Ruby trunk - Feature #14609

`Kernel#p` without args shows the receiver

03/16/2018 01:29 AM - ko1 (Koichi Sasada)

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

Description

Abstract

Kernel#p(obj) prints obj as inspected.
How about to show the receiver if an argument is not given?

Background

We recently introduce yield_self which encourages block chain.

https://zverok.github.io/blog/2018-01-24-yield_self.html
Quoted from this article, we can write method chain with blocks:

```
construct_url
  .yield_self { |url| Faraday.get(url) }.body
  .yield_self { |response| JSON.parse(response) }
  .dig('object', 'id')
  .yield_self { |id| id || '<undefined>' }
  .yield_self { |id| "server:#{id}" }
```

There is a small problem at debugging.
If we want to see the middle values in method/block chain, we need to insert tap{|e| p e}.

With above example,

```
construct_url
  .yield_self { |url| Faraday.get(url) }.body
  .yield_self { |response| JSON.parse(response) }.tap{|e| p e} # debug print
  .dig('object', 'id')
  .yield_self { |id| id || '<undefined>' }.tap{|e| p e} # debug print
  .yield_self { |id| "server:#{id}" }
```

Proposal

obj.p shows same as p(obj).

We can replace
block{...}.tap{|e| p e}
to
block{...}.p

For above example, we can simply add .p at the end of line:

```
construct_url
  .yield_self { |url| Faraday.get(url) }.body
  .yield_self { |response| JSON.parse(response) }.p # debug print
  .dig('object', 'id')
  .yield_self { |id| id || '<undefined>' }.p # debug print
  .yield_self { |id| "server:#{id}" }
```

Compatibility issue

(1) Shorthand for nil

This spec change can introduce compatibility issue because p returns nil and do not output anything.
That is to say, p is shorthand of nil. Some code-golfers use it.

Maybe we can ignore them :p

(2) make public method

Kernel#p is private method, so if we typo obj.x to obj.p (not sure how it is feasible), it will be NoMethodError because of visibility.
We need to change this behavior.

Note

Past proposal and discussion

Endoh-san proposed same idea 10+ years ago [ruby-dev:29736] in Japanese.
I think we should revisit this idea because of yield_self introduction.

At this thread, Matz said "simple p shows p(self), it is not clear".

[ruby-dev:30903]

```
  p
```

はどう動くのかとか (`p self`と同じ、は変な気が)

```
  self.p(obj)
```

はどうなのかとか。その辺が解決 (納得) できたら、ということで。

English translation:

```
What the behavior of (I feel strange that it is similar to `p(self)`):

  p

What happen on

  self.p(obj)
```

pp

If this proposal is accepted, we also need to change pp behavior.

gems

tapp method is provided by gem.
https://github.com/esminc/tapp

I'd thought to propose this method into core. But I found that p is more shorter than tapp.
Disadvantage is p is too short and difficult to grep.

Related issues:

Related to Ruby trunk - Feature #15112: Introducing the short form of `STDERR...	**Open**

History

#1 - 03/16/2018 03:11 AM - mame (Yusuke Endoh)

+1

Kernel#p is one of the greatest feature in Ruby. It would be further great to make it useful.

#2 - 03/16/2018 08:00 AM - zverok (Victor Shepelev)

Small notice: If #13581 would be once acted upon, attaching p in the middle of the chain could be as simple as (using one of the proposed syntaxes)

```
  .yield_self { |response| JSON.parse(response) }.tap(&:p)
  .dig('object', 'id')
  .yield_self { |id| id || '<undefined>' }.tap(&:p)
```

Just .p is still nicer, though.

#3 - 03/16/2018 10:55 AM - Hanmac (Hans Mackowiak)

hm i have a slightly problem with this

check out the different return types there:

```
a = []
p *a #=> nil

a = [1]
p *a #=> 1

a = [1,2]
p *a #=> [1,2]
```

#4 - 08/10/2018 07:03 AM - ko1 (Koichi Sasada)

Hanmac: do you rely on this behavior?

#5 - 08/10/2018 01:09 PM - Hanmac (Hans Mackowiak)

ko1 (Koichi Sasada) wrote:

> Hanmac: do you rely on this behavior?

Me? not so much, in general i don't trust the return value of print commands (print returns null)

it was months ago, i probably wanted to point out that the return value of p might be differ depending on the parameters
It might be inconsistency?

There is Object.display too but i don't use it much because it doesn't have new lines

#6 - 08/11/2018 01:49 AM - osyo (manga osyo)

Good proposal.
This can solve the following problem.

```
# NG: syntax error, unexpected ':', expecting '}'
p { name: "homu", age: 14 }

# OK:
{ name: "homu", age: 14 }.p
# => {:name=>"homu", :age=>14}
```

#7 - 08/11/2018 03:29 AM - nobu (Nobuyoshi Nakada)

osyo (manga osyo) wrote:

> Good proposal.
> This can solve the following problem.
>
> ```
> # NG: syntax error, unexpected ':', expecting '}'
> p { name: "homu", age: 14 }
> ```

You can use parentheses.

```
p(name: "home", age: 14)
```

#8 - 08/18/2018 05:59 PM - shevegen (Robert A. Heiler)

I don't mind, personally. To me, the biggest improvement was
that we could omit doing:

```
require 'pp'
```

;)

Since I did that a lot in my code. (I love pp; I think I use it more than just p)

I personally have not been using (or needing) yield_self or tap so far, so the change would probably not be of immediate benefit to me; but probably also not require of me to change anything either.

