As a developer, I mainly program in two languages: Swift and Ruby.

I love Ruby with all my heart, but I came to like a particular feature of Swift: Argument labels; I would love to see them appear in Ruby.

Matz on Twitter suggested I post a feature request, so here we are!

Context

Naming is hard. Keyword arguments helps defining methods that are readable and allow for flexibility ordering arguments, but the result may be a little too verbose as I use to create methods like:

```ruby
def change_color_for_user_until_date(new_color:, user:, end_date:)
  do_something_with(new_color)
  do_something_else_with(user, end_date)
end
```

Also, that's not as readable in a "natural way" as plenty of code we can produce with Ruby.

Copying Swift, I would love to have argument labels, aka the option to define a different name for the argument and its related parameter.

Right now

This code is a no-go, as it would be a nightmare to maintain:

(the code below is using reserved keywords so would not work in reality, but it's for the sake of clarity and the sake of the example)

```ruby
def change_color(to:, for:, until:)
  new_color, user, end_date = to, for, until

  do_something_with(to)
  do_something_else_with(for, until)  # What does this do with which data again?
end
```

This being said, I can simulate such a behaviour this way:

(the code below is using reserved keywords so would not work in reality, but it's for the sake of clarity and the sake of the example)

```ruby
def change_color(to:, for:, until:)
  new_color, user, end_date = to, for, until
  # well, those are reserved keywords, I would never be able to use them like that

  do_something_with(new_color)
  do_something_else_with(user, end_date)
end
```

That's not perfect though:
Not standardized enough
- Conflict with reserved keywords (in reality, I would try to find synonyms? change_color(with: :blue, regarding: user, up_until: DateTime.tomorrow) ?)
- An extra line, and not so elegant

The feature request mirroring Swift

It does not have to be implemented exactly this way, of course!

```ruby
def change_color(to new_color:, for user:, until end_date:)
  do_something_with(new_color)
  do_something_else_with(user, end_date)
# No use of reserved keywords anymore, and readable variable name!
end

change_color(to: :blue, for: user, until: DateTime.tomorrow)
```

Thanks in advance for your insight on this matter,
          Cheers!

Related issues:
Related to Ruby master - Feature #17785: Allow named parameters to be keywords

History

#1 - 12/11/2021 01:33 AM - Dan0042 (Daniel DeLorme)

- tentative +1
  if/for/until are useful as keyword arguments but almost impossible to use as local variables.  
  But the suggested syntax makes it look to me like new_color is the keyword argument rather than the local alias.
  I'd like to suggest re-using rightward assignment syntax:

```ruby
def change_color(to: => new_color, for: @current_user => user, until: DateTime.tomorrow => end_date)
```

#2 - 12/11/2021 04:37 AM - Lomig (Lomig Enfroy)

Oh, I think the rightward assignment idea is brilliant; it would make sense and be kind of consistent syntactically speaking!

#3 - 12/11/2021 09:05 AM - byroot (Jean Boussier)

Sounds quite similar to [Feature #17785].

The current working solution is:

```ruby
def change_color(to:, for:, until:)
  do_something_with(to)
  do_something_else_with(binding.local_variable_get(:for), binding.local_variable_get(:until))
end
```

A few new solutions were proposed in that ticket.

#4 - 12/11/2021 09:06 AM - byroot (Jean Boussier)

- Related to Feature #17785: Allow named parameters to be keywords added

#5 - 12/13/2021 08:37 AM - Lomig (Lomig Enfroy)

Well, using keywords is only a bonus side effect of arguments labels.

The main idea / need is to use different names for variables/arguments when calling the method and inside the method itself.

Thanks to you and this link, I was able to find that this feature request is indeed a duplicate — https://bugs.ruby-lang.org/issues/16460 that I did not find when I created mine. (I don't know how to tag a request as duplicate)

#6 - 12/16/2021 05:46 AM - duerst (Martin Dürst)

Lomig (Lomig Enfroy) wrote in #note-5:

  The main idea / need is to use different names for variables/arguments when calling the method and inside the method itself.
Does this mean that both names can be used inside the method? And in that case, would they alias the same variable, or would there be two variables? Or can only one of the names be used inside the method, and only the other outside?

#7 - 12/20/2021 09:08 PM - Lomig (Lomig Enfroy)
duerst (Martin Dürst) wrote in #note-6:

Does this mean that both names can be used inside the method? And in that case, would they alias the same variable, or would there be two variables? Or can only one of the names be used inside the method, and only the other outside?

The outer name is the only one we can use externally, and the inner name the only one we can use inside the method, and they represent the exact same data.

#8 - 12/29/2021 10:50 PM - janosch-x (Janosch Müller)
i really like this in Swift so i’ll try to give another example for how this addition might improve both readability and conciseness.

currently there are two approaches: using verbose names for keyword arguments or short ones. both have disadvantages that could be overcome with the proposed feature.

current approach 1: spelling it all out

this makes both the invocation as well as the body of the method verbose, and can make them grammatically weird or misleading to read.

```ruby
def extract_bad_records(from_relation:)
  # lots of code referring to `from_relation`, e.g.
  # from_relation.select do ... 
end
```

current approach 2: keeping it short

this makes the method nice to use, but makes it hard to read the method body itself.

```ruby
def extract_bad_records(from:)
  # lots of code referring to `from`, e.g.
  # from.select do ... 
end
```

with labelled arguments

makes the method nice to use AND read.

```ruby
def extract_bad_records(from => relation:)
  # lots of code referring to relation, e.g.
  # relation.select do ... 
end
```

of course this example is a bit artificial as one would usually make a single or primary argument a positional argument instead of a kwarg, but you get the idea. if you know the struggle between too verbose and too non-descript kwarg names, i think you want this feature :)

#9 - 01/31/2022 06:14 PM - pabloh (Pablo Herrero)
This is a cool idea but I'm not sure about the syntax, if we eventually want Ruby to have pattern matching at the method definition level like Elixir has, using rightward assignment for this feature could potentially create a conflict in the future.