Ruby master - Feature #11161

Proc/Method#rcurry working like curry but in reverse order

05/19/2015 12:41 PM - Hanmac (Hans Mackowiak)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

currenty with curry you can only replace elements in order
#rcurry should be added to be able to return the last parameter first.

```
def abc(a,b); "a=#{a}, b=#{b}"; end
c= method(:abc).curry

c[1,2] #=> "a=1, b=2"
c[1][2] #=> "a=1, b=2"
```

i image rcurry to be like that:

```
def abc(a,b); "a=#{a}, b=#{b}"; end
c= method(:abc).rcurry(2)

c[1,2] #=> "a=2, b=1"
c[1][2] #=> "a=2, b=1"
```

because of optional parameters, rcurry might be only be used when giving the arity

History

#1 - 01/24/2018 09:11 AM - zverok (Victor Shepelev)

Started to write my own ticket, but found this one :)

My explanation was like this:

Considering two widespread Ruby patterns:

- Last method arguments are frequently options/settings/clarifications of the call;
- Method#to_proc as a useful idiom for function-style programming...

I thought about constructs like this (some hate them, but not me):

```
URLS.
  map(&Faraday.method(:get)).
  map(&JSON.method(:parse))
```

For me, it seems pretty useful to be able to do this:

```
URLS.
  map(&Faraday.method(:get).rcurry[some_get_param: 'value']).
  map(&JSON.method(:parse).rcurry[symbolize_names: true])
```

Of course, you can say that it makes you eyes bleed.... But I somehow love what we can get this way.

Can we please consider it for next Ruby version?

#2 - 05/17/2018 06:39 AM - nobu (Nobuyoshi Nakada)

- Description updated

#3 - 05/17/2018 06:47 AM - akr (Akira Tanaka)

- Status changed from Open to Rejected

We discussed this issue at DevelopersMeeting20180517Japan [Misc #14698].

It needs better (practical) usages to show usefulness of this proposal.

#4 - 05/17/2018 09:49 AM - zverok (Victor Shepelev)

> It needs better (practical) usages to show usefulness of this proposal.

akr (Akira Tanaka) I believe the code in my comment provides the justification:

```
URLS.
  map(&Faraday.method(:get).rcurry[some_get_param: 'value']).
  map(&JSON.method(:parse).rcurry[symbolize_names: true])
```

In general, the idea is: a lot of methods have "options" as their last argument(s), and first argument(s) is the main "subject" of the method.
So, when the method is converted to proc, you can "bind" those options with rcurry, and then pass it, like in my example above, as a 1-argument proc with some options set.

The alternative, BTW, would be not currying last arguments, but currying any keyword argument by name.

#5 - 05/18/2018 03:27 AM - shyouhei (Shyouhei Urabe)

zverok (Victor Shepelev) wrote:

> > It needs better (practical) usages to show usefulness of this proposal.
>
> akr (Akira Tanaka) I believe the code in my comment provides the justification:
>
> ```
> URLS.
>   map(&Faraday.method(:get).rcurry[some_get_param: 'value']).
>   map(&JSON.method(:parse).rcurry[symbolize_names: true])
> ```
>
> In general, the idea is: a lot of methods have "options" as their last argument(s), and first argument(s) is the main "subject" of the method.
> So, when the method is converted to proc, you can "bind" those options with rcurry, and then pass it, like in my example above, as a 1-argument proc with some options set.
>
> The alternative, BTW, would be not currying last arguments, but currying any keyword argument by name.

I then think this might be called something other than currying-related name. Keyword arguments are not directly FP-oriented concepts. If you want to make keyword arguments better name it as such; if you want functional programming better we need another example that is more FP.