Ruby - Feature #20498

Negated method calls

05/19/2024 10:52 PM - MaxLap (Maxime Lapointe)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

I want to propose the following syntax: foo.!bar. I know it's already valid syntax, but please read on for details.

When someone write a somewhat long line of code that is negated, the main way I've seen of doing it is:

```
must_create_user = !User.where(somelong: :condition, even_more: "thing").exists?
```

I personally highly dislike it, as I must keep the "not" in the back of my mind as I read the line. When quickly reading a line like this, it's super easy to misread and understand the opposite result.

The current ways around this I can think of are:

- rename the variable (can be annoying)
- Use unless (only possible when in a condition; some people, like me, have a hard time grapsping a unless)
- use a .!in the end (foo.exists?.!), I've never seen that and it looks ugly to me (this is subjective).
- create a new method name with the negated meaning (not always possible)

My proposal would look like this:

```
must_create_user = User.where(somelong: :condition, even_more: "thing").!exists?
```

You cannot forget the bang that you saw 15 words ago, it's right there.

It also basically reads as English: "user where ... doesn't exists".

The main argument against this I can think of is that it's technically already a valid syntax. I believe it's frowned upon to override the bang operator and I'm not aware of places where it is overridden to with also having a parameter.

I made a prototype in RubyNext, which you can try here: https://ruby-next.github.io/#gist:0e133bf6f27f2437193dc034d58083dc

Clarification: the prototype is not perfect and does not handle foo&.!empty?. In that case, if foo is nil, the result of the expression would be nil.

Related issues:

Related to Ruby - Feature #12075: some container#nonempty?	**Feedback**

History

#1 - 05/20/2024 03:23 AM - duerst (Martin Dürst)

I think defining an explicit method for this purpose (on Object, I guess) would be better, because it wouldn't add complications to the syntax. The name could be not or invert or `negate or some such. Your result would look like:

```
must_create_user = User.where(somelong: :condition, even_more: "thing").exists.not
```

BTW, different languages use different places (start, middle, end, several places) for the negation. (Japanese puts it at the end.)

#2 - 05/20/2024 04:24 AM - nobu (Nobuyoshi Nakada)

- Related to Feature #12075: some container#nonempty? added

#3 - 05/20/2024 04:36 AM - nobu (Nobuyoshi Nakada)

In your prototype, foo . ! exist? was transpiled to !(foo . exist?).
Your proposal is not a new operator, but a syntax sugar?

#4 - 05/20/2024 05:57 AM - akr (Akira Tanaka)

We can use foo.!.

```
must_create_user = User.where(somelong: :condition, even_more: "thing").exists?.!
```

#5 - 05/20/2024 08:04 AM - ufuk (Ufuk Kayserilioglu)

I wonder how wild it would be to make ! accept an optional block. That way we could write:

```
must_create_user = User.where(somelong: :condition, even_more: "thing").!(&:exists?)
```

which very close to what the original poster wants, and is a much smaller method change instead of a syntax change.

#6 - 05/20/2024 08:42 AM - hmdne (hmdne -)

I saw such a proposal before and I thought of some syntax and implementation, but I didn't submit that in the previous issue:

```
class MethodNegator < BasicObject
  def initialize(obj)
    @obj = obj
  end

  def method_missing(method, ...)
    @obj.public_send(method, ...)
  end

  def respond_to_missing?(include_all = false)
    @obj.respond_to?(include_all)
  end
end

module Kernel
  def non(sym=nil)
    if sym
      proc { |*args,**kwargs,&block| !sym.to_proc.call(*args,**kwargs,&block) }
    else
      MethodNegator.new(self)
    end
  end
end

p "".non.empty?
# => true
p ["", "a", "b"].select(&non(:empty?))
# => ["a", "b"]
```

#7 - 05/20/2024 01:02 PM - MaxLap (Maxime Lapointe)

Thanks for the feedback. I updated the gist to have an example with arguments in the call:

```
puts A.new.!exists?(with_friends: true, skip_administrator: true)
```

This highlights some problems with the alternatives:

- Calls at the end (such as .! or .not) are very easy to miss if there are arguments given to the method. The longer the args, the most likely to be missed:

```
puts A.new.exists?(with_friends: true, skip_administrator: true).!
puts A.new.exists?(with_friends: true, skip_administrator: true).not
```

- Using a no block (ex: foo.!(&:exists?)) as ufuk (Ufuk Kayserilioglu) suggested is no longer clean looking. The extra syntax ends up almost hiding the !:

```
foo.!{ _1.exists?(with_friends: true, skip_administrator: true) }
```

About p "".non.empty?: I personally dislike this pattern because there is a clear performance cost to it (extra method call & creating an object). As a result, whenever I would have to use it, I would wonder if the cleaner looking code is worth it knowing it's wasteful at the same time. (I know it's not much, but it's enough for me to ask myself each time, I dislike Rails' .not.where pattern since it could have simply been not_where or where_not.)

@nobu (Nobuyoshi Nakada): I'd say it's both syntax sugar and an operator, just like &..

#8 - 05/20/2024 01:26 PM - marcandre (Marc-Andre Lafortune)

MaxLap (Maxime Lapointe) wrote in #note-7:

> @nobu (Nobuyoshi Nakada): I'd say it's both syntax sugar and an operator, just like &..

Seems like pure syntax sugar.

That's not to say it's not a good idea. It's difficult to say if the added convenience is worth it versus the added cognitive load required / added difficulty to learn the language.

#9 - 05/20/2024 01:33 PM - MaxLap (Maxime Lapointe)

- Description updated

#10 - 05/20/2024 01:36 PM - MaxLap (Maxime Lapointe)

- Description updated

#11 - 05/20/2024 01:41 PM - MaxLap (Maxime Lapointe)

For foo&.!empty?, the result would be nil if foo is nil. This is not handled by the prototype.

The alternatives mentionned so far look like this:

foo&.empty?&.!
foo&.non&.empty?
foo&.empty?&.not
foo&.!(&:empty?)

Which I hope no one writes □□

#12 - 01/05/2025 06:45 AM - rubyFeedback (robert heiler)

MaxLap wrote:

> Which I hope no one writes

For instance, at:

```
foo&.empty?&.!
```

I would not write that myself because I do not use "&.". I understand that other people use it but to me it feels at odds with other syntax of ruby (which is of course very subjective; for similar reasons I don't use -> for lambda.)

But the "I hope no one writes like that" part is, I think, a bit too optimistic to assume. People writing ruby code use the whole range of flexibility ruby offers, including writing code that is (in my opinion) somewhat ugly or alien-looking. Sometimes it is super-creative, such as zverok's use, but the style of code is kind of orthogonal to how I would write ruby code. I am more in the "simple OOP camp" in regards to ruby - object.foobar1.foobar2 and so forth. (Although I tend to start with modules first, before I add classes; usually in my projects I have a base class called base.rb, Foobar::Base, which includes features based on key-modules.)

In regards to the proposal here: I think it is an interesting proposal. While I myself most likely would not write code like that, if I understood the suggestion correctly then it is kind of like using "unless", without using "unless", and focusing on the trailing part of a method call, similar to using ".foobar" versus ".foobar?". And I use the latter a LOT; I think that was a great idea by matz to have conditional queries also end with '?' as a possibility. I use that many times in the ruby code I write. So, from this point of view, I can somewhat understand the proposal, e. g. ".!exists?". I am not sure I like it, as my brain also kind of has to process e. g. both '!' and '?', but at the least I understand it. ".exists?.not" is a bit different to that, so I think it is not a full replacement. I understand duerst's reasoning, but I think if someone has the use case of just using "!" as a "flip-operator" then using "!" here would make more sense than using ".not". I use "!" a lot to flip logic. So to me personally the main question is more whether my brain would be able to deal with both "!" and "?", but as said, I am not really the target audience anyway; I often do the logic grouping not in one line like that, but use "if x ... new line ... do something there ... new line ... add end"; or unless. And I also use "!", so I am fine with the status quo.