Ruby master - Feature #11390

Allow symbols starting with numbers

07/22/2015 07:32 PM - v0dro (Sameer Deshmukh)

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

Description

Currently it is not possible to create a symbol that looks like :1twothree.

Converting to a string and then symbolizing causes hash lookup problems and proves counter-intuitive. What's also surprising is that ruby allows symbols to start with special characters but not numbers.

History

#1 - 07/22/2015 09:13 PM - 0x0dea (D.E. Akers)

Permitting such syntax would needlessly complicate the parser for virtually no gain, and potentially break existing code that uses a numerical literal as the alternative of a non-padded ternary expression (foo?1:2). Just use :'1twothree'.

#2 - 07/23/2015 05:14 PM - v0dro (Sameer Deshmukh)

But it goes against the principle of least surprise that ruby follows throughout the language. It's counter-intuitive and IMO from a layman ruby programmer's point of view the solution you propose does not confirm to the elegance of the syntax.

#3 - 07/30/2015 09:57 AM - duerst (Martin Dürst)

- Assignee set to matz (Yukihiro Matsumoto)

Sameer Deshmukh wrote:

> But it goes against the principle of least surprise that ruby follows throughout the language. It's counter-intuitive and IMO from a layman ruby programmer's point of view the solution you propose does not confirm to the elegance of the syntax.

It's virtually impossible to make any language, even Ruby, unsurprising in all cases for all people. I agree with D.E. Akers that being able to write "foo?1:2" (as compared to that resulting in a syntax error, and forcing people to write "foo?1: 2" or so) is more important overall than to be able to write :1twothree.

I have assigned this issue to Matz so that he can reject it.

#4 - 08/03/2015 05:07 PM - shevegen (Robert A. Heiler)

I have no particular pro or con on the suggestion itself, but I
did want to comment on one part:

> But it goes against the principle of least surprise that
> ruby follows throughout the language.

As far as I can tell, there is not really a general "principle of least
surprise" - that one was coined by Dave, if I remember correctly, the
Pickaxe author that also helped popularize Ruby.

While matz has had his story to share about the complexity of C++, and
the use of orthogonal functionality, the principle of least surprise is
primarily about how matz designed ruby, not as a "general principle of
least surprise" applicable to everyone and everywhere all the time (because
people have different opinions, different backgrounds, different expectations
and so forth, so what may be surprising to some people, may be quite
logical to others).

A good example is:

if File.exists? '/tmp/foo.txt'

versus

if File.exist? '/tmp/foo.txt'

Matz allowed the first alias at one point, because in written english
this one is correct "if the file exists, do something". But this is
not how one should ask the question in the Ruby way, one should ask:

"object, do you exist?

And then the second variant suddenly becomes the more consistent /
logical one.

So the principle of least surprise is not a general one applicable
all the time, in all situations. It will be specific to the example
at hand.

Perhaps it should be called more an overall strive towards simplicity,
elegance and consistency rather than complexity, but people will
still disagree on where more complexity may be required and when
it may not be. Someone has to decide on the features after all.

C++ gained new functionality too since 2010.

http://www.codeproject.com/Articles/570638/Ten-Cplusplus-Features-Every-Cplusplus-Developer

> It's counter-intuitive and IMO from a layman ruby programmer's
> point of view the solution you propose does not confirm to the
> elegance of the syntax.

Ruby does not always have elegant syntax. I found global variables
not really elegant, for instance. I think Ruby is, in many ways,
a cleaned-up, OOP-centric "sister" language to the old perl family,
with features added to it that were found to be useful, and
consistent - e. g. the usage of blocks. They give you additional
flexibility when you require it.

There are other syntax elements such as -> rather than lambda.

Now I don't use lambdas much at all myself, but I always found
the usage of -> really strange in my own code. It just does not
seem to "fit" stylistically with the rest of the code that I
write, so I opted to not use -> in my libraries.

My favourite interview from matz is still this one here, I think
it is still applicable even 12 years lateron:

http://www.artima.com/intv/rubyP.html

Variables that start with numbers are often a bit strange, just
see this:

$123456789123 = 42

foo.rb:1: Can't set variable $-1097262461

Peculiar error message - I was not using the variable -1097262461. :)

PS: By the way, other people actually propose to do away with the
distinction between Symbols and Strings altogether, so your suggestion
to allow more flexibility in Symbols, could be changed altogether
when we say that we should not use Symbols at all. ;)

I am also neutral on the Symbol versus String debate, and it's not
one to have for Ruby 2.x era, perhaps Ruby 3.x era, but these guys
also have a point - see the strange HashWithIndifferentAccess,
which arises only because one has to separate between symbols and
string as keys. It might simplify the language if one could always
use strings or string-like objects (and the conversion be handled
internally - I understand that this is not easy to change since
Symbols are internal anyway, but I myself used to wonder whether
I should use a symbol as key or a string as key in the past).

#5 - 08/14/2015 05:23 AM - gwelch (Grant Welch)

Sameer Deshmukh wrote:

> Currently it is not possible to create a symbol that looks like :1twothree.
>
> Converting to a string and then symbolizing causes hash lookup problems and proves counter-intuitive. What's also surprising is that ruby allows symbols to start with special characters but not numbers.

The non-quoted symbol format follows the same naming rules as variables (local, instance, class, global), methods, and constants (Modules, Classes, and Constants). Otherwise, you'll have to resort to the quoted string format.

Also, notice that:

```
:foo == :'foo' # => true
```

Variable and Method Naming Rules: http://ruby-doc.org/core-2.2.2/doc/syntax/assignment_rdoc.html