Ruby master - Feature #16341

Proposal: Set#to_proc and Hash#to_proc

11/11/2019 03:02 PM - Nondv (Dmitry Non)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

```
class Set
  def to_proc
    -> (x) { include?(x) } # or method(:include?).to_proc
  end
end
```

Usage:

```
require 'set'

banned_numbers = Set[0, 5, 7, 9]
(1..10).reject(&banned_numbers) # ===> [1, 2, 3, 4, 6, 8, 10]
```

UPD

also for hash:

```
class Hash
  def to_proc
    ->(key) { self[key] }
  end
end

dogs = ['Lucky', 'Tramp', 'Lady']
favourite_food = { 'Lucky' => 'salmon', 'Tramp' => 'pasta', 'Lady' => 'pasta' }

food_to_order = dogs.map(&favourite_food)
```

History

#1 - 11/11/2019 03:05 PM - Nondv (Dmitry Non)

- Backport deleted (2.5: UNKNOWN, 2.6: UNKNOWN)

- Tracker changed from Bug to Feature

#2 - 11/11/2019 03:08 PM - zverok (Victor Shepelev)

Since 2.5, Set implements #===, so you can just:

```
(1..10).grep_v(banned_numbers)
# => [1, 2, 3, 4, 6, 8, 10]
```

which is pretty clear and probably more effective than proc conversion.

#3 - 11/11/2019 03:18 PM - Nondv (Dmitry Non)

Well, to_proc allows to send objects as blocks which can be quite useful not just in case of select/reject. Also, probably, those two are used more often than grep/grep_v.

Another example from the top of my head is count:

```
dogs = Set[:labrador, :husky, :bullterrier, :corgi]
pets = [:parrot, :labrador, :goldfish, :husky, :labrador, :turtle]
pets.count(&:dogs) # ===> 3
```

#4 - 11/11/2019 03:26 PM - zverok (Victor Shepelev)

Fair enough. ...well, you still can

```
pets.count(&dogs.:include?)
```

until the core team haven't reverted it :))))

(Which, for me, is more clear than value-objects-suddenly-becoming-procs, but apparently it is only me)

#5 - 11/11/2019 03:37 PM - Nondv (Dmitry Non)

Well, to be fair, this change is just nice-to-have sugar. I don't expect it to become a thing.

I guess for now the best way to do that is:

```
pets.count { |x| dogs.include?(x) }
# or
pets.count(&dogs.method(:include?))
```

They both are "more clear than value-object-suddenly-becoming-procs". But having implicit conversion would be just a nice feature to make code more compact and expressive (MHO).
Clojure treats sets as functions, btw:

```
(def dogs #{:labrador :husky :bullterrier :corgi})

(count (filter dogs [:parrot :labrador :goldfish :husky :labrador :turtle]))
```

#6 - 11/11/2019 11:43 PM - shevegen (Robert A. Heiler)

> this change is just nice-to-have sugar. I don't expect it to become a thing.

The ruby core team often points out that having good use cases may help a proposal; and of course avoiding other problems such as backwards-incompatibility or such.

Your initial comment is quite sparse, so zverok sort of got you to explain more lateron. ;)

I am not really using ruby in a functional-centric manner nor do I know clojure (aside from superficial glances), but to me personally I am not completely sure if the use case has been explained. Unless it was only syntactic sugar of course.

#7 - 11/12/2019 06:58 PM - Nondv (Dmitry Non)

This *is* a syntactic sugar. Using & + to_proc in this case is the same (not technically, but algorithmically, I guess) as providing an explicit block something.some_method { |x| some_set.include?(x) }

I don't find it crucial in any way and, to be honest, I don't really use sets that much (I prefer using hashes directly). But this feature could make some code a tiny bit easier to read from English language perspective (I *think*)

#8 - 11/12/2019 07:05 PM - Nondv (Dmitry Non)

Speaking of hashes, they could implement implicit proc conversion as well:

```
class Hash
  def to_proc
    ->(key) { self[key] }
  end
end

dogs = ['Lucky', 'Tramp', 'Lady']
favourite_food = { 'Lucky' => 'salmon', 'Tramp' => 'pasta', 'Lady' => 'pasta' }

food_to_order = dogs.map(&favourite_food)
```

#9 - 11/12/2019 07:09 PM - Nondv (Dmitry Non)

The main problem is that implicit conversion can be confusing, especially, if it's not obvious what the resulting proc is going to do.

However, I think that hashes are being used *mainly* for making key-value pairs and accessing them and sets are being used for checking if something is included.

So usage of :[] and :include? seems appropriate and relatively straight-forward to me.

Of course, depending on the context. With map/reduce/count it does make sense indeed but maybe there're cases when it can make things hard to understand

#10 - 11/12/2019 07:10 PM - Nondv (Dmitry Non)

- Subject changed from Proposal: Set#to_proc to Proposal: Set#to_proc and Hash#to_proc

#11 - 11/12/2019 07:11 PM - Nondv (Dmitry Non)

- Description updated

#12 - 11/12/2019 07:44 PM - shan (Shannon Skipper)

Nondv (Dmitry Non) wrote:

> Speaking of hashes, they could implement implicit proc conversion as well:
>
> ```
> class Hash
>   def to_proc
>     ->(key) { self[key] }
>   end
> end
>
> dogs = ['Lucky', 'Tramp', 'Lady']
> favourite_food = { 'Lucky' => 'salmon', 'Tramp' => 'pasta', 'Lady' => 'pasta' }
>
> food_to_order = dogs.map(&favourite_food)
> ```

This already works! Hash#to_proc was added in Ruby 2.3. ruby-core:11653

I like the idea of Set#to_proc.

#13 - 11/12/2019 08:12 PM - Nondv (Dmitry Non)

shan (Shannon Skipper) wrote:

> This already works!

I can't believe I'm so oblivious :D