In Ruby, when a case-when statement is written, the when branches accept expressions which will be evaluated to objects, then `===` is called to check if any of them returns true:

```ruby
case 'a'
  when 'abc'
    # not matched
  when Regexp.new('[abc]')
    puts :matched # => matched
end
```

To demonstrate what is being done here, this is a mock:

```ruby
equal_all_mock = Object.new
class << equal_all_mock
  def ==(anything) true end
end
# 1
case 'a'
  when equal_all_mock
    puts :matched # => matched
end
# 2
if equal_all_mock == 'a'
  puts :matched # => matched
end
```

Often times when matching for conditional statements, they have values in addition to being truthy or falsey; for example, it is very tempting to write (bugged) code like this (context: parsing 2D robot path instructions):

```ruby
case when i = '^v<>'.index(code)
  x += [0, 0, -1, 1][i]
  y += [1, -1, 0, 0][i]
when code = '/\'[code]
  if code == '/'
    dx, dy = dy, dx
  else
    dx, dy = -dy, -dx
  end
when code == '#'
  dx = -dx
  dy = -dy
end
```

This pattern has problems:

1. Using assignment to capture expressions "leaks" the local variable into the current scope, which the case block doesn't lock into a block scope, as it's not a proc

2. Even if the match fails, the expression is still written; code = '/\'[code] in this case may assign nil, of which then code == '#' will fail

3. The alternative would be using regex, such as `/^v<>/` and then using `$&` to fetch match data... but the global variable pattern is said to be discouraged, and while it works in this specific case it doesn't work in others, like if I want to act upon the index of an
array search (but not when the search result is nil)

Thus my proposal:

case
  when /^v<>/.index(code) => i
    x += [0, 0, -1, 1][i]
    y += [1, -1, 0, 0][i]
  when /\[/[code] => code
    if code == '/'
      dx, dy = dy, dx
    else
      dx, dy = -dy, -dx
    end
  when code == '#'
    dx = -dx
    dy = -dy
end

This is based on the rescue Exception => e syntax. The when expression => i format could potentially even be extended to:

case 'foobar'
  when /fo./ => match
    p match # => foo
end

or with a proc that accepts 0~1 parameters (if it expects one, ruby could feed in the truthy value):

case
  when /^v<>/.index[code] do |i|
    x += [0, 0, -1, 1][i]
    y += [1, -1, 0, 0][i]
  end
  when /\[/[code] do |code|
    if code == '/'
      dx, dy = dy, dx
    else
      dx, dy = -dy, -dx
    end
  end
  when code == '#'
    dx = -dx
    dy = -dy
end

While some cases like these could be replaced by if-else statements I feel like this would be much better as an enhancement on the pattern-matching side. Scala, for example, does have case x if x % 15 == 0 => { statements } in its pattern-matching; handy when writing fizzbuzz.

Related issues:
Is duplicate of Ruby master - Feature #14912: Introduce pattern matching syntax
Closed

History
#1 - 05/01/2019 04:25 AM - unihedron (Unihedron 0)
  - Description updated

I messed up brackets.

#2 - 05/01/2019 04:31 AM - tad (Tadashi Saito)
What is the relationship with #14912 that already committed?

#3 - 05/01/2019 04:34 AM - nobu (Nobuyoshi Nakada)
  - Description updated

03/14/2022
That is the exactly same feature which I proposed yeas ago and was rejected.

#4 - 05/01/2019 08:03 AM - uniiedad (Uniiedad 0)
   #14912 is more thought-out and seems to have made good progress, my regret is I didn't come upon it when trying to search for duplicate issues. #14709 seems to have a collection of case studies and really interesting discussions, but "if we were going to add pattern matching in Ruby, we should add it with better syntax" which #14912 seems to address. Unfortunately I can't seem to close this ticket, even after having written this.

#5 - 05/01/2019 08:21 AM - duerst (Martin Dürst)
   - Is duplicate of Feature #14912: Introduce pattern matching syntax added

#6 - 05/01/2019 08:21 AM - duerst (Martin Dürst)
   - Status changed from Open to Closed