Mixing kwargs with optional parameters changes way method parameters are parsed

I have the following method:

def test(first_param = nil, keyword_arg: nil)
  puts "first_param: #{first_param}"
  puts "keyword_arg: #{keyword_arg}"
end

All the following calls do what I expect them to do:

test(:something)
#=> first_param: something
#   keyword_arg:

test(nil, keyword_arg: :keyword_arg)
#=> first_param:
#   keyword_arg: keyword_arg

test({ first_param: :is_a_hash }, keyword_arg: :is_still_working)
#=> first_param: {first_param=>:is_a_hash}
#   keyword_arg: is_still_working

But omitting the optional keyword_arg and passing a hash as first argument gives me an error:

test(first_param: :is_a_hash)
#=> test.rb:1:in `test': unknown keyword: first_param (ArgumentError)
# from test.rb:12:in `<main>'

I'd expect it to set first_param to { first_param: :is_hash } and keyword_arg being nil.

It seems it is interpreting every hash as keyword arg:

test(keyword_arg: :should_be_first_param)
#=> first_param:
#   keyword_arg: should_be_first_param

This should have set first_param to { keyword_arg: :should_be_first_param }, leaving keyword_arg nil in my opinion.

Making the first parameter mandatory and everything works like I'd expect to:

def test(first_param, keyword_arg: nil)
  puts "first_param: #{first_param}"
  puts "keyword_arg: #{keyword_arg}"
end

test(first_param: :is_a_hash)
#=> first_param: {:first_param=>:is_a_hash}
#   keyword_arg:

test(keyword_arg: :should_be_first_param)
#=> first_param: {:keyword_arg=>:should_be_first_param}
#=> keyword_arg:
I'd expect making a parameter optional does not change the way parameters are parsed.

Is this a parser oddity or expected behaviour?

Related issues:
Related to Ruby master - Feature #14183: "Real" keyword argument

History

#1 - 01/07/2016 02:36 PM - markus_d (Markus Doits)
- Subject changed from Mixing kwargs with optional parameters change way method parameters are parsed to Mixing kwargs with optional parameters changes way method parameters are parsed

#2 - 01/08/2016 01:21 AM - duerst (Martin Dürst)
- Assignee set to matz (Yukihiro Matsumoto)

This looks like a spec issue, so I have assigned it to Matz. I can see arguments for both ways, the current one and the one proposed by Markus. But we can't have it both ways :-(.

#3 - 01/09/2016 07:20 PM - marcandre (Marc-Andre Lafortune)
- Status changed from Open to Rejected

This behavior may be surprising but it is intentional.

It boils down to giving priority to filling keyword arguments first instead of filling unnamed parameters. It is actually the only possible way to go. Among other things, think about the following example:

```ruby
def foo(*rest, bar: 42)
end
```

If we don't prioritize named arguments first, then there is simply no way to specify a value for bar in this example!

So Ruby checks that:

- after all mandatory unnamed arguments are filled
- if the last remaining argument is hash-like
- and all its keys are symbols
- and the method called uses keyword arguments => then that parameter is used for keyword arguments.

Note the requirement on keys being symbols. This can yield even more surprising if you pass a hash with some keys that are not symbols:

```ruby
def foo(a = nil, b: nil)
  p a, b
end
foo(:b => 42) #=> nil, 42
foo('b' => 42) #=> {'b' => 42}, nil
```

#4 - 12/14/2017 07:24 AM - hsbt (Hiroshi SHIBATA)
- Related to Feature #14183: "Real" keyword argument added