Ruby master - Feature #7978

boolean to_i

02/27/2013 11:00 PM - alexeymuranov (Alexey Muranov)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	2.6	

Description

=begin
The current behavior is the following:

```
nil.to_i
=> 0
false.to_i
NoMethodError: undefined method `to_i' for false:FalseClass

true.to_i
NoMethodError: undefined method `to_i' for true:TrueClass
```

This does not look very consistent to me. I think it could be useful to define (({false.to_i})) as 0 and (({true.to_i})) as 1. I think those are fairly common numeric values for False and True. These values as strings "0" and "1" are also commonly used in HTML forms to represent boolean values.
=end

History

#1 - 02/27/2013 11:02 PM - alexeymuranov (Alexey Muranov)

Right now i am creating my own helper method int_from_bool to be used in an HTML form :).

Well, probably i will go just for bool ? 1 : 0 for now.

#2 - 02/27/2013 11:45 PM - Eregon (Benoit Daloze)

I agree true/false#to_i would be nice and used it on some occasions.

#3 - 02/28/2013 04:06 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

- Assignee set to matz (Yukihiro Matsumoto)

Ruby is not C. 0 is not false. false is not 0.
nil has its role as a default value, true/false are not.

Matz.

#4 - 02/28/2013 05:10 AM - alexeymuranov (Alexey Muranov)

matz (Yukihiro Matsumoto) wrote:

> Ruby is not C. 0 is not false. false is not 0.

In some respects they are similar, and 0 and 1 are often used in relation to logical operations. For example, the characteristic function of a set P is defined as

$\chi_P(x) = 1$ if $x \in P$ is true
$\chi_P(x) = 0$ if $x \in P$ is false

The simplest boolean algebra consists of 0 and 1 with the usual algebraic operations mod 2.

Update. Of course, defining #to_int on booleans would be highly inappropriate, but i only suggested #to_i. "0" is not the 0, but "0".to_i works.

#5 - 02/28/2013 05:59 AM - funny_falcon (Yura Sokolov)

Well, yes: ruby is not C and false is not 0. But why false could not be converted to 0 by #to_i ?

27.02.2013 23:07 пользователь "matz (Yukihiro Matsumoto)" <matz@ruby-lang.org> написал:

> Issue #7978 has been updated by matz (Yukihiro Matsumoto).
>
> Status changed from Open to Rejected
> Assignee set to matz (Yukihiro Matsumoto)
>
> Ruby is not C. 0 is not false. false is not 0.
> nil has its role as a default value, true/false are not.
>
> Matz.
>
> ---
>
> Feature #7978: boolean to_i
> https://bugs.ruby-lang.org/issues/7978#change-37162
>
> Author: alexeymuranov (Alexey Muranov)
> Status: Rejected
> Priority: Normal
> Assignee: matz (Yukihiro Matsumoto)
> Category: core
> Target version: next minor
>
> =begin
> The current behavior is the following:
>
> ```
> nil.to_i
> => 0
> false.to_i
> NoMethodError: undefined method `to_i' for false:FalseClass
>
> true.to_i
> NoMethodError: undefined method `to_i' for true:TrueClass
> ```
>
> This does not look very consistent to me. I think it could be useful to define ({false.to_i}) as 0 and ({true.to_i}) as 1. I think those are fairly common numeric values for False and True. These values as strings "0" and "1" are also commonly used in HTML forms to represent boolean values.
> =end
>
> --
> http://bugs.ruby-lang.org/

#6 - 02/28/2013 06:59 AM - phluid61 (Matthew Kerwin)

funny_falcon (Yura Sokolov) wrote:

> Well, yes: ruby is not C and false is not 0. But why false could not be converted to 0 by #to_i ?

That seems to imply that the reverse should hold, but ({!!0 => true}).

Similarly, why should true.to_i return 1, and not -1 (as in Visual Basic) or 43 or 0 (which is also a truthy value)?

It makes sense that if such to- (and maybe from-) conversions are required, they should be bundled in a Module that explicitly defines the mappings.

#7 - 02/28/2013 08:21 AM - alexeymuranov (Alexey Muranov)

phluid61 (Matthew Kerwin) wrote:

> funny_falcon (Yura Sokolov) wrote:
>
> > Well, yes: ruby is not C and false is not 0. But why false could not be converted to 0 by #to_i ?

That seems to imply that the reverse should hold, but (({!!0 => true})).

1. I do not think there is a rule that such fuzzy typecasting in Ruby has to be invertible:

"".to_i.to_s # => "0"

1. !!x is not one of Ruby type casting methods. If Ruby had a function Boolean(x), it would be natural to define it as !!x, but each class would need to define its own typecasting method. In my opinion, 0.to_b would have to be false.

   Similarly, why should true.to_i return 1, and not -1 (as in Visual Basic) or 43 or 0 (which is also a truthy value)?

1. 1 is simpler than 43 or -1. This is the usual convention of boolean algebra that 0 is false and 1 is true. The logical AND becomes simply the multiplication (mod 2). If it is false that you have some object, you have 0 of it. :)

#8 - 02/28/2013 02:29 PM - Anonymous

Dne 27.2.2013 15:00, alexeymuranov (Alexey Muranov) napsal(a):

> Issue #7978 has been reported by alexeymuranov (Alexey Muranov).
>
> ---
>
> Feature #7978: boolean to_i
> https://bugs.ruby-lang.org/issues/7978
>
> Author: alexeymuranov (Alexey Muranov)
> Status: Open
> Priority: Normal
> Assignee:
> Category: core
> Target version: next minor
>
> =begin
> The current behavior is the following:
>
> ```
> nil.to_i
> => 0
> false.to_i
> NoMethodError: undefined method `to_i' for false:FalseClass
>
> true.to_i
> NoMethodError: undefined method `to_i' for true:TrueClass
> ```
>
> This does not look very consistent to me. I think it could be useful to define (({false.to_i})) as 0 and (({true.to_i})) as 1. I think those are fairly common numeric values for False and True. These values as strings "0" and "1" are also commonly used in HTML forms to represent boolean values.
> =end

Actually I am wondering the opposite, why nil.to_i is possible? It does
not make any sense to me.

Vít

#9 - 02/28/2013 06:51 PM - alexeymuranov (Alexey Muranov)

phluid61 (Matthew Kerwin) wrote:

> That seems to imply that the reverse should hold, but (({!!0 => true})).

Then what about

!!false.to_s # => true ?

There is no reason for fuzzy typecasting to preserve true-ish-ness.

#10 - 03/02/2013 01:48 AM - alexeymuranov (Alexey Muranov)

phluid61 (Matthew Kerwin) wrote:

> Similarly, why should true.to_i return 1, and not -1 (as in Visual Basic) or 43 or 0 (which is also a truthy value)?

If some event is going to happen with probability 1, it is almost surely true that it will happen. If it is going to happen with probability 0, it is almost surely false that it will happen.

#11 - 03/14/2013 04:49 PM - alexeymuranov (Alexey Muranov)

I understand however that there can be a problem if one wants, for example, to use true, false, nil for ternary logic http://en.wikipedia.org/wiki/Three-valued_logic .

#12 - 03/14/2013 04:55 PM - matz (Yukihiro Matsumoto)

What kind of problem do you imagine? I cannot think of any.

Matz.

#13 - 03/14/2013 04:59 PM - alexeymuranov (Alexey Muranov)

I mean, my suggestion can cause problems if one uses true, false, nil for ternary logic, because nil.to_i and false.to_i would be both 0. According to Wikipedia, in ternary logic context, it is common to represent false by -1.