Ruby trunk - Feature #15612

A construct to restrict the scope of local variables

02/19/2019 12:42 PM - sawa (Tsuyoshi Sawada)

Status:	Feedback	
Priority:	Normal	
Assignee:		
Target version:		

Description

We sometimes have local variables that are to be used only to keep track of some temporal states/values during a short routine:

```
...
foo = some_initial_value
some_routine_that_uses_foo
...
```

Currently, the scope of local variables are either a proc, a block, loop body, a method definition, or a class/module definition, but such routines are sometimes just only a part of them.

In order to improve readability of the code by explicitly indicating the scope of such local variables, and to avoid pollution by the variable, I propose to have some construct to restrict the scope of local variables.

One possibility, without adding a new keyword to the current syntax, is to use the begin...end construct. The expected behavior would be:

```
begin
  foo = "foo"
  foo # => "foo"
end
foo # => `nil`, or "Undefined local variable or method error"

foo = "bar"
begin
  foo = "foo"
  foo # => "foo"
end
foo # => "bar"
```

Or, does this break the existing code too much? If so, can a new construct be added to the current syntax?

History

#1 - 02/19/2019 01:28 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

I am not sure how readability improved by adding explicit scoping. So you have to prove it first.
From my point of view, overriding local variables in the example is a source of confusion, rather than readability.

Matz.

#2 - 02/19/2019 02:38 PM - shevegen (Robert A. Heiler)

I think this is an interesting idea but the syntax is a bit confusing (to me).

This may be because I am so used to:

```
begin
  code_that_may_break
rescue Duck
  puts 'all ducklings were rescued.'
end
```

It seems a bit surprising to see "begin" used with the intent to

modify/restrict/extend local variables specifically.

Personally I do not need many local variables for most of my methods.
The longer a method becomes, the harder it is to modify it (at the
least when I try to).

It may be better to try another construct than begin/end, if only
for illustration purpose (it may read better with an alternative
to begin/end, but I can not think of a good alternative myself).

#3 - 02/19/2019 03:15 PM - jeremyevans0 (Jeremy Evans)

sawa (Tsuyoshi Sawada) wrote:

> In order to improve readability of the code by explicitly indicating the scope of such local variables, and to avoid pollution by the variable, I propose to have some construct to restrict the scope of local variables.
>
> One possibility, without adding a new keyword to the current syntax, is to use the begin...end construct. The expected behavior would be:
>
> ```
> begin
>   foo = "foo"
>   foo # => "foo"
> end
> foo # => `nil`, or "Undefined local variable or method error"
> ```

This would definitely break existing code. There is a lot of code that expects to be able to access local variables first assigned inside a begin/end block after the begin/end block.

As blocks already do what you want, why not just:

```
tap do
  foo = "foo"
  foo # => "foo"
end
foo # => NameError

foo = "bar"
tap do
  foo = "foo"
  foo # => "foo"
end
foo # => "foo"
```

You can substitute another method that yields once for tap if you want.

#4 - 02/19/2019 10:39 PM - duerst (Martin Dürst)

On top of what others have said, methods in Ruby should normally be quite short. If you have a method that's so long that you think you need restricted scopes for local variables, you should look at how to split the code into several methods.

#5 - 02/20/2019 01:55 AM - nobu (Nobuyoshi Nakada)

jeremyevans0 (Jeremy Evans) wrote:

> As blocks already do what you want, why not just:
>
> ```
> foo = "bar"
> tap do
>   foo = "foo"
>   foo # => "foo"
> end
> foo # => "foo"
> ```

You can "declare" overriding local variables.

```
foo = "bar"
tap do |;foo|
  foo = "foo"
  foo # => "foo"
end
foo # => "bar"
```