# Ruby master - Feature #16670

## Reverse order of `expression` in `pattern` for 1-line pattern matching while it's still experimental

03/03/2020 03:14 PM - ttilberg (Tim Tilberg)

| | |
|---|---|
| **Status:** | Closed |
| **Priority:** | Normal |
| **Assignee:** | |
| **Target version:** | |

**Description**

Currently the 1-line syntax for pattern matching is:

```
# Usage: <expression> in <pattern>

expression = {
  pattern: "Example"
}

expression in {pattern: something}
# something => "Example"
```

Is it technically possible, and desirable to switch the order of this syntax to:

```
# Usage: <pattern> in <expression>

expression = {
  pattern: "Example"
}

{pattern: something} in expression
# something => "Example"
```

?

Here are my reasons:

- It is more intuitive in English -- we are "finding a pattern in something". Finding "something in a pattern" doesn't seem to make sense.
- Assignment is happening, and this keeps assignment on the left side of the operator which feels more natural.
- It matches existing behavior with the workings of the case statement:

Understanding that a case block evaluates each when expression using when_expression === case_expression makes more consistency with when_pattern in case_pattern using the new operator.

```
case something
when /pattern/
end

# is equivalent to

/pattern/ === something

# This creates more parity with

case something
in {pattern: x}

# would be equivalent to

{pattern: x} in something
```

Please see the following discussion on Reddit:
https://www.reddit.com/r/ruby/comments/favshb/27s_pattern_matching_official_docs_recently_merged/fj2c7ng/

## History

#### #1 - 03/03/2020 03:31 PM - shevegen (Robert A. Heiler)

I do not think reddit discussions are that useful, largely because it is difficult to find what is really going on (content-to-noise ratio), and the article title was actually "2.7's pattern matching official docs (recently merged)". zverok is active here on the bugtracker, though, so I suppose he can comment on this issue if he has time and is motivated. :)

If I understood your comment, then you refer mostly to the " in " part, and not regular pattern-matching via case/in, yes?

If this is the case, and if I do not remember incorrectly, then I think the one-line " in " was suggested by mame, not by the original author who suggested pattern matching. (I mention this just so there is no confusion, because we should be clear with this - ruby users may be confused about which syntax is valid, and which one is not, even more so when it would suddenly change.)

Personally I have no specific opinion either way, largely because I am sticking to what is very simple for me to understand; and pattern matching, while interesting, is way over my head. :D But specifically, it may be best if mame and zverok could comment on the proposal, if possible, to compare trade-offs, in particular upon this:

```
expression in {pattern: something}
```

versus

```
{pattern: something} in expression.
```

#### #2 - 03/03/2020 04:15 PM - zverok (Victor Shepelev)

In favor of the proposal, standalone match looks this way in other languages:

```
# Rust:
let (x, y, z) = (1, 2, 3);
# Erlang:
{X, Y} = {1, 2}.
# Elixir:
{a, b, c} = {:hello, "world", 42}
# F#:
let {First=first} = alice
# Swift (?):
case let Media.movie(title, _, _) = m
```

(Probably there could be more examples, though I am not super-familiar with many languages, and for what I can judge from online docs, lot of them -- like Haskell or OCaml -- seem not to have standalone let expression, just match with several alternative patterns)

#### #3 - 03/03/2020 04:39 PM - decuplet (Nikita Shilnikov)

For the record, both Haskell and OCaml have let expressions. And yes, both have patterns on the left side, just like in F# or Rust.

#### #4 - 03/03/2020 04:49 PM - zverok (Victor Shepelev)

....and also, one more example, now just in Ruby:

```
# that's what we always had:
a, b = [1, 2, 3]
# could be pretty complicated:
a, (b, c), *d = [1, [2, 3], 4, 5]

# ...but now let's imagine we need to unpack a simple Hash...
# ...that's how it will look:
{a: 1, b: 2, c: 3} in {a:, **rest}
# ...while "intutively" I always want it look as close array examples above as possible:
{a:, **rest} in {a: 1, b: 2, c: 3}
```

I believe it should've been discussed on "standalone in" introduction ticket (and probably it was found that the "natural" order will be impossible to parse?), but I don't see detailed discussion here: #15865

#### #5 - 03/04/2020 04:16 AM - nobu (Nobuyoshi Nakada)

The pattern syntax looks very close to hash literals, but its semantics is quite different.
So I think it is very hard to distinguish them without a preceding mark (in keyword for now).

#### #6 - 03/04/2020 12:43 PM - decuplet (Nikita Shilnikov)

I think it's important to realize that it's the goal of pattern matching to be similar to data constructors. It proved itself to be a good solution in the long run, this is why languages mentioned above chose this approach. Of course, I'm not taking into consideration implementation difficulties here.

#### #7 - 04/16/2020 02:07 PM - Dan0042 (Daniel DeLorme)

I think the in syntax and order feels natural when you're actually matching a pattern. It's not AI-like pattern recognition like "find a pattern in this" but rather pattern correspondence like "is this in the range expressed by this pattern".

But when using it only as destructuring assigment it does feel that something is off. Maybe it's because expr in var has the opposite order of for var in expr.

So rather than reversing the order, I'd like to tentatively propose ~|> as a more natural-feeling alias for rightward destructuring assignment. Full proposal at #16794

#### #8 - 10/29/2020 04:23 PM - ttilberg (Tim Tilberg)

There were notes in the recent dev meeting that affect this issue:
https://github.com/ruby/dev-meeting-log/blob/master/DevelopersMeeting20201026Japan.md#feature-17260-promote-pattern-matching-to-official-feature-ktsj

=> is a new feature to support right hand assignment to allow things like do_stuff.then{|things| make_things things } => hotdogs.

1-line pattern matching is currently "expression" in pattern. Being very similar to right-hand assignment, it seems that => will now act as 1-line pattern matching, removing expr in pattern. I find this attractive, as it reduces the number of concepts being added, and makes the 1-line matching more intuitive. It removes the interpretation of the word in, and shares the existing notions of rescue HotDogError => e and {this => that}.

Given this new direction, this issue can be closed.

Discussion:

- https://github.com/ruby/dev-meeting-log/blob/master/DevelopersMeeting20201026Japan.md#feature-17260-promote-pattern-matching-to-official-feature-ktsj
- https://bugs.ruby-lang.org/issues/17260

#### #9 - 10/29/2020 04:28 PM - jeremyevans0 (Jeremy Evans)

*- Status changed from Open to Closed*