let me know and I'll stop asking about this.

Again, sorry if I'm bothering you.

#8 - 04/10/2012 02:12 AM - Eregon (Benoit Daloze)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> My understanding is that adding a new keyword to Ruby will break backward-compatibility and so it should be avoided at most.
>
> In the other side, 'then' is not an option because it would conflict with if-then syntax.
>
> My understanding is that "then" is totally optional in "if" statements, and it wouldn't make any difference if we just removed it.

"then" can also be used in case statements.

> So I thought that maybe we could deprecate "then" in "if" statements for Ruby 2.0, emiting warnings for it, while keeping it a reserved word.
>
> After that we could include the proposed 'then' syntax in Ruby 3.0.

That sounds like a really good way to confuse users ;)

I think it isn't worth adding a new keyword for such a specific case. I'm more than happy with the multi-line way. Returning inside a method is something that should be clear as it modifies the control flow quite heavily.

#9 - 04/10/2012 02:18 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

You're right, Benoit, I completely forgot about the case statements.

Just forget about this then. By the way, did you like the pun? ;)

#10 - 04/10/2012 03:45 AM - Eregon (Benoit Daloze)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> Just forget about this then. By the way, did you like the pun? ;)

Yes, and the "." match well ";" if you see what I mean in my return statement.

#11 - 04/10/2012 07:05 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Eregon (Benoit Daloze) wrote:

> Yes, and the "." match well ";" if you see what I mean in my return statement.

Sorry, I didn't :D I guess I'm too tired to get Yusuke's joke on #6265 and your statement :) Or maybe I'm just old enough :D