There are the following differences between case ... when and case ... in. Is this an expected behavior?

```ruby
% ruby -v
ruby 3.1.0dev (2021-05-28T16:34:27Z master e56ba6231f) [x86_64-darwin19]
% ruby -ce 'case expression when 42; end'
Syntax OK
% ruby -ce 'case expression in 42; end'
-e:1: warning: One-line pattern matching is experimental, and the behavior may change in future versions of Ruby!
-e:1: syntax error, unexpected `end', expecting `when'
case expression in 42; end
```

So, I have two concerns.

- Since the pattern matching syntax is different from case ... when, I can't user write semicolon one-line case ... in in the same semicolon one-line as case ... when?
- Does case expression in 42; end display an experimental warning of one-line pattern matching. Right?

This is reproduced in Ruby 3.1.0-dev and Ruby 3.0.1.

### Additional Information

**NOTE 1:** I understand that only syntax that doesn't use case and end is experimental one-line pattern matching syntax.

```ruby
% ruby -ce 'expression in 42'
-e:1: warning: One-line pattern matching is experimental, and the behavior may change in future versions of Ruby!
Syntax OK
```

**NOTE 2:** The syntax is OK if a semicolon is used between expression and in. But case ... when is a valid syntax to omit.

```ruby
% ruby -e ruby -ce 'case expression; in 42; end'
Syntax OK
```

### History

**#1 - 05/30/2021 08:41 AM - koic (Koichi ITO)**

NOTE 1: I understand that only syntax that doesn't use case and end is experimental one-line pattern matching syntax.

A little supplement. The following is also an experimental one-line pattern matching syntax since Ruby 3.0, but the => one-line pattern matching syntax is no problem.

```ruby
% ruby -e "*** => ***
-e:1: warning: One-line pattern matching is experimental, and the behavior may change in future versions of Ru
```
It is a known restriction that the code is parsed as case (expression in 42). Let me consider if it is possible to make that a priority.