Ruby - Feature #16451

Special ternary operator for methods ending in `?`

12/25/2019 03:46 PM - myxoh (Nicolas Klein)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

When a method ends in ?, we should be able to skip the ? in the ternary operator. For example, we should be able to do:

```
question_method? true_statement : false_statement
```

This shouldn't interfere with implementations, as it currently fails to parse. Below are examples:

```
def normal_method; true end
def question_method?; false end

question_method? ? 'was true' : 'was false' # => 'was false'
question_method?   'was true' : 'was false' # => 'was false'
normal_method ?    'was true' : 'was false' # => 'was true'
```

History

#1 - 12/25/2019 04:37 PM - shevegen (Robert A. Heiler)

I am slightly against this proposal.

To me personally it does not make a lot of sense to differ between the trailing character of a method in regards to the ternary operator. IMO the ternary operator should not discriminate against last characters of any method, ever. It would also add a bit more complexity and I am not sure if the trade-off is worth to be had here.

Another problem I see is the prolific use of '?'. It would be better, in my opinion, to keep the syntax simple(r), if possible. With many special syntax and combinations of syntax in use, it becomes harder to determine what is going on in general.

Ultimately you only have to convince matz about the use case and proposed benefit of a suggestion, though, not me. :)

(I should also add that I use the trailing token '?' for methods a LOT, but I avoid using the ternary operator, largely because I have found that it takes me longer to determine what is going on, as opposed to e. g. if/else checks, even though they require more lines of code.)

#2 - 12/25/2019 05:13 PM - sawa (Tsuyoshi Sawada)

- Subject changed from Special terniary operators for methods ending in `?` to Special ternary operator for methods ending in `?`

- Description updated

#3 - 12/25/2019 05:18 PM - sawa (Tsuyoshi Sawada)

The current ternary operator does not work that way. It works like this:

```
question_method? ? 'was true' : 'was false' # => 'was false'
```

#4 - 12/25/2019 10:25 PM - myxoh (Nicolas Klein)

- Description updated

#5 - 12/25/2019 10:26 PM - myxoh (Nicolas Klein)

sawa (Tsuyoshi Sawada) wrote:

> The current ternary operator does not work that way. It works like this:
>
> ```
> question_method? ? 'was true' : 'was false' # => 'was false'
> ```

You are right, I've corrected the example

#6 - 12/25/2019 10:36 PM - myxoh (Nicolas Klein)

shevegen (Robert A. Heiler) wrote:

> I am slightly against this proposal.
>
> To me personally it does not make a lot of sense to differ between the
> trailing character of a method in regards to the ternary operator. IMO
> the ternary operator should not discriminate against last characters
> of any method, ever. It would also add a bit more complexity and I
> am not sure if the trade-off is worth to be had here.
>
> Another problem I see is the prolific use of '?'. It would be better,
> in my opinion, to keep the syntax simple(r), if possible. With many
> special syntax and combinations of syntax in use, it becomes harder
> to determine what is going on in general.
>
> Ultimately you only have to convince matz about the use case and
> proposed benefit of a suggestion, though, not me. :)
>
> (I should also add that I use the trailing token '?' for methods a
> LOT, but I avoid using the ternary operator, largely because I have
> found that it takes me longer to determine what is going on,
> as opposed to e. g. if/else checks, even though they require more
> lines of code.)

While I'm not going to argue that if else end statements read nicer than the ternary statement. Reality is that the ternary terms to be the preferred option by many rubyist for one-liners. Case in point: [https://github.com/rubocop-hq/ruby-style-guide] prefers the ternary operator)

There is a strong precedent of ruby making an exceptional case for the case of readability. Case in point: Rocket Hash syntax vs modern hash syntax

My reasoning for why the exception is guaranteed is that
1- It is best practice to use the ternary operator on actual true/false rather than truthy, falsey statements
2- It is best practice in ruby to define methods returning booleans with a trailing ?
3- For many projects, it is best practice to use ternary operators on one liners.

The corollary is that things like
variable.nil? ? 'something' : 'something else'
Happen a lot

And I think it doesn't read as nice. Which is one thing we tend to value.

I definitely understand the point against it (having an exception on the ternary operator)

#7 - 12/26/2019 08:09 AM - nobu (Nobuyoshi Nakada)

A method ends with ? also can take argument(s), so the proposed syntax introduces an ambiguity which doesn't seem solvable.

#8 - 12/26/2019 10:15 AM - myxoh (Nicolas Klein)

nobu (Nobuyoshi Nakada) wrote:

> A method ends with ? also can take argument(s), so the proposed syntax introduces an ambiguity which doesn't seem solvable.

I won't claim to be an expert in how the interpreter works, so I'll take your word for it.
I'm not sure why that would be the case, here are some examples as to how I'd expect this to work

```
foo.include?(object) included_statement : none_included_statement
foo.include? object included_statement : none_included_statement
foo.include? included_statement : none_included_statement
#=> raises ArgumentError (wrong number of arguments expected 1 given 0)
# Assuming the method bar? has optional arguments
foo.bar? bar_is_true : bar_is_false # foo.bar? is executed with no optional arguments
foo.bar? true bar_is_true : bar_is_false # foo.bar? is executed with `true` as the first optional param
foo.bar? true, nil, third(false) bar_is_true : bar_is_false
# foo.bar? is executed with arguments `(true, nil, third(false))`
```

Alternatively, I think a reasonable compromise is to raise an syntax error if the arguments are not surrounded by bracks (like we do if you call method submethod argument rather than method submethod(argument)

Or even raise a syntax error if the question method uses arguments with this shortened syntax

#9 - 12/26/2019 10:32 AM - xBartu (Bartu Demirkıran)

I am also against the proposal.

Having said that I love using one-liners but newcomers to Ruby (from another language) will presumably look for current syntax.

This might be a bit odd yet we can advocate the following:

```
(question_method?) ? 'was true' : 'was false'
```

#10 - 12/26/2019 09:03 PM - Dan0042 (Daniel DeLorme)

Also against.

I agree that the double question mark looks weird in foo? ? 1 : 2, but that's just a code formatting issue and doesn't warrant such an exceptional syntax. xBartu's suggestion is good, or you could customize your editor's syntax highlighting to de-emphasize the second question mark.