To the name "tapp" - that name is a bit weird. To me it reads as if we combine "tap" and then add a "p" to it. Reminds me of a joke proposal to condense "end" into "enddd" and such. :D

To be fair, I consider the name yield_self to be also weird :D - but matz added an alias called "then" to it if I understand it correctly (though the semantic confuses me a bit as well ... but I don't really want to distract here since I don't really feel too strongly either way; picking good names is not always easy).

On a side note, perhaps in the long run we could have something to "experiment" with - like new or changed features in ruby that have not been 100% approved in the sense of a name AND the associated functionality, so we can try them out for some time, which may help build a stronger opinion either way. (I mean this in general, not just in regards to #p here).

It may still be best to ask matz again though. Syntax shortcuts (syntactic sugar) has always been an area in ruby where code changes has happened (e. g. yield_self to then, or omitting the end value for infinite ranges and so forth).

#9 - 09/13/2018 06:04 AM - nobu (Nobuyoshi Nakada)

How about:

```
self.P
```

:P

#10 - 09/13/2018 06:16 AM - matz (Yukihiro Matsumoto)

I vote for #tapp.

Matz.

#11 - 10/10/2018 05:30 AM - mrkn (Kenta Murata)

- Related to Feature #15112: Introducing the short form of `STDERR.puts expr.inspect`. added

#12 - 10/13/2018 01:57 PM - docx (Lukas Dolezal)

Allowing debugging within yield_self is great! If yield_self was inspired by Elixir, we can look at Elixir for inspiration here as well:

https://hexdocs.pm/elixir/IO.html#inspect/2

One of the examples that is very interesting is that they have optional parameter to describe the printout:

```
[1, 2, 3]
|> IO.inspect(label: "before")
|> Enum.map(&(&1 * 2))
|> IO.inspect(label: "after")
|> Enum.sum
```

Prints:

```
before: [1, 2, 3]
after: [2, 4, 6]
```

I'm wondering, would that be something we would like in Ruby too?

The problem with the proposal of making obj.p is that the same method already is defined with parameter in the form of p(obj).

Hence it would be impossible to implement obj.p("after") - as that would conflict with the current version of p with parameter.

My feeling is that given that both yield_self and tap are defined on Object, it does not feel to me right that .p should be defined on Kernel. I can see the advantage of already existing method there, but I feel like it would be wrong overloading.

Continuing on the argument that yield_self is defined on Object, should we define this "print inspect and return self" method be defined on Object too?

If so, we could make new method p_self (terrible name yes) and implement it with new interface when called as instance method and avoid making Kernel.p public:

```
Object.p_self() - p self and return self
Object.p_self(label:) - puts "#{label}: #{self.inspect}" and return self
```

What do you think?

#13 - 10/16/2018 11:13 AM - zverok (Victor Shepelev)

docx (Lukas Dolezal) That are pretty interesting comparison with Elixir!
In fact, what the examples show (to me) is a need for more powerful partial application of procs. E.g. what would be a parity for Elixir is something like...

```
def p_with_label(label, obj)
  puts "#{label}#{obj.inspect}"
end

[1, 2, 3]
  .tap(&method(:p_with_label).curry['before: '])
  .map { |x| x * 2 }
  .tap(&method(:p_with_label).curry['after: '])
  .sum
```

This example already works, yet doesn't look VERY expressive, to be honest. I see two pretty different solutions to the problem:

First is **core**. Considering the shorten of &method(:name) is discussed for ages (and almost agreed to be &:.name), it could be discussed on addition of more powerful partial application than current (mostly unusable) Proc#curry to language core. The result may look somewhat like (fantasizing)...

```
[1, 2, 3]
  .tap(&:.p.with('before: '))
  .map { |x| x * 2 }
  .tap(&:.p.with('after: '))
  .sum
```

The second one is **rise of proc-producing libraries:**

```
module Functional
  module_function
  def inspect(label:)
    ->(obj) { puts "#{label}#{obj.inspect}" }
  end
end

[1, 2, 3]
  .tap(&Functional.inspect(label: 'before: '))
  .map { |x| x * 2 }
  .tap(&Functional.inspect(label: 'after: '))
  .sum
```

#14 - 10/16/2018 01:10 PM - nobu (Nobuyoshi Nakada)

zverok (Victor Shepelev) wrote:

> First is **core**. Considering the shorten of &method(:name) is discussed for ages (and almost agreed to be &:.name),

You may want to say &self.:name at https://bugs.ruby-lang.org/issues/13581#change-72072?
It wouldn't be able to omit the receiver, syntactically.

#15 - 10/16/2018 01:18 PM - zverok (Victor Shepelev)

> It wouldn't be able to omit the receiver, syntactically.

Hmm, I missed this point is the discussion. This limitation makes :. syntax sugar substantially less useful :(

#16 - 10/17/2018 04:21 PM - ko1 (Koichi Sasada)

How about obj.p! or obj.pp! ?

#17 - 12/10/2018 07:08 AM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#18 - 03/11/2019 04:50 AM - akr (Akira Tanaka)

matz (Yukihiro Matsumoto) wrote:

> I vote for #tapp.

I don't like #tappp (for pp self).

#19 - 03/11/2019 04:54 AM - nobu (Nobuyoshi Nakada)

obj.?

#20 - 03/11/2019 04:59 AM - ko1 (Koichi Sasada)

Discussion:

- obj.p has compatibility issue (some code expect p returns nil) -> reject
- p! is not danger. -> reject
- tapp seems long. tappp is weird
- tap_p, tap_pp are too long.
- introduce obj.? new method name -> don't modify syntax! (by matz). it is not predicate.

No conclusion.