Ruby master - Feature #18369

users.detect(:name, "Dorian") as shorthand for users.detect { |user| user.name == "Dorian" }

11/30/2021 12:40 PM - dorianmariefr (Dorian Marié)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Hi,

I was thinking I often do things like collection.detect { |item| item.attribute == value } and a shorthand like collection.detect(:attribute, value) would be quite useful

What do you think?

And I know there is collection.detect { _1attribute == value } but I try not to use _1 and this syntax would be shorter and simpler

Could also apply to other methods like all? (collection.all?(:attribute, value)), and basically any Enumerable method

https://www.rubydoc.info/stdlib/core/Enumerable

**History**

#1 - 11/30/2021 12:40 PM - dorianmariefr (Dorian Marié)
- Backport deleted (2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN)
- Tracker changed from Bug to Feature

#2 - 11/30/2021 12:40 PM - dorianmariefr (Dorian Marié)
Could also be users.detect(&:name, "Dorian")

#3 - 11/30/2021 01:53 PM - Dan0042 (Daniel DeLorme)
So you try not to use _1 ... just out of curiosity, would you use this?

`collection.detect { .attribute == value }`

#4 - 11/30/2021 01:55 PM - dorianmariefr (Dorian Marié)
Dan0042 (Daniel DeLorme) wrote in #note-3:

So you try not to use _1 ... just out of curiosity, would you use this?

`collection.detect { .attribute == value }`

I don't think so, still not explicit what is going on, and there is the overhead of new syntax

#5 - 11/30/2021 02:21 PM - sawa (Tsuyoshi Sawada)
I think that is too specific to be a part of Ruby core. I don't think this feature would be accepted.

I think you can define a Proc constructor method for yourself like the following:

```ruby
def attreq! k, v
  Proc.new { |_1| send(k) == v}
end
```

Then, you can do:

```ruby
class A
  attr_reader :foo, :bar
  def initialize foo, bar
    @foo, @bar = foo, bar
  end
end

collection = [A.new(1, 2), A.new(3, 4), A.new(5, 6)]
```

03/09/2022 1/3
collection.detect(&attreql(:foo, 3)) # => #<A:0x00007fb751064630 @foo=3, @bar=4>
collection.all?(attreql(:bar, 7)) # => false

The strength of doing it like this compared to your proposal is that it is more flexible. You can do:

def attrlt k, v
  Proc.new{|l.send(k) < v|}
end

collection.detect(&attrlt(:foo, 3)) # => #<A:0x00007fd3ab8a4680 @foo=1, @bar=2>
collection.all?(&attrlt(:bar, 7)) # => true

#6 - 11/30/2021 02:28 PM - dorianmariefr (Dorian Marié)
Maybe the feature would be to be possible to have arguments after a block, e.g.

def detect(&block, value)
  User.all.detect { |user| block.call(user) == value }
end
detect(&:first_name, "Dorian")

#7 - 12/02/2021 06:00 AM - nobu (Nobuyoshi Nakada)
Since Enumerable#detect or Enumerable#find has the argument for the different purpose, I think that the extension in this way is not acceptable and should be a separate method.
The "arguments after a block" is one of rejected ideas before the numbered parameters.

#8 - 12/02/2021 01:52 PM - cvss (Kirill Vechera)
It's a good occasion to use the composition of Proc/Method objects:

collection.detect(&:first_name.to_proc>>"Dorian".method(==))

If we had a shorthand operator for Object#method (#12125), it would look nicer:

collection.detect(&:first_name.to_proc>>"Dorian").==)

And if we make a shorthand Symbol#>> for the composition of a Symbol and a Proc, it would look even wonderful:

class Symbol
  def >> b
    to_proc >> b
  end
end

collection.detect(&:first_name>>"Dorian").==)

When you are frequently using such constructions you can read it easily, but it is definitely more confusing comparing to the old plain variant:

collection.detect(_1.first_name == "Dorian")

#9 - 12/02/2021 04:16 PM - sawa (Tsuyoshi Sawada)
cvss (Kirill Vechera) wrote in #note-6:

It's a good occasion to use the composition of Proc/Method objects:

collection.detect(&:first_name.to_proc>>"Dorian".method(==))

Your trick forces the use of Yoda conditions, which may be tricky and cryptic.

#10 - 12/02/2021 05:58 PM - baweaver (Brandon Weaver)
Pattern Matching may make a very interesting tie-in here for a short-hand:

# Struct provides built-in pattern matching abilities
Person = Struct.new(:first_name, :last_name, :age)

jim = Person.new("Jim", "Smith", 30)
jill = Person.new("Jill", "Smith", 20)
sue = Person.new("Sue", "Smith", 40)
people = [jim, jill, sue]

# Currently works
people.select { _i in { first_name: /^J/, age: 18.. } }

# Potential 1: bare keywords
people.select { _i in first_name: /^J/, age: 18.. }

# Potential 2: 'in' shorthand
people.select(&in first_name: /^J/, age: 18..)

Generally I think 1 is doable, 2 is stretching, though it would be nice to have syntax that allows to shorten one-line matchers for predicates where they would be commonly used.

#11 - 12/02/2021 07:41 PM - zverok (Victor Shepelev)

it would be nice to have syntax that allows to shorten one-line matchers for predicates where they would be commonly used

TBH, since pattern matching inception I hope for some way of putting patterns into values—to store them in constants, and, in that case, simple grep will do (if that value would respond to #=== which it should!):

MY_PATTERN = _pm_(first_name: /^J/, age: 18..)

# ...and then
if value in MY_PATTERN ...

# ...and, consequently,
people.grep(_pm_(first_name: /^J/, age: 18..))

(I am marking the dreamed-of PM constructor as ugly _pm_ here to underline it is not a ready proposal, but "something should be here")

#12 - 12/03/2021 02:04 PM - Dan0042 (Daniel DeLorme)
zverok (Victor Shepelev) wrote in #note-11:

TBH, since pattern matching inception I hope for some way of putting patterns into values

Close enough?

MY_PATTERN = proc { _i in { name: /^B/, age: 18.. } }
people = [{:name=>"Jim", :age=>18}, {:name=>"Bob", :age=>40}]
people[0] in MY_PATTERN #=> false
people[1] in MY_PATTERN #=> true
people.grep(MY_PATTERN) #=> [{:name=>"Bob", :age=>40}]

#13 - 12/03/2021 02:23 PM - zverok (Victor Shepelev)

Close enough?

Obviously :) But still a "hack". E.g. it is not a "value representing the pattern", this way we can talk ourselves into "we didn't need PM at all, we always could

MY_PATTERN = -> { _i[:name] =~ /^B/ && _i[:age] > 18 }

# ...and
case foo
when -> { _i[:name] =~ /^B/ && _i[:age] > 18 }

:shrug: