Ruby master - Feature #15352

Mandatory block parameters

11/28/2018 01:29 AM - gfx (Goro FUJI)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

There are too many block parameter assertions (it's kind of idioms: raise NoBlockGiven unless glock_given?).

It's very useful if there's a syntax to declare mandatory block parameters, such as:

```
def foo(&!block)
  block.call
end

foo() # raises ArgumentError "in `foo`: no block given"
```

History

#1 - 11/28/2018 11:05 AM - shevegen (Robert A. Heiler)

This is an interesting suggestion in the sense that block parameters, or rather, blocks as a whole, are a bit like additional arguments to every method in ruby, at all times, optional; but they are also quite special entities in ruby. For example, to use .call on a proc; is a bit similar to yield, and unbinding/rebinding a method and passing this as a block to another method.

Typically if we have a method such as foo(), in ruby we can call it via:

```
foo()
foo
foo { :something }
```

All three ways are more or less the same, except for the last which pushes more information onto the method foo; but that information may not be evaluated within the method body (of foo).

I think this is where matz should come in since he designed ruby and added blocks, so he knows best how blocks are seen (from within ruby itself). I could see it go either way with statements, where ruby users may say that block parameters should behave like regular parameters of methods; or that blocks are special and so block parameters should also be special. I myself have no real preference either way. I think backwards compatibility and backwards behaviour may be a reason in retaining the present behaviour, though; but again, I am neutral on the feature suggestion in itself. We can also reason that this would allow for more functionality, since he would gain the ability to cause blocks to raise an error, in this case an ArgumentError.

There is, however had, one part I slightly dislike, and that is the syntax. I don't have an alternative suggestion, which is bad, but I don't like the close associoated of &! there. Perhaps my opinion is partially affected by other unrelated suggestions, e. g. to add arguments to &: invocation styles, such as &.: or variants that use even more characters. But anyway, I guess the syntax is partially separate from the suggested functionality, so I think it may be best to ask matz about the principle situation first, that is whether block arguments should/could behave like regular methods too (specifically whether they should be able to raise ArgumentError, if a ruby user wants to have it that way).

One last point; not sure if they are worth mentioning but I will mention it, if only for symmetry:

- In regular parameters signature we have something like:

  def foo(bar = '')
  def foo(bar)

In the second case we must supply an argument; in the first, the default will be to an empty String ''. I understand that this can not be the same for blocks, at the least not without changing how they behave, but I wanted to point this out because to me this is quite different in behaviour and syntax, from the proposed &! or any similar proposal - I guess it all has to be different to the two method definitions above.

Anyway, I wanted to write a bit more, but I tend to write too much so I will stop here.

#2 - 11/28/2018 12:19 PM - sawa (Tsuyoshi Sawada)

I don't find this feature useful. If you wanted to raise an error when no block is given, all you have to do is call yield within the method body, which will not be an extra code if you are going to use the block somewhere in the method body.

```
def foo
  ... yield ...
end

foo
#  >> `foo': no block given (yield) (LocalJumpError)
```

And in case you want to return an error with a different message, then that is when you want to implement your custom validation clause in your code like the ones found in the examples that you have searched.