# Ruby trunk - Feature #13395

## Add a method to check for not nil

04/01/2017 08:24 PM - JustJosh (Joshua Stowers)

| | |
|---|---|
| **Status:** | Open |
| **Priority:** | Normal |
| **Assignee:** | |
| **Target version:** | |

**Description**

There does not seem to be a method in Ruby to check if an object is *not* nil.
Such a method could help with readability.

Example:

```
array = [1, 'dog', nil]
array.count(&:not_nil?)
```

vs

```
array = [1, 'dog', nil]
array.reject(&:nil?).count
```

**Related issues:**

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

### History

#### #1 - 04/03/2017 01:25 AM - sawa (Tsuyoshi Sawada)

Many methods that take a block come in positive-negative pairs, unlike count. I am not sure how frequent the use case is, but if it is, I rather claim that there should be a negative version of count.

#### #2 - 04/03/2017 01:54 AM - duerst (Martin Dürst)

Any ideas about the name?

#### #3 - 04/03/2017 07:52 AM - Hanmac (Hans Mackowiak)

```
array.count(&:itself)
```

does work for something **similar** like that.

its not 100% what you might want, because it does ignore false too

#### #4 - 04/03/2017 12:45 PM - nobu (Nobuyoshi Nakada)

Hanmac (Hans Mackowiak) wrote:

> ```
> array.count(&:itself)
> ```
>
> does work for something **similar** like that.

IIRC, it equals to simple array.count.

#### #5 - 04/03/2017 02:19 PM - Hanmac (Hans Mackowiak)

nobu (Nobuyoshi Nakada) wrote:

> Hanmac (Hans Mackowiak) wrote:
>
> > ```
> > array.count(&:itself)
> > ```
> >
> > does work for something **similar** like that.

IIRC, it equals to simple array.count.

are you sure? i am currently on ruby 2.3.3 Windows

```
[1, "bc", nil].count #=> 3
[1, "bc", nil].count(&:itself) #=> 2
```

#### #6 - 04/03/2017 02:28 PM - nobu (Nobuyoshi Nakada)

Thank you for the correction.

#### #7 - 04/04/2017 05:57 PM - ogarci5 (Oliver Garcia)

What about as a condition for if statements? For example:
Case 1

```
if !object.nil?
  # Do something
end
```

Case 2

```
if object
  # Do something
end
```

Case 3

```
if object.not_nil?
  # Do something
end
```

I end up using Case 2 a lot because it reads better than Case 1. However this doesn't work if object can be false in which case Case 3 would be the most readable.

#### #8 - 04/04/2017 11:41 PM - darix (Marcus Rückert)

```
if !object.nil?
  # Do something
end
```

```
unless object.nil?
  # Do something
end
```

#### #9 - 04/08/2017 08:02 PM - shevegen (Robert A. Heiler)

I think that these discussions come up with some frequency; if I recall correctly, Tsuyoshi Sawada was proposing something that I agreed with in principle. But I did point out that the english language appears to have it easier with positive assertions rather than negative ones.

I am not against .not_nil? per se, mind you, since it may be symmetrical for .nil? cases and the example give by darix (if nil, if !nil?, unless nil?) but I think it is something that is somewhat showing a limitation of the english language as such.

I actually found that the simplest way, for my own code, is to try to formulate things "positively" whenever possible.

So:

```
if condition
end

if condition1 and condition2
end
```

A similar explanation I use for .select or .reject, I almost always go to pick exactly what I need to have. Like a filter where you apply the filtering in a

forward fashion (you could of course always revert the
filter, like to use .reject rather than .select, or
within the block clause, invert the checks).

To the suggestion itself in regards to .count, perhaps
this may be worthwhile to have some special meaning for
what a user may want to count for.

In non-legal ruby, consider this:

```
array = [1, 'dog', nil]
array.count(! &:nil?)
```

Granted, not very readable. :)

Then again, I also consider the & not really readable either.

How about:

```
array = [1, 'dog', nil]
array.count(:not_nil)
```

or

```
array.count(:not_nil)
```

Hmm...

To be honest, I don't think that these examples are really that
great.

In the above example, using .compact may be simpler:

```
array.compact.size
```

For the latter, perhaps a method that does it, like .not_nil? but
I am not sure if this is used that much to warrant an addition.

I do somewhat agree with ogarci5 by the way - not necessarily because
of the explanation, but because in case 3 he gave, you do not have
to use "unless" and neither the invert "operator" "!", which is
usually much easier and more straightforward. So in that context,
I actually agree, having that flexibility may be a good thing,
even if I don't like the name .not_nil? a lot.

I still think that it is a limitation of the english language.

Consider a backpack in a RPG/rogue-like game. You want to
query whether it is empty or filled:

```
if backpack.empty? # Seems simple.

if !backpack.empty? # Seems ok although more difficult to understand

if backpack.filled? # Hmmm... filled with what?

if backpack.not_empty? # Not ideal but perhaps also better than the other two examples before
```

So I kinda semi-agree with ogarci5; I still think that the english language
itself is the one that has the biggest problems here with negations, followed
by the human brain modeling concepts. (OOP is a modeled concept too after all)

#### #10 - 04/09/2017 01:19 AM - kernigh (George Koehler)

Because ! is a method, one can also write !object.nil? as object.nil?.! using the method chain .nil?.! to check for not nil. This works since Ruby 1.9, if I recall.

#### #11 - 04/12/2017 12:30 PM - nobu (Nobuyoshi Nakada)

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

#### #12 - 08/31/2017 05:41 AM - matz (Yukihiro Matsumoto)

Your example is bit weak. Is there any **realistic** use-case for the method?

Matz.

#### #13 - 09/30/2017 05:44 AM - znz (Kazuhiro NISHIYAMA)

I think Array#nitems is like this. But it was removed since ruby 1.9.1.

In doc/NEWS-1.9.1:

> o Array#nitems was removed (use count {|i| !i.nil?})

#### #14 - 09/30/2017 12:53 PM - MSP-Greg (Greg L)

I believe the english equivalent to not_nil would be exist (instead of exists, to follow prev use). I believe core only uses exist? with Dir, File, and FileTest.

Having it would allow showing the standard case first in an if-else-end structure, given that unless-else-end may be considered 'strange' code.

Also, I'm sure many of us have used a tri-state nil/false/true variable when it is helpful.