Ruby trunk - Feature #9777

Feature Proposal: Proc#to_lambda

04/26/2014 06:02 PM - schneems (Richard Schneeman)

Status:	Feedback	
Priority:	Normal	
Assignee:		
Target version:		

Description

Currently different block objects such as a lambda can be converted into to a proc: http://www.ruby-doc.org/core-1.9.3/Proc.html#method-i-to_proc

However you cannot turn a Proc instance into a lambda. Since a Proc and lambda behave differently sometimes you may want to convert between the two functionalities. One example is a return inside of the block. In a lambda the return keyword exits the closure, in a Proc the return keyword raises an exception.

There is currently no implementation standard way to convert a Proc to a lambda. I made a gem that makes this easier: https://github.com/schneems/proc_to_lambda but it seems overkill.

If MRI introduces a to_lambda method on Proc then we can standardize on an interface for this behavior. This question on stack overflow has been upvoted many times: http://stackoverflow.com/questions/2946603/ruby-convert-proc-to-lambda. I think other Ruby developers would like this behavior supported by Ruby core.

History

#1 - 04/27/2014 01:06 AM - nobu (Nobuyoshi Nakada)

- Description updated

Richard Schneeman wrote:

> Currently different block objects such as a lambda can be converted into to a proc:
> http://www.ruby-doc.org/core-1.9.3/Proc.html#method-i-to_proc

It changes nothing, but returns self as-is, without any effects.

#2 - 04/27/2014 01:14 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Feedback

Why can't you use break and next?

#3 - 04/28/2014 05:55 PM - schneems (Richard Schneeman)

Sorry for the delayed response. I was not "watching" this issue by default.

> Why can't you use break and next?

I wanted this to make a public api based on anonymous functions: https://github.com/opro/opro#password-token-exchange (note the return used in this section towards the bottom). New developers know the semantics of return very well but the differences between lambda and Proc are confusing to them, they are not familiar with the break keyword. If i tell new developers to use this API, create a block, and use break many would not do so and still use return by accident. This is my use-case, but others may have different or better ones. As mentioned 16 people were interested in this enough to google it, find the stack overflow post and up-vote the answer.

#4 - 04/29/2014 12:36 AM - nobu (Nobuyoshi Nakada)

return in a proc exits the defined method, unless it has returned already.
Turning a proc into a lambda disappoints that expectation.
It doesn't feel a good idea.

#5 - 04/29/2014 04:20 PM - avit (Andrew Vit)

Would it work to just wrap it inside a lambda to get the semantics you want?

#6 - 04/29/2014 07:02 PM - schneems (Richard Schneeman)

Andrew Vit wrote:

> Would it work to just wrap it inside a lambda to get the semantics you want?

Like lambda &proc? That would be fine. As Nobu mentioned Proc and lambda behave differently, sometimes I want control over the behavior of my program so I want the ability to change the object I am using from proc to lambda or lambda to proc.

When I made the original request I did not know that lambda#to_proc was basically a no-op. I thought it actually changed the behavior.

#7 - 04/29/2014 07:10 PM - avit (Andrew Vit)

I find that using throw is useful in the situation you describe. It gives a nice explicit message about what's going: it's like a multi-level return to the point where you want to catch it.

#8 - 05/06/2014 05:17 PM - schneems (Richard Schneeman)

Another possible reason to convert a Proc to a lambda is to for raising error on arguments

```
foo = -> { puts "hello"  }
foo.call(1)
ArgumentError: wrong number of arguments (1 for 0)
  from (irb):4:in `block in irb_binding'
  from (irb):5:in `call'
  from (irb):5
  from /Users/schneems/.rubies/ruby-2.1.1/bin/irb:11:in `<main>'

bar = Proc.new { puts "hello" }
bar.call(1)
# => "hello"
```

If a user passes in a proc there will be no errors or warnings that the proc was improperly declared. Converting to a lambda would change the behavior to then raise an error.

#9 - 05/07/2014 12:08 AM - nobu (Nobuyoshi Nakada)

If a user wanted to write it as a proc, then it means that he/she doesn't want it to raise error, doesn't it?

#10 - 05/08/2014 12:28 AM - kernigh (George Koehler)

Beware! The answer on Stack Overflow (http://stackoverflow.com/a/2946734) is wrong, because it does not preserve the value of self in the block.

This is better, but still wrong:

```
def convert_to_lambda &block
  obj = block.binding.eval('self')
  Module.new do
    define_method(:_, &block)
    return instance_method(:_).bind(obj).to_proc
  end
end
```

It preserves self in the block, but it fails when I pass the lambda to #instance_exec or #module_exec. I will try to post explanation to Stack Overflow.

There is no way to convert proc to lambda, unless Ruby adds a new feature.

I have no reason to support this new feature. I am not wanting libraries to convert my blocks to lambdas. I know how to use next. If someone converts my block to a lambda, then my block can no longer return from the enclosing method, or ignore extra arguments. If I want my block to be a lambda, I can write &->, as in

```
3.times &->(_) { puts "It works!" }
```

#11 - 06/21/2014 07:27 AM - myronmarston (Myron Marston)

We have a need for this feature in RSpec. Specifically, for some of the new features in RSpec 3, we use Method#parameters and Proc#parameters. However, for procs, parameters does not distinguish between args with defaults and args without:

```
irb(main):001:0> Proc.new { |a| }.parameters
=> [[:opt, :a]]
irb(main):002:0> Proc.new { |a=1| }.parameters
=> [[:opt, :a]]
```

I understand why, because with procs you can call it without providing the args so all args are optional and it's essentailly like |a=nil|. Lambdas, on the other hand, do distinguish:

```
irb(main):003:0> lambda { |a| }.parameters
=> [[:req, :a]]
irb(main):004:0> lambda { |a=1| }.parameters
=> [[:opt, :a]]
```

I attempted to use some code like @schneem's gem to convert a proc to a lambda for the purpose of looking at the parameters (but, importantly, not to call the lambda). Unfortunately, we ran into a ruby bug (#9967) and had to back it out.

It would be nice for an officially supported solution to convert a proc to a lambda. I can understand the issues with calling a converted lambda, but simply using the converted lambda to get more precise info about the block parameters seems more reasonable and safe.

#12 - 01/05/2018 09:00 PM - naruse (Yui NARUSE)

- Target version deleted (2.2.0)