Should `expr in a, b, c` be allowed or not?

09/26/2019 08:04 AM - mame (Yusuke Endoh)

<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>matz (Yukihiro Matsumoto)</td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

In [#15865](https://gitlab.com/ruby/ruby/-/issues/15865), a new syntax `<expr> in <pattern>` was introduced. By using this, we can write:

```ruby
json = { foo: 1, bar: 2 }
if json in { foo:, bar: }
  p [foo, bar] #=> [1, 2]
end
```

However, we cannot write:

```ruby
p(json in { foo:, bar: }) #=> expected: true, actual: syntax error
```

This is because `<expr> in <pattern>` is an expression but not an argument. For example, `foo(json in a, b, c)` is ambiguous: it is considered `foo((json in a), b, c)` and `foo((json in a, b, c))`.

What should we do?

1. Do nothing; we admit that it is a spec
2. Revert the feature
3. Disallow a pattern like `a, b, c` or `a:, b:, c:` in this one-line pattern matching syntax; we ask a user to write `json in [a, b, c]` or `json in {a:, b:, c:}`

**Related issues:**

Related to Ruby master - Feature #15865: `<expr> in <pattern>` expression  Closed

**Associated revisions**

Revision 6e70fa49 - 11/10/2019 01:34 PM - ktsj (Kazuki Tsujimoto)

Disallow omission of parentheses/brackets in single line pattern matching [Feature #16182]

**History**

#1 - 10/01/2019 01:36 PM - shevegen (Robert A. Heiler)

I can not comment/answer on the issue and questions; I think this is for matz and the core team to decide either way, whatever the way.

I did, however had, want to add that:

```ruby
json in [a:, b:, c:]?
```

is quite difficult to read (for me). So even if this may not be an ideal explanation, but ... I would not be at all opposed to disallowing that, merely syntax-wise alone. ;-)  

(I do not really have a big opinion on the functionality in general but ideally my personal taste is to prefer simpler syntax, whenever that is possible. We have in general quite some suggestions that combine a lot of complex syntax together, which I think is not ideal, in general; also in other proposals.)

#2 - 10/02/2019 03:24 AM - baweaver (Brandon Weaver)

I wonder if it would make sense to reverse this to be left-to-right (LTR) rather than right-to-left (RTL) to make it easier to parse.

I cannot think of another RTL syntax in Ruby at the moment, including the current for ... in statement:

```ruby
for item in collection
```
A full example might be:

```ruby
for a, b in { a: 1, b: 2 }
  p a, b
end
:a
1
:b
2
=> {:a=>1, :b=>2}
```

Of course this does not currently work with keyword arguments:

```ruby
[2] pry(main)> for a: 1, b: 2 in [{ a: 1 }, { b: 2 }]
SyntaxError: unexpected ':', expecting '.' or &. or :: or ']
for a: 1, b: 2 in [{ a: 1 }, { b: 2...`
```

What if we leveraged some of the current logic for parsing a for ... in statement to make single-line pattern matching into a LTR syntax? This may be a solution for the parsing difficulties, as well as build on the intuition of Ruby developers expecting LTR syntaxes naturally.

#3 - 10/02/2019 08:05 AM - matz (Yukihiro Matsumoto)
I vote for 3 in the OP.

Matz.

#4 - 11/10/2019 01:39 PM - ktsj (Kazuki Tsujimoto)
- Status changed from Open to Closed

Applied in changeset git|6e70fa49b111e2a2839297b057a3df354cae265a.

Disallow omission of parentheses/brackets in single line pattern matching [Feature #16182]

#5 - 11/10/2019 01:40 PM - ktsj (Kazuki Tsujimoto)
- Status changed from Closed to Open

#6 - 11/10/2019 01:41 PM - ktsj (Kazuki Tsujimoto)
- Related to Feature #15865: `<expr> in <pattern>` expression